

# Main Page

From Omni-bot Wiki

**Welcome to the Omni-bot Wiki,  
a place to find comprehensive information about Omni-bot.**

## Latest Omni-bot Release

Omni-bot 0.7 Enemy Territory  
Download Omni-bot 0.7  
Windows  
Linux

## About Omni-bot

Intro to Omni-bot  
Omni-bot Releases  
Supported Games  
Enemy Territory •  
Doom 3 • Quake 4  
Fortress Forever •  
RTCW  
Team Fortress 2  
Development Logs

## Getting started

Installing Omni-bot  
Configuring Omni-bot  
FAQ  
Download PDF

## Customizing Omni-bot

Omni-bot Waypointing  
Omni-bot Map Scripting  
Omni-bot Map Goals  
Omni-bot Scripting  
Omni-bot Weapon Scripting  
Omni-bot Routing  
Omni-bot Script Goals  
Omni-bot Community Scripts

## Reference

Omni-bot Command Reference  
Omni-bot Script Reference  
Omni-bot F.A.Q.  
General Bot F.A.Q.  
Omni-bot Community Portal

## Useful Tools

SciTE  
PSPad  
Notepad++  
ET Minimizer

Retrieved from "http://omni-bot.com/wiki/index.php?title=Main\_Page"

This page has been accessed 15,952 times. This page was last modified 02:48, 26 August 2008.

# Omni-bot 0.7

From Omni-bot Wiki

## Contents

- 1 Omni-bot 0.7 Changelog
- 2 Updates
  - 2.1 Bot
  - 2.2 Enemy Territory
  - 2.3 Scripts
  - 2.4 Waypoints
- 3 Additional Info
  - 3.1 New Goal System
  - 3.2 New Script Goal System
  - 3.3 Updating Map Scripts from 0.66 to 0.7
  - 3.4 Updating Script goals from 0.66 to 0.7

## Omni-bot 0.7 Changelog

### Updates

#### Bot

- Completely new bot goal system - see below
- New trigger system to allow box or sphere triggers to be set up to call script functions when certain entities enter/exit.
- Updated linux builds to libstdc++ .6 to fix memory leaks that occur with libstdc++ .5
- Debug window to view much useful information - bot down
- Ability to group all goal types. see SetGoalGroup
- Stance property for attack, defend, and snipe goals
- More bug fixes than I can count

#### Enemy Territory

- Added XP accessor, see GetEntityStat
- Added a bunch of new triggers that can be used in map scripts, simplified their naming.
  - Trigger for mover states opened, closed, opening, closing (team doors, additional lever support, etc)

- 30 seconds left trigger
  - 2 minutes left trigger
  - Round End trigger
- Added additional entities to breakable / shootable category ( chairs, etc )
- Added seperate goals for Panzer and Flamethrower camp locations
- Added Mobile Mortar goal
- Bots will commit suicide 2 seconds before next spawn if a class or weapon change request has been given rather than immediatly
- Fixed invalid characters from being used in goal names so it doesn't cause script errors.
- Simplified names for map goals. This effects the naming used in scripts. Apologies for breaking existing scripts. This is why we went through the scripts in our download section and fixed them.
  - MAP\_CONSTRUCTION is now BUILD
  - MAP\_DYNAMITE\_TARGET is now PLANT
  - MAP\_DYNAMITE is now DEFUSE
  - MAP\_FALLEN\_TEAMMATE is now REVIVE
  - MAP\_MOVER is now MOVER
  - MAP\_MOUNTABLE\_MG42 is now MOUNTMG42
  - MAP\_REPAIR\_MG42 is now REPAIRMG42
  - MAP\_MOBILE\_MG42\_SPOT is now MOBILEMG42
  - MAP\_MOBILE\_MORTAR\_SPOT is now MOBILEMORTAR
  - MAP\_ARTYSPOT is now CALLARTILLERY
  - MAP\_ARTYTARGET\_S is now ARTILLERY\_S
  - MAP\_ARTYTARGET\_D is now ARTILLERY\_D
  - PLANT\_MINE\_GOAL is now PLANTMINE
  - HEALTH\_CABINET is now HEALTHCAB
  - AMMO\_CABINET is now AMMOCAB
  - MAP\_FLAG is now FLAG
  - MAP\_CAPPOINT is now CAPPOINT
  - MAP\_CAPHOLD is now CAPHOLD
  - MAP\_FLAG\_RETURN is now FLAGRETURN
- See below for a more complete guide to updating 0.66 map scripts to 0.7

## Scripts

- Completely new script goal system - see below
- Moved MinBots, MaxBot functionality completely into script, added BalanceTeams functionality to it. See [global\\_scripts/server\\_manager.gm](#)

## Waypoints

- Updated a metric shitload of waypoints and scripts to new goal naming conventions.

# Additional Info

## New Goal System

- Overhauled goal system into a hierarchical behavior tree, based on Halo2 Hierarchical Behavior Tree Gamasutra Halo2 AI Article
- Much more optimized, extensible, and scalable.

## New Script Goal System

- Script goals can be added by simply adding them to the goals folder.
- Script goals compete directly with internal goals, allowing them to be balanced much easier along side other goals.
- Script goals hide much functionality, allowing complex scripts to be written with much less code and much less cpu usage.
- No longer requires edits to def\_bot.gm or any other script to load them.
- Much more automatic handling of bot control by individual script goals.
  - SetScriptControlled, SetScriptControlledWeapon not necessary anymore, gone.
  - No need for a bunch of special logic to prevent leaving bot in a bad state.
- See Omni-bot Script Goals

## Updating Map Scripts from 0.66 to 0.7

Due to the sheer amount of changes from 0.66 to 0.7, unfortunately the structure of many things changed such that we could not maintain compatibility with 0.66 map scripts(though the waypoints should still be usable). Here is an overview of what is necessary to update a map script to 0.7

- Rename all references to map goals in the map script using the new naming convention outlined above.
- Remove all calls to SetBiasGoals, it has been removed, and replaced with SetGoalPriority, see Map Scripting Enemy Territory
- Remove all calls to bot.SetGoalProperty, it has been removed.
- Util.ResetGoals has been removed since the new goal system over-rides current goals with higher priority goals
- The 'Go' nav flag and goal have been removed, so any references to them in map scripts should be removed
- Any custom bot control threads need to be replaced with the new script goal system ( i.e. WatchForElevator threads that take control of a bot)

## Updating Script goals from 0.66 to 0.7

The old method of writing scripts that would go into the scripts folder, and then executed by adding a few lines into `def_bot.gm` is completely deprecated, unsupported, and should not be used. In 0.7 all bot controlling functionality is relocated into script goals.

It is highly recommended that you run 0.7 with a completely fresh installation, and not one that retains a bunch of old scripts that can cause problems.

See the new script goals located in the `scripts/goals` folder for reference, and see the Omni-bot Script Goals for additional information.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.7](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.7)"

This page has been accessed 481 times. This page was last modified 06:31, 25 August 2008.

# Intro to Omni-bot

From Omni-bot Wiki

## Basic Omni-bot Info

Omni-bot is an artificial intelligence(AI) controlled opponent for first person shooter games. Unlike many bots that are written for specific games, Omni-bot was designed from the beginning to exist mostly as a generic framework with which support for many different games can be made, and most of the functionality can be shared among them.

See Supported Games for a current list of games that Omni-bot supports.

## Omni-bot Developers

- DrEvil - Primary Developer.
- Magik - Started Enemy Territory support, hosts the website and code repository.
- Geekfeststarter - Testing, Waypointing, Scripting.
- Crapshoot - Testing, Waypointing, Scripting, Documentation.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Intro\\_to\\_Omni-bot](http://omni-bot.com/wiki/index.php?title=Intro_to_Omni-bot)"

This page has been accessed 1,249 times. This page was last modified 20:45, 15 September 2007.

# Omni-bot Releases

From Omni-bot Wiki

## Omni-bot Releases

- Omni-bot 0.7
- Omni-bot 0.66
- Omni-bot 0.65
- Omni-bot 0.61
- Omni-bot 0.6
- Omni-bot 0.532
- Omni-bot 0.531
- Omni-bot 0.53
- Omni-bot 0.52
- Omni-bot 0.52 beta 6
- Omni-bot 0.52 beta 5
- Omni-bot 0.52 beta 4
- Omni-bot 0.52 beta 3
- Omni-bot 0.52 beta 2
- Omni-bot 0.51
- Omni-bot 0.4
- Omni-bot 0.31
- Omni-bot 0.3
- Omni-bot 0.231
- Omni-bot 0.23
- Omni-bot 0.22
- Omni-bot 0.21
- Omni-bot 0.2

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Releases](http://omni-bot.com/wiki/index.php?title=Omni-bot_Releases)"

This page has been accessed 855 times. This page was last modified 19:41, 24 August 2008.

# Supported Games

From Omni-bot Wiki

## Contents

- 1 Enemy Territory
- 2 Fortress Forever
- 3 Quake 4
- 4 Doom 3
- 5 Want to support Omni-bot?

## Enemy Territory

Enemy Territory is a free for download team based game that currently is the most advanced in terms of bot support as it has been focused on more than other games.

When you install Omni-bot for ET, it will create a mod called omnibot, which is Omni-bot implemented on top of the default multi-player game type.

In addition to vanilla etmain, Omni-bot has been integrated into many of the popular ET mods, which contain enhancements, fixes, additional weapons, etc on top of the ET game type.

- etpub
  - 0.8.1 -> 0.66 (included)
- jaymod
  - 2.1.2 -> 0.60 - 0.61
  - 2.1.4 -> 0.61
  - 2.1.5 -> 0.65
  - 2.1.6 -> 0.66
  - 2.1.7 -> 0.66
- noquarter
  - 1.1.1 -> 0.6.0 - 0.6.1

## Fortress Forever

Fortress Forever is a Team Fortress modification for Half-life 2. Omni-bot will come with the default installation for training and scrimmage purposes, with training scripts targeted to helping those that are new to Team Fortress, as it can be a difficult game to attract new



players.

## Quake 4

Quake 4 by Raven Software has had an alpha Omni-bot release in an early version, and more recently has been getting beta releases on the Omni-bot forum [here](#)

## Doom 3

Doom 3 by id software has recently joined the Omni-bot supported games list. It is being supported currently by Geekfeststarter. There has not been a public release yet.

## Want to support Omni-bot?

Any mods that would like to add support for Omni-bot should PM DrEvil for information. Porting to a new mod is typically pretty easy, particularly if the mod is built on an already supported engine.

Omni-bot support is added to a mod by adding a lightweight interface file to the mod that compiles into it and communicates with the bot dll. After that, all the bot behavior is handled in the bot dll. If you are a minor variation of an existing supported game chances are your mod can use the same dll, like Jaymod, etpub, NQ.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Supported\\_Games](http://omni-bot.com/wiki/index.php?title=Supported_Games)"

This page has been accessed 1,765 times. This page was last modified 16:15, 14 January 2008.

# Enemy Territory

From Omni-bot Wiki

**Welcome to the Enemy Territory Section of the Omni-bot Wiki**

## Latest Release

Omni-bot 0.66 Enemy Territory

Omni-bot 0.66  
Current Known Bugs

Download Omni-bot 0.66  
Windows  
Linux

## Getting started

Installing Omni-bot  
Configuring Omni-bot  
FAQ

## Waypointing

Tutorial  
Omni-bot Waypointing  
Waypointing Tips  
Vehicle Pathing  
Sniper Spots  
General Tips  
Supported Maps

## Scripting

Map Scripts  
Map Goals  
Bot Scripting  
Routing  
Script Goals  
Community Scripts

## Reference

Omni-bot Command Reference  
Omni-bot Script Reference  
Omni-bot F.A.Q.  
General Bot F.A.Q.  
Omni-bot Community Portal

## Useful Tools

SciTE  
PSPad  
Notepad++  
ET Minimizer

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Enemy\\_Territory](http://omni-bot.com/wiki/index.php?title=Enemy_Territory)"

This page has been accessed 777 times. This page was last modified 02:05, 2 May 2008.

# RTCW:Main page

From Omni-bot Wiki

**Welcome to the RTCW Section of the Omni-bot Wiki**

## Latest Release

Not Yet Released / In Development

## Getting started

Installing Omni-bot  
Configuring Omni-bot  
FAQ

## Waypointing

Tutorial  
Omni-bot Waypointing  
Waypointing Tips  
    Sniper Spots  
Supported Maps

## Scripting

Map Scripts  
Map Goals  
Bot Scripting  
Routing  
Script Goals

## Reference

Omni-bot Command Reference  
Omni-bot Script Reference  
Omni-bot F.A.Q.  
General Bot F.A.Q.  
Omni-bot Community Portal

## Useful Tools

SciTE  
PSPad  
Notepad+ +

Retrieved from "http://omni-bot.com/wiki/index.php?title=RTCW:Main\_page"

This page has been accessed 285 times. This page was last modified 04:33, 1 May 2008.

# Development Logs

From Omni-bot Wiki

## Contents

- 1 Intro
- 2 DrEvil
  - 2.1 April 28 2008
  - 2.2 January 28 2008
  - 2.3 January 7 2008
  - 2.4 October 22 2007
  - 2.5 July 9th 2007
- 3 Crapshoot
  - 3.1 June 1st 2008
  - 3.2 April 19th 2008
  - 3.3 July 9th 2007
- 4 Jaskot
  - 4.1 Feb 1st 2008
  - 4.2 July 13th 2007

## Intro

In case anyone is interested on what is going on between releases, here is a development log page, where dev members(and anyone really), can post about progress or developments they are making in their respective areas.

## DrEvil

### April 28 2008

Progress has been good lately. Jaskot and crapshoot have been feeding me bugs to fix and I've been busting them out pretty quickly. Due to the sheer number of changes in the architecture of the bot, and the fact that all the goals had to be re-implemented, we've effectively had to start over in getting all the stock ET maps and some custom ones set up with good navigation and goals. Goals either work slightly differently, have additional features we want to take advantage of, or there is just a general desire to improve on them from last release.

There is also a nice selection of script goals that will come with the bot, to serve as both

examples of what can be done with script goals, and to provide some nice additional functionality for the bots. The 0.66 method of making script goals is a steaming pile of crap in comparison to the new way to do it.

As of this writing we have the following script goals that will be included with the bot.

- **Airstrike** - Allows maps to configure locations that a team will attempt to perform an airstrike at.
- **AskForAmmo** - Simple script that allows a bot to use the voice macros to request ammo when low on ammo.
- **AskForHealth** - Simple script that allows a bot to use the voice macros to request health when low on health.
- **CombatMovement** - Performs some strafing and combat oriented movement during combat.
- **DeliverSupplies** - Allows Field Ops and Medics to respond to ammo/health voice macros and deliver supplies to the requester.
- **DispenseAmmo** - Updated from 0.66 allows a bot drops ammo shortly after spawn.
- **DispenseHealth** - Updated from 0.66, allows a bot drops health shortly after spawn.
- **EscortVehicle** - Updated from 0.66 with new features. Allows a vehicle to be escorted by multiple bots, each using different escort offsets.
- **Fireteam** - Simple example script that shows how a bot can be scripted to work with fireteams.
- **GrenadeTarget** - Similar to Airstrike, this goal allows bots to throw grenades at targets. Currently useful for MG42 nests, allows one to set up a safe place to lob a grenade from.
- **MountVehicle** - Updated from 0.66 vehicle mount goal, for rideable vehicles such as fueldump.
- **RideVehicle** - Updated from 0.66, used to support railgun and similar maps where the bots must ride some sort of automatic tram.
- **SelectWeapons** - New in 0.7, this script allows soldiers to more intelligently change weapons during a map. No mobile mg42 or mortar goals? No problem, they will switch to something else. No more soldiers running around with useless weapons due to lack of goal types(mg42/mortar).
- **SupplySelf** - Gives a Medic or Field Ops the know-how to give himself needed health or ammo.
- **UseSwitch** - Updated from 0.66, generic script supported switches, originally used in railgun, now more widely used in other maps such as battery.

As you can see there is plenty of script examples available in 0.7, and it should be easy to see just how much simpler the scripting is done, especially if you compare some of the 0.66 scripts to their updated counterparts. Often they will have additional functionality, and still manage to be smaller, more optimal scripts.

Special thanks to Jaskot and Crapshoot for their hard work. Most of these scripts have come

from them, as have pretty much all of the navigation and map scripts. Their contributions are invaluable.

## **January 28 2008**

Despite looking for a new apartment on the weekends the past few weeks this weekend was probably one of the most productive bot weekends I've had.

Thanks to the efforts of Jaskot and his uncanny ability to flood me with bugs and requests (and nice and easy little test scripts that show me the problem immediately), this weekend was a good weekend as I was able to knock them out and fix them about as fast as he could give them to me. As with all big refactoring tasks, the brand new goal system is going to inevitably require some tweaks and changes to existing goals to ensure things keep working as expected, but in the end the goals are simpler, more maintainable, and often have more functionality than the goals in the  $\leq 0.66$ . I've finally written a MobileMg42 goal as well, and I plan to take another stab at some decent mortar usage functionality soon too.

Also goal related, nearly all script goals have been ported to the new script goal format as well. This includes givehealth, giveammo, ridevehicle, mountvehicle, escortvehicle. Not only are the resulting scripts simpler than they were in 0.66, weighing in at fewer lines, but they are much more reliable and they can't really leave the bot in a broken state as could happen in 0.66. In addition, I took the opportunity to expand on a few of them, namely the escort and mount script. Each now has the ability to support more than 1 user, so for example in fueldump we can set up 4 or more escorts for the tank, each positioned in different slots around it.

By the end of this week I should have ported every goal that was present in 0.66(I think repair mg42 is the only one left), and a few additional ones, not to mention the existing goals that have been ported often contain better behavior than they previously did.

All in all, pending a good amount of testing and potential bug fixing, we're getting pretty close to being able to get into 'release mode' for 0.7 I guarantee that their behavior is much cooler, and scripting is much easier and more efficient with this version.

## **January 7 2008**

We're almost caught up to 0.66 in terms of supported functionality for 0.7, so once we do some more testing for stability and making sure all the existing maps can be completed by the bots, we should be able to put together a release in the near future. 0.7 is overall a large step up for myself and for scripters. The new goal system that treats native and script goals equally and allows them both to compete fixes the old problem of having script goals blindly override everything.

While we test and make sure everything is good I plan to update the wiki with as much information on 0.7 and 0.7 scripting as I can, including documenting the new script implemented goals that have been ported to the new system. I think you will see easily that things have been much improved over 0.66. Authoring script goals is much simpler, as is debugging what the bot is thinking thanks to the Debug Window.

## **October 22 2007**

I spent the weekend on porting some of the ET goals to the new goal system for the 0.7 release. Overall it is pretty easy and so far I've got the PlantExplosive, MountMg42, and BuildConstruction goals ported and working. We're going to try to work towards a soonish ET update that will get everyone up to date and using the latest goal system, which provides a huge improvement to how scripting and script goals are handled. No more editing def\_bot.gm to add hooks to various other scripts and much easier handling of the lifetime of a goal and how the script goal competes with normal goals. I plan to port the existing script goals to use the new system for the release as well, so ideally we won't lose any functionality between the current release and next one.

There's also been significant progress toward the new navigation system, but that won't be part of the update. I'll post some screens and more info soon about that.

## **July 9th 2007**

I took the week of the 4th of July off. I figured I could get alot of work done on the bot, and we had company. Unfortunately I spent about half of it sick, but the last few days have been pretty productive.

Just to bring everyone up to date, the release of Omni-bot 0.66 is to be the last release before the big plans of overhauling large parts of the bot code. Actually, 0.65 was to be the last release, but bug fix releases can override that, as it did here. That doesn't mean last release, period. It means I've come to a point where I need to spend some time to make large structural changes to the bot code. In this case, it will be the largest refactoring tasks done to date in Omni-bot.

The desire to do this is pushed by the difficulties of some user customization becoming more obvious over the past few releases, performance, the difficulty of expanding functionality in the code itself, and functionality that is lacking in the core of the bot in order to support some types of goals, such as for example, my desire to implement rocket jumping and concussion jumping in Fortress Forever and realizing that my path finding system has never taken into account movement capabilities only certain classes have. This has also been the limiting factor preventing an intuitive implementation of the covert ops class in Enemy Territory using disguise doors.

As a general overview, the major refactoring tasks are as follows.

- **Completely new goal system** - The old one turned out too difficult to expand, debug, and script. **[done]**
- **Completely new script goal system** - Current releases are difficult to write custom goal scripts for. There is no framework for a number of script goals to compete against each other for relevance, and this results in unpredictable and hard to debug scripts to do things like hit switches, ride vehicles, etc. This also includes drop in script goals, no longer adding a couple lines to the `def_bot.gm` in order to use a script goal, just drop the script into the right folder and go. **[done]**
- **Scripting optimizations** - In 0.66 and before, many script goals required expensive looping for aiming the bot, choosing a weapon, checking proximity to a desired destination, etc. Most of this will be event based, more reliable, easier to write, easier to debug, and much more efficient in the new system. **[done]**
- **Better Debugging** - There wasn't much in the way of debugging tools available previously to see what was going on inside the bots *head*. Now there is, as the entire behavior tree can be watched for each bot in the expanded Debug Window. **[done]**
- **Navigation System** - The navigation system needs some large enhancements. It needs to be aware of movement modes needed to follow certain paths in order to support some game features. It also needs a better understanding of switches, ladders, elevators, vehicles, and a better integration with scripting to expand functionality of navigation. This is by far the most complex of the tasks, and likely the last to be taken on, and it may mean an invalidation of all current waypoint files. I may still develop and use my Auto Navigation FloodFill if I can work out some of the remaining issues with it, but the fallback is likely a new waypoint system.

After the Enemy Territory 0.66 release, I have needed to refocus my efforts on Fortress Forever, and am doing so with the first 3 of the above tasks complete, and it is already proving much easier to work with. I want to get the Fortress Forever bots as tight as possible, as well as the bot powered training scripts that we are experimenting with for the first time. In addition, we are working in parallel on trying to get a Doom3 and Quake4 version officially released soon, also using the new systems.

---

## Crapshoot

June 1st 2008

It's been a busy 6 weeks since I posted last. A lot of my time has been invested in getting RTCW to a beta stage. Thirteen maps are fully waypointed and scripted. Two servers are currently running a beta version of the bot. Huge thanks to Guerna for providing one of the test servers. And a big thanks to 420Blunt for `axis_complex` and `mp_sub` waypoints. With some nice feedback from AG3NT, Guerna, and 420Blunt, a few bugs have been fixed, some



features have been added, and some waypoint improvements have been made.

I've spent some time helping with getting Enemy Territory ready for a release. Most of my effort went into a first pass of converting existing community map scripts to 0.7 syntax. Since a lot of improvements have been made, the syntax is quite different for the map scripts. 89 map scripts needed to be updated, so this was no small task. Thankfully d00d has been able to get up to speed rather quickly and he's been working very hard on getting quite a few maps more thoroughly updated than I had time for.

We are hoping to have both games at a releasable state very soon. There are some finishing touches to be added, some script optimization and of course a lot of testing.

## **April 19th 2008**

After an unexpected hiatus, I am back. I have spent the last week working on getting caught up with all that has changed as well as implementing (porting from ET) goals for RTCW. Progress has been good and I am happy to report that Omnibot RTCW bots are fully functional. All objectives can be completed and all weapons are supported. Map support has begun and currently three maps are completely waypointed / scripted. The goal is to get as many maps supported as possible in a timeframe that coincides with the release of 0.7 for ET.

## **July 9th 2007**

Since the 0.66 release for Enemy Territory, my focus shifted to porting Omni-bot to Return to Castle Wolfenstein; the prequel to Enemy Territory. Although the games are similar in style, they have enough of a difference in gameplay that I thought it warranted it's own Omni-bot version. It's also a nice project for me to familiarize myself with the framework as the SDK's are very similar.

Work on the RTCW bot has come along nicely and it is a few minor bugs away from being caught up to the ET bot in terms of 0.7 readiness. Because the popular mods for RTCW are inactive in terms of development, it means some features will need to be added. Some features have been added already and if there is enough of a demand for a particular feature, I'll see about including it.

Porting to 0.66 is still being considered as playability would be realised sooner, but existing and pending changes for 0.7 have me prepared to wait a bit longer for it. So most likely, the focus will shift to Quake 4 since it's release (along with Doom 3) is next on the schedule.

For Quake 4, I'm working on finishing up basic waypoints and then will move on to map scripting. Specifically routing and attack / defend goals for CTF and Deadzone gameplay. The goal is to create a similar gameplay experience to online play.

Enemy Territory is still being worked on. There are still a bunch of things I'd like to do scriptwise to help enhance gameplay. Stopwatch mode in particular is one that I'll most likely focus on as it's something that isn't supported as well as it could be. After some discussions with Red Dwarf about this prior to the 0.65 release, some tools were added to help (like CountClass, CountTeam, and the extended ChangeClass), but it needs to be completed. The good news is that it's something that can be worked on in conjunction with RTCW.

---

## **Jaskot**

**Feb 1st 2008**

Been a while, but a lot has been going on with Omni development. Currently I have been working on ET, getting .66 functionality and beyond working in .7. Using the new goal system, I created a goal to grenade targets, currently mounted or built mg42's, although I will soon add vehicles. Basically you set up a table in the map script calling out the goal you want (MAP\_MOUNTABLE\_MG42\_blah) and an offset from the target that is safe for throwing grenades ( for example: 240, 0, 0 ) and the bot will go and toss grenades at it. I am now in the process of implementing all of the new goals and changed functionality into the official map scripts. In .7 we will be rolling out testmap, which is a way to test if the bots are capable of doing all the objectives in a waypointed and scripted map, as well as a method to capture bugs. We have been using this since .6 to test maps for release and now you will be able to submit files demonstrating bugs you find as well. Testmap is also good for testing out new goals as well. Doom3 is on back burner for now, but all official maps have been waypointed and all code updates are being propagated to it. Quake4 is also on backburner, and it still needs a bunch of official maps waypointed. RTCW was crapshoot's baby and we have been without him for some time. Code is being built against it and most ET updates benefit it as well, so it should be fairly up to date, although we don't have too many waypoint files for it. I have been doing some work for DrEvil on Fortress Forever (waypointing scripting, bug finding, etc). ---

**July 13th 2007**

For ET, I'm looking at our Linux build to upgrade to a newer version of g++.

For Doom 3, I'm working on finishing up the base and the expansion pack wp files. I'm now down to 3 ctf maps (d3xpctf2, d3xpctf3 and d3xpctf4). I have also been doing some experimental work with bot difficulty as currently the bots are insane to battle.

For Quake4, I'm working on finishing up the base dm and the community map wp files. I'm also investigating problems with bot difficulty, but unlike Doom3, the Quake 4 bots are

pathetically easy.

For RTCW, I,m working on wp files for the base maps.

In general, I have been working on some tools to place arrays of wps. I currently have a simple raw system in place, but I need to add logic for terrain and obstacles. DrEvil has implemented a selection box feature, and now groups of wps can be manipulated.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Development\\_Logs](http://omni-bot.com/wiki/index.php?title=Development_Logs)"

This page has been accessed 1,081 times. This page was last modified 02:46, 2 June 2008.

# Installing Omni-bot

From Omni-bot Wiki

## Contents

- 1 Installing Omni-bot
  - 1.1 How to install Omni-bot on Windows
  - 1.2 How to install Omni-bot on Linux
  - 1.3 How to install Omni-bot on 3rd Party Host

## Installing Omni-bot

### How to install Omni-bot on Windows

The windows version of Omni-bot is distributed with an NSIS [1] based installer. On windows, the installer will add the Omni-bot installation folder to your PATH environment variables. This allows supported games to easily find Omni-bot in the centralized installation folder. If the Omni-bot directory doesn't get set up in your path, games may have difficulty finding the bot dlls and fail to load. You will find more "How To's" on the Omni-bot F.A.Q. page.

### How to install Omni-bot on Linux

- Unzip Omni Bot Linux zip to the games root folder. Depending on where the game is installed, you may need root privileges. A typical Enemy Territory installation will result in a file layout similar to this:

```
/usr/local/games/enemy-territory
/usr/local/games/enemy-territory/omni-bot
/usr/local/games/enemy-territory/omnibot
```

The omni-bot folder contains the Omni-bot libraries, scripts and waypoints while the omnibot folder contains the game files.

- Start the game normally and launch the omnibot mod in game or optionally use a command line to launch omnibot. For Enemy Territory, the command line would be similar to this:

```
#et +set fs_game omnibot +set com_hunkmegs 64
```

*note: Omni-bot requires glibc-2.2 or newer to be installed.*

### Enemy Territory Addendum

- After starting the game the first time, the game will create a user folder in the users home

directory. In version 0.65 and above, it is required to set the omnibot\_path cvar for the bot library to load correctly. This needs to be set in the etconfig file located in:

```
/home/<username>/.etwolf/omnibot/profiles/<etusername>/
```

To set the omnibot\_path cvar, add this line to the server.cfg:

```
set omnibot_path = "/usr/local/games/enemy-territory/omni-bot"
```

Or set cvar as start parameter:

```
#et +set fs_game omnibot +set com_hunkmegs 64 +set omnibot_path "/usr/local/games/enemy-territory/omni-bot"
```

Be sure to set the path to exactly where the omni-bot directory is located. Note that if you play the game as the root user (not recommended), the etconfig.cfg file will be in /root/.etwolf/omnibot/profiles/<etusername>/

note: ET-instance needs write permission on waypoint-folder to create vis-files for Omni-bot versions prior to 0.65

## How to install Omni-bot on 3rd Party Host

### Basic Requirements

- Server Host Allows for Bots
- FTP Access

If you are unsure about either of these, ask your server provider or read the server providers usage agreement.

### Installation

If you do not know what operating system the server is running, you will need to ask the server provider or get your installation-path via CVAR 'fs\_basepath'. Windows and Linux use different file types. CVAR 'arch' or 'version' containing info which OS is used. Be aware Windows uses left-slashes '\' and Linux/UNIX based-systems use right-slashes '/' as separator-char.

Depending on the server provider, you may or not be able to unzip files once they are on the server. If you are not sure, ask your server provider. If you cannot unzip the files remotely, you will need to upload the files after extracting them locally. The omnibot folder needs to be placed in the games root folder, while the omni-bot folder can be placed anywhere. If you only plan on running etpub, jaymod, or noquarter, you do not need to upload the omnibot folder; only the omni-bot folder.

note: for a windows installation, you may need to run the installer locally and then upload the extracted files.

### Setup

In your server config, add the omnibot\_path cvar. The omnibot\_path cvar needs to be set to the absolute path of the omni-bot folder. If you are unsure of the absolute path, you will need to ask your server provider. An example of an absolute path for a gameservers.com installation is:

```
seta omnibot_path = "/usr/local/games/et/[password]/[IP:PORT]/omni-bot/"
```

Once the omnibot\_path cvar is configured, use the server providers tools to add the +set fs\_game option to the startup commands. It is also recommended that com\_hunkmegs is set to at least 64:

```
etded +set fs_game jaymod +set com_hunkmegs 64 +set omnibot_enable 1 +exec server.cfg
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Installing\\_Omni-bot](http://omni-bot.com/wiki/index.php?title=Installing_Omni-bot)"

This page has been accessed 8,392 times. This page was last modified 11:42, 22 August 2008.

# Configuring Omni-bot

From Omni-bot Wiki

**Sorry, this page is a stub at the moment.** For the time being, we refer you to the following pages:

- [Installing Omni-bot](#) - Information on installing Omni-bot to your computer or server.
- [Omni-bot F.A.Q.](#) - Frequently asked questions about Omni-bot.
- [Omni-bot Script Reference](#)
- [Bot Library](#) (mainly for bot scripts)
- [Omni-bot Map Scripting](#)
- [Community provided scripts](#)

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Configuring\\_Omni-bot](http://omni-bot.com/wiki/index.php?title=Configuring_Omni-bot)"

This page has been accessed 2,564 times. This page was last modified 03:57, 28 April 2008.

# FAQ

From Omni-bot Wiki

## Contents

- 1 General FAQ
  - 1.1 What is a bot?
  - 1.2 How do I install Omni-bot?
- 2 Scripting FAQ
  - 2.1 What are map scripts for?
  - 2.2 Why do I need a map script?
  - 2.3 What does a map script do?

## General FAQ

### What is a bot?

See About Bots

### How do I install Omni-bot?

See Installing Omni-bot, and make sure you are running a supported mod by looking here  
Supported Games

## Scripting FAQ

### What are map scripts for?

When a bot plays a game of some complexity such as Enemy Territory, where there is a lot of different things each class can do, and a specific sequence of objectives that need to be accomplished to 'win' the map, bots need help to determine what they need to do and in what order. Omni-bot does this using scripts that are written for each map. The script is responsible for disabling and enabling certain goals in the map in order to make sure the bots focus on goals that are relevant to the current objective. Without a map script, the bots would potentially choose goals to pursue that serve no purpose in furthering their team's objectives.

### Why do I need a map script?



Technically, you don't *need* a map script. Simply having a waypoint for a map should be sufficient enough to get basic working bots. Without a decent script though, there is a good chance that there will be certain map objectives that the bots won't do. A good quality map script will greatly improve the bots behavior in a map. A good map script is set up to focus the attention of the bots toward the objectives that are relevant to the particular stages of the game. A game type and/or map with highly dynamic objectives will likely need a map script to help the bots understand the dynamic nature of the maps objectives.

## What does a map script do?

First, an overview of how the bot system is set up. For any given map in any given game, there will be a number of Map Goals that become available for the bots to perform. These maps goals can be automatically detected goals, or goals that were a result of specific flags placed on waypoints.

### Examples of Map Goals

- Flags for CTF
- Capture Points for CTF
- Checkpoints to capture
- Defend points
- Attack points
- Sniper points

Regardless of the source of each type of map goal, whether it be auto detected, manually placed on a waypoint, or possibly even created from script, they are all treated the same internally.

There are some problems with having so many map goals, each representing something that bot should do.

- What order should they do the goals in?
- When should a map goal, or group of mapgoals, be ignored?

These questions just can't be answered by the bot alone. It's impossible to detect or otherwise calculate the answer to these questions in any meaningful manner. This is where map scripts come in. The entire point of map scripts is to supply direction to the goals that the bots focus on. By adjusting goal priorities, or enabling/disabling entire groups of goals in certain circumstances, a map script can ensure the bots are focused on those objectives that mean the most depending on the context of the game.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=FAQ>"

# Omni-bot Waypointing

From Omni-bot Wiki

## Contents

- 1 What are waypoints?
- 2 Important Waypoint Notes
  - 2.1 Waypointing in ET
- 3 Backwards Compatibility
- 4 How do bots use waypoints?
  - 4.1 Getting Stuck
- 5 How to waypoint
- 6 Waypoint Visual Aids
- 7 Special Case Waypointing
- 8 Placing Waypoints at MGs
- 9 Blockable Waypoints
- 10 Waypointing Tips
- 11 Waypoint Commands
  - 11.1 waypoint\_add
  - 11.2 waypoint\_addflag
  - 11.3 waypoint\_addflagx
  - 11.4 waypoint\_autobuild
  - 11.5 waypoint\_autoradius
  - 11.6 waypoint\_benchmark
  - 11.7 waypoint\_benchmarkgc
  - 11.8 waypoint\_benchtrace
  - 11.9 waypoint\_biconnect
  - 11.10 waypoint\_biconnectx
  - 11.11 waypoint\_changeradius
  - 11.12 waypoint\_clearallflags
  - 11.13 waypoint\_clearcon
  - 11.14 waypoint\_clearproperty
  - 11.15 waypoint\_color
  - 11.16 waypoint\_connect
  - 11.17 waypoint\_connectx
  - 11.18 waypoint\_dcall
  - 11.19 waypoint\_del
  - 11.20 waypoint\_deleteaxis
  - 11.21 waypoint\_info
  - 11.22 waypoint\_load
  - 11.23 waypoint\_lockselected
  - 11.24 waypoint\_mirror
  - 11.25 waypoint\_move
  - 11.26 waypoint\_save
  - 11.27 waypoint\_select
  - 11.28 waypoint\_setdefaultradius
  - 11.29 waypoint\_setfacing

- 11.30 waypoint\_setname
- 11.31 waypoint\_setproperty
- 11.32 waypoint\_setradius
- 11.33 waypoint\_shownames
- 11.34 waypoint\_stats
- 11.35 waypoint\_translate
- 11.36 waypoint\_unlockall
- 11.37 waypoint\_view
- 11.38 waypoint\_viewfacing
- 12 Waypoint Flag List
  - 12.1 allies (ET specific)
  - 12.2 ammo
  - 12.3 armor
  - 12.4 arty\_spot (ET specific)
  - 12.5 arty\_target\_d (ET specific)
  - 12.6 arty\_target\_s (ET specific)
  - 12.7 attack
  - 12.8 axis (ET specific)
  - 12.9 blockable
  - 12.10 bridge (ET specific)
  - 12.11 cappoint
  - 12.12 climb
  - 12.13 closed
  - 12.14 crouch
  - 12.15 defend
  - 12.16 detpack (ETF specific)
  - 12.17 door
  - 12.18 elevator
  - 12.19 goto
  - 12.20 health
  - 12.21 inwater
  - 12.22 jump
  - 12.23 jumpgap
  - 12.24 jumplow
  - 12.25 mg42 (ET specific)
  - 12.26 mine
  - 12.27 mortar
  - 12.28 mortar\_target\_d
  - 12.29 mortar\_target\_s
  - 12.30 movable
  - 12.31 pipetrapped (ETF specific)
  - 12.32 pipetrapped2 (ETF specific)
  - 12.33 prone
  - 12.34 script
  - 12.35 sentry (ETF specific)
  - 12.36 sneak
  - 12.37 snipe
  - 12.38 sprint

- 12.39 suplystation (ETF specific)
- 12.40 team1
- 12.41 team2
- 12.42 team3 (Not ET related)
- 12.43 team4 (Not ET related)
- 12.44 teamonly
- 12.45 teleport
- 12.46 underwater
- 12.47 wall (ET specific)
- 12.48 waterblockable
- 13 Community Waypoint Guidelines

## What are waypoints?

Omni-bot, like every bot needs a data structure that it can use to find its way around the map. Nearly every bot requires waypointing of some sort, whether it be automatically calculated from looking at the map geometry or hand placed like the majority of 3rd party bots.

Waypoints are basically a graph representation of the map geometry. Waypoints are vertices, and paths between them are edges. When a bot wants to get from where he his to a destination, the waypoint graph is used to find the way.

Waypoints are stored in `.way` files in the **nav** directory.

## Important Waypoint Notes

**Enemy Territory** uses a unique method of drawing waypoints. At first, lines were drawn similar to `g_debugbullets`, but this method uses an entity for each line, and is dependent on free entities and scales horribly to large numbers of drawn lines. Unfortunately, the debug drawing functionality present in vanilla Quake3 bot lib seems to have been removed in ET. With the entity based drawing, the client would hitch every 2 seconds while they recieved a new snapshot containing hundreds of lines. My solution to this problem was to set up an interprocess communication channel between the bot dll and the ET client dll(cgame). This means that the bot dll and client DLL communicate with a message queue of draw requests. The resulting system is much more capable of drawing much larger numbers of lines (now limited only by the polygon buffer on the client) without being constrained by free entities, and without hitching the client or bad flickering of waypoints.

This method is not without its quirks. Due to how the InterProcess Communication works, it requires the use of a temporary file. To make sure it can create this temporary file, make sure you have the following.

- Linux: A temporary directory pointed to by `TMP` or `TEMP` environment variable. They are checked in that order.
- Windows: A temporary directory pointed to by `TMPDIR`, `TMP`, or `TEMP` environment variable. They are checked in that order.

## Waypointing in ET

Due to the special method of waypointing in ET, waypointing is only available in the omnibot mod, which is a modification of etmain. **This means you cannot waypoint in etpub, jaymod, or NQ.** The authors of these mods and I thought it best to not add a needless dependency to their mods client dll. **If you need to waypoint in ET, run the omnibot mod as a listen server, and run cg\_omnibotdrawing 1 in the client console. Waypointing is not supported on dedicated servers, you must run a listen server.**

A recap.

- Waypoint only in omnibot mod in ET. **Not** in the mod(s) etpub, jaymod, or NQ.
- cg\_omnibotdrawing 1
- listen server only

## Backwards Compatibility

Occasionally the waypoint file format needs to be updated in order to support a new feature. Whenever this happens, Omni-bot is still capable of loading old version waypoints, and will save them as the new format if you waypoint\_save them. Omni-bot isn't however, forward compatible. That means old versions of Omni-bot cannot load a waypoint version file that was added in a newer version.

## How do bots use waypoints?

A basic understanding of how the bots use the waypoint graph can help you make good decisions on how to place waypoints, how to set radius, and how to understand when things might not be behaving as expected.

When a bot spawns into the map and decides where to goal, he performs a path search to determine how to get there. The overview of how the path search works is:

- Get the closest allowed waypoint to the bot.
- Get the closest allowed waypoint to the goal.
- Find a path between them, ignoring paths through waypoints that are limited to other teams, closed, or that exceed the bots movement capabilities.

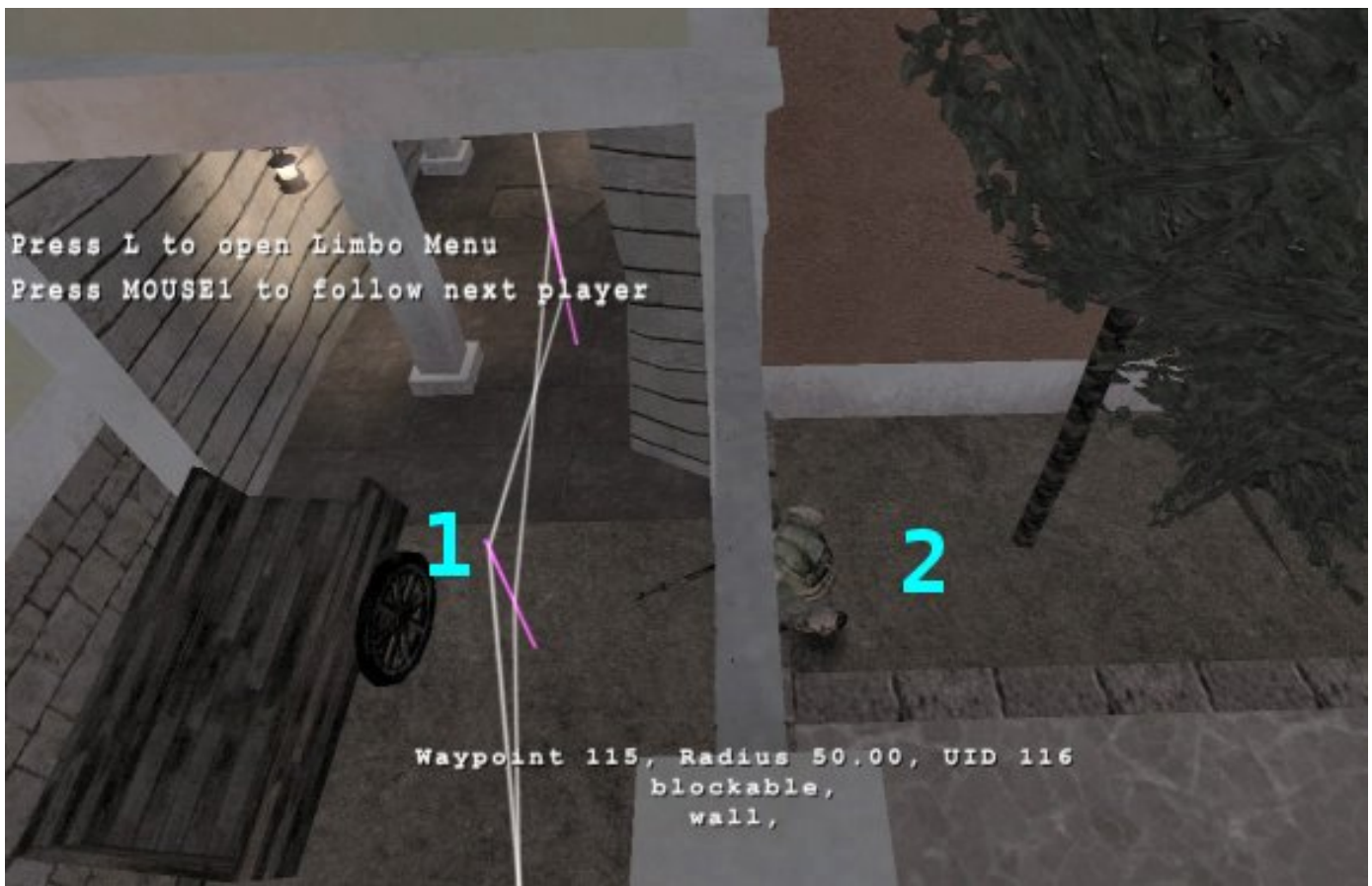
## Getting Stuck

Bots can't move perfectly, and sometimes they can fall off edges or otherwise get pushed or dislodged from their intended path, usually due to weapon push or game physics. When this happens, they can sometimes get stuck. When the bot is unable to make meaningful progress in his path for a small amount of time, it assumes it has become stuck, and the current path attempt will fail. The important part of this process is that there is suitable waypoint placement to recover from getting stuck. Here's an example.



In this screenshot, we see a path that goes across the top of the thin wall. Occasionally the bot may fall off the wall and end up in the small square enclosure below it. You may not think it necessary to place waypoints on that square area below the edge, because the bot should never need to go there, but without the 3 waypoints, when the bot falls down and fails his goal and goes to re-plan a new path, we need to make sure he finds a starting waypoint in a reachable location. If we didn't have the waypoints in the square area below the wall chances are he would pick the waypoints on top of the wall as the nearest points to start from, and result in a stuck loop since he can't get to the waypoints on top.

Here's another example that illustrates this kind of problem. In the following screenshot, the bot is obviously trying to reach the waypoint marked 1, because this is the closest waypoint to the bot's current position. An obvious solution would be to add another waypoint, close to the spot marked 2 in the image, and (bi)connect it to some other waypoint on the right.



This is a common problem with stuckage in maps. By understanding this you should be able to pick out areas of a map that could have this problem. Simply place a waypoint in the corner where they tend to get stuck and link it back to make them backtrack a bit and try again. Also see the section about blockable paths below. Failure to mark paths as blockable is another very common cause of stuckage problems.

## How to waypoint

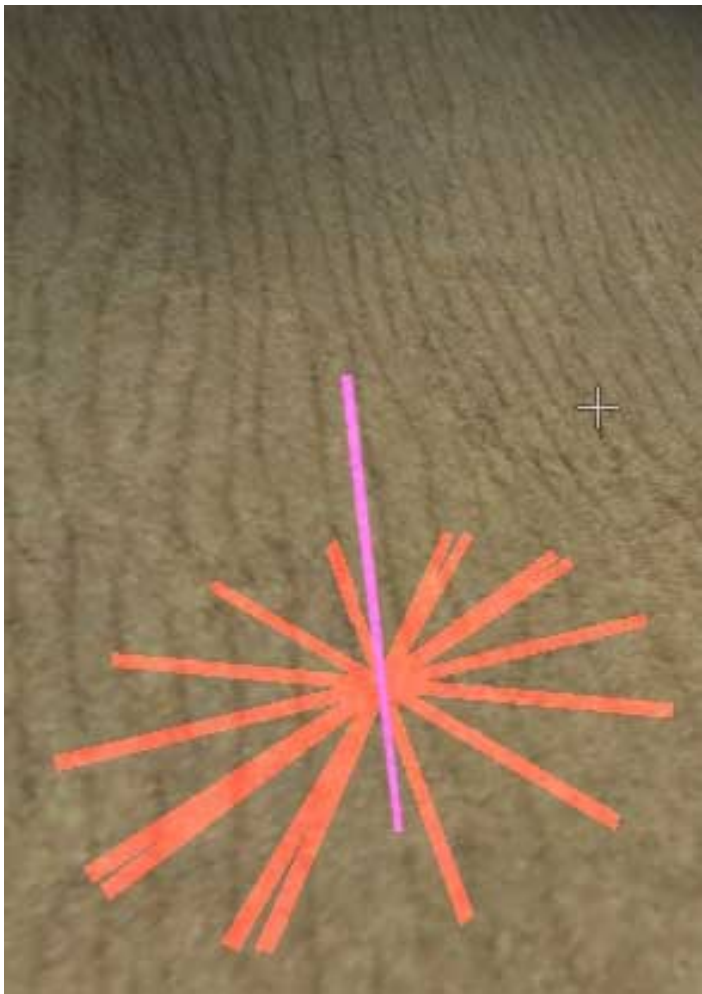
- Overview
  - Start a listen server
  - Load a map
  - Join as a player (it won't work while you're spectating)
  - Enable waypoint view: `/bot waypoint_view 1`
  - Place waypoints while running around the map as a normal player would: `/bot waypoint_add`
  - Connect waypoints: `/bot waypoint_(bi)connect`
  - Save the waypoints to a file: `/bot waypoint_save`
  - Create a script for the map that provides hints and assists the bots to focus on applicable goals. (This is strongly recommended, but might be unnecessary on very simple maps.)

See the [Enemy Territory Waypointing Tutorial](#).

## Waypoint Visual Aids

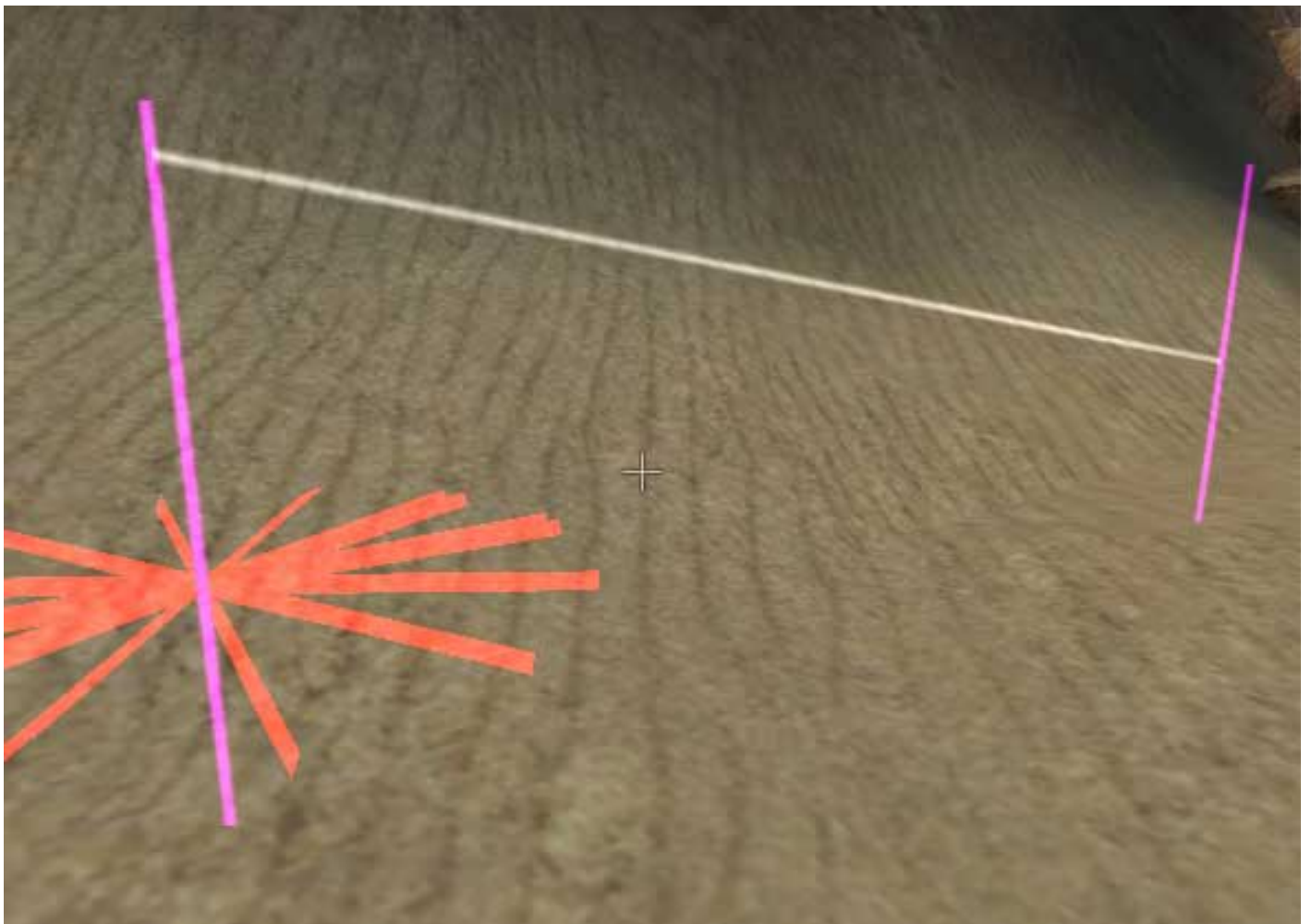


These screenshots show the waypointing aids in their default color configurations.

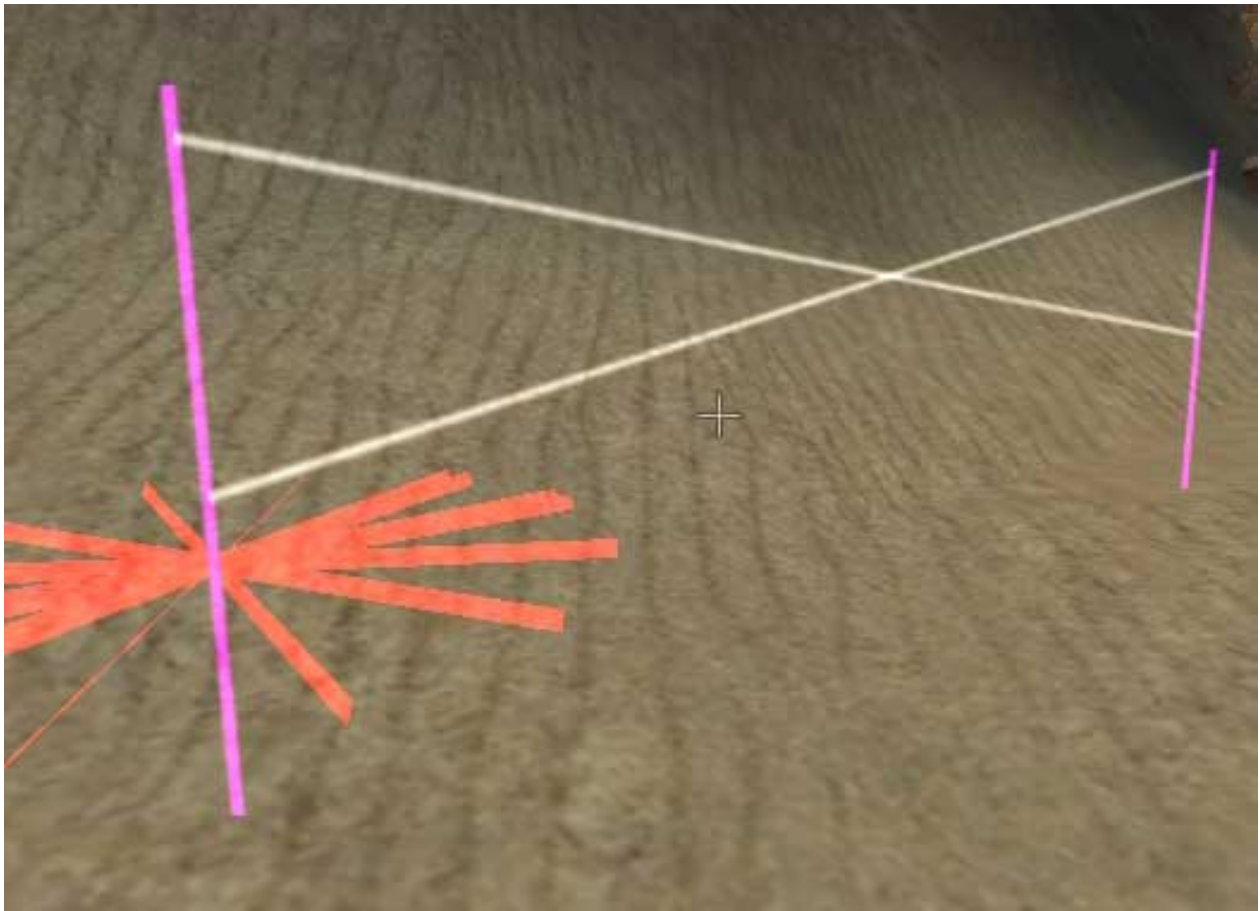


This is what a single waypoint looks like after a `waypoint_add` command. The vertical line is the visual representation of the waypoint. When aiming at waypoints, you will see a circular configuration representing the waypoints radius. The size will vary according to the radius of the waypoint and is useful to determine just how large the radius actually is. The default radius shown here is 35 game units, which is almost always a good choice to start with.



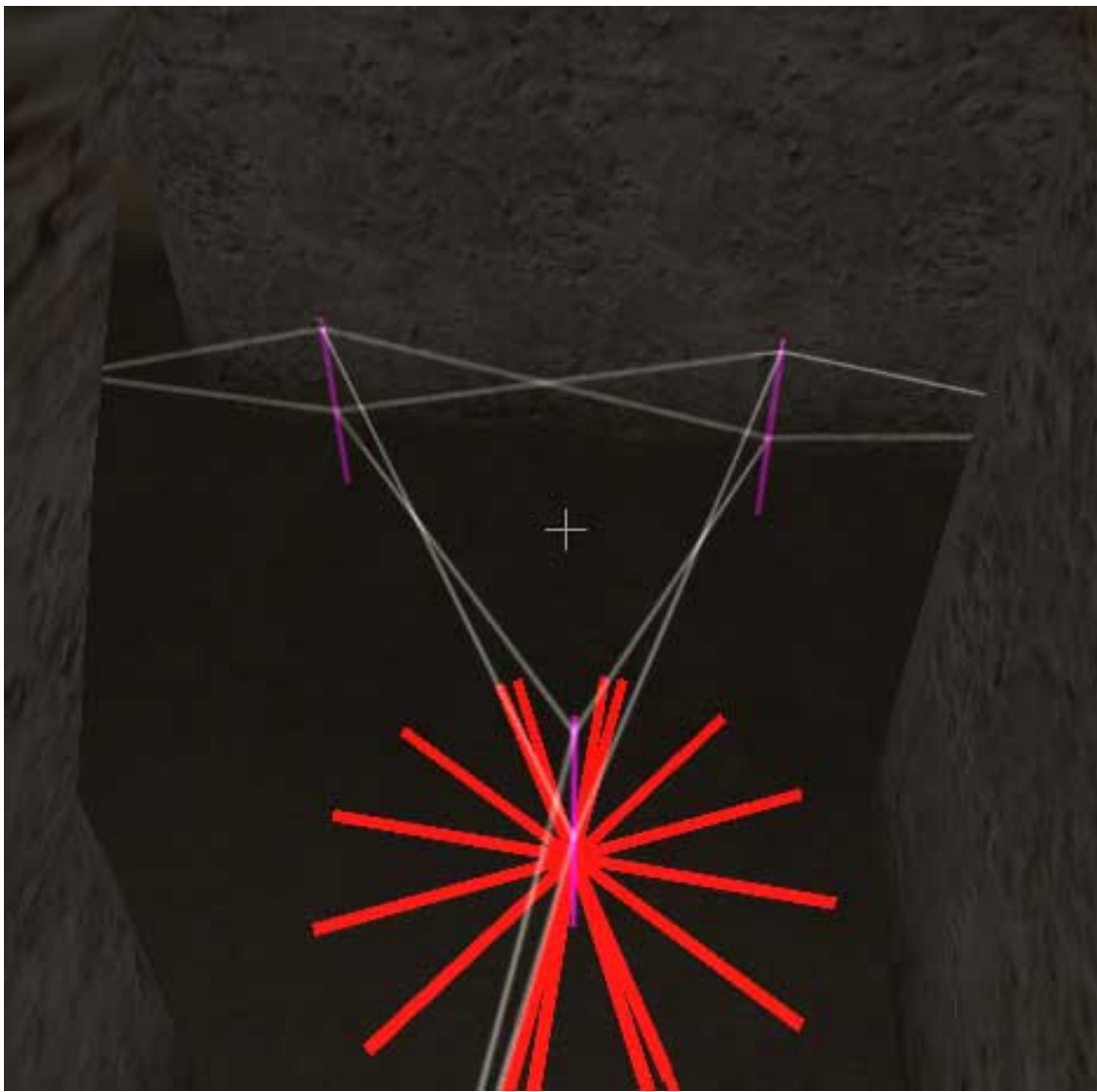


When connecting waypoints, it is important to understand how connections are displayed. In this screenshot, it shows 2 waypoints connected by a one way path. We know the connection here is from the left waypoint to the right because of the angle of the link. Link direction is shown with a downward sloping link. The reason for this is because a slope is a better indication of direction and one/two way links than a single line, even if it is colored differently.



Here we see what a two way connection looks like. Since there is a connection going both ways, the resulting link appears as an 'X' between the waypoints.

## Special Case Waypointing



Although not required, there are some configurations of waypoints that can help streamline bot navigation a bit. T-connections for example can be waypointed with triangular shape of 3 waypoints in order to have a bit more control over the bots corner cutting. This isn't a required configuration for T-connections, and you can get pretty much the same behavior by increasing the radius of the waypoint at the intersection.



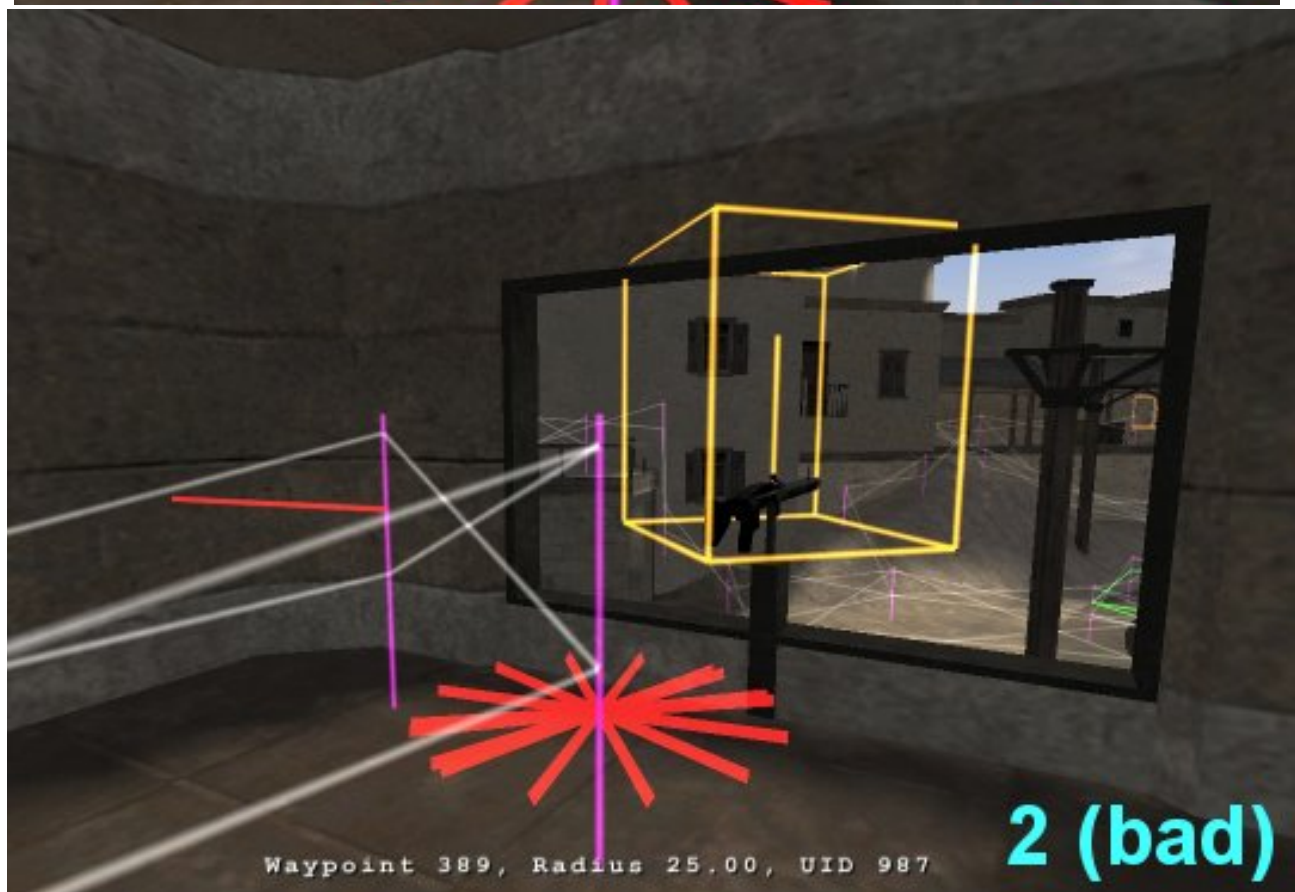
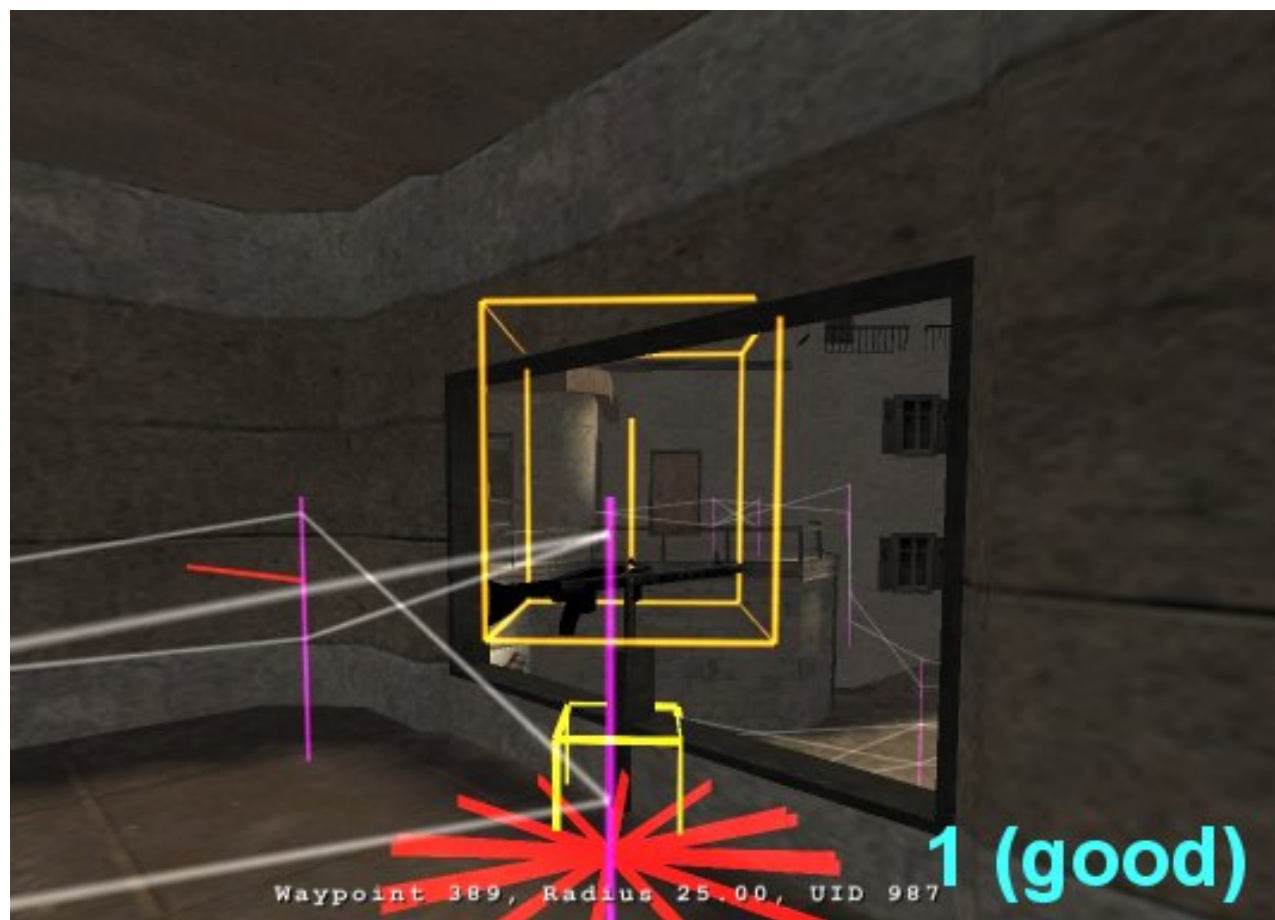
Doors are typically waypointed depending on how they operate. In many games, doors are automatic and simply open when the player approaches them. It isn't necessary to do anything special in the case of those doors. In some games, doors have to be 'used', such as in Enemy Territory. In these cases, the "door" waypoint flag is useful. The "door" flag is used by adding the flag to the 2 waypoints on either side of a door. This tells the bot to hit the USE key while moving across the link between the 2 waypoints. In this picture we see both waypoints flagged with door, and in this case they are team specific.

See also: an extra page on adding sniper spots to a map.

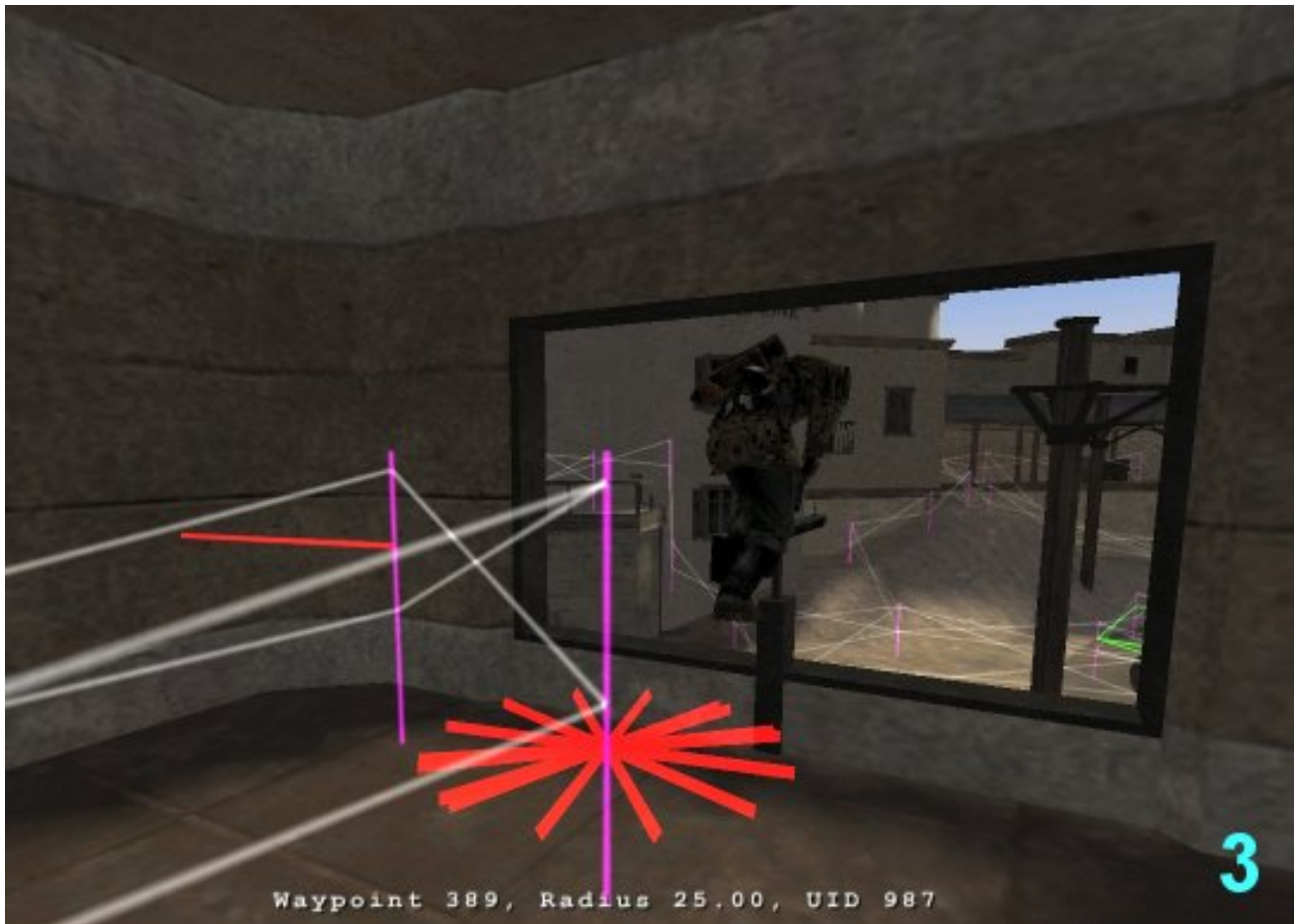
## Placing Waypoints at MGs

Before placing waypoints at an MG, you should ideally display the MG's bounding box, using the command `/bot draw_goals on ".*mg42.*"` Then make sure you place the waypoint immediately in front of one of the horizontal edges of the bounding box, not at one of its corners. This will make it much easier for the bots to mount the MG. The following screenshot 1 shows a good position for a waypoint, screenshot 2 a bad one:





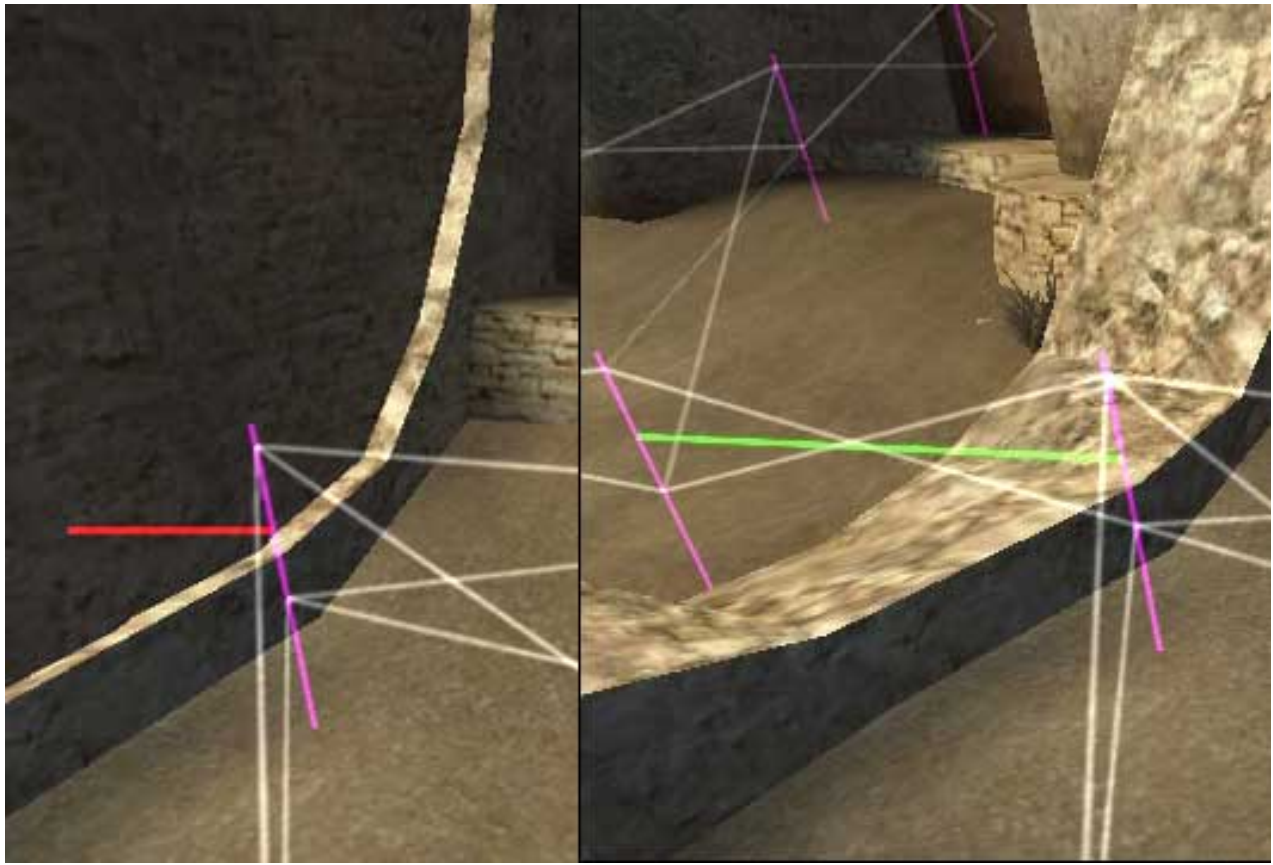
Although screenshot 2 shows a position that is a perfectly natural way for a human player to approach the weapon, it is badly suited for the bots' navigation. Frequently this will result in strange bot movements, as shown in screenshot 3:



## Blockable Waypoints

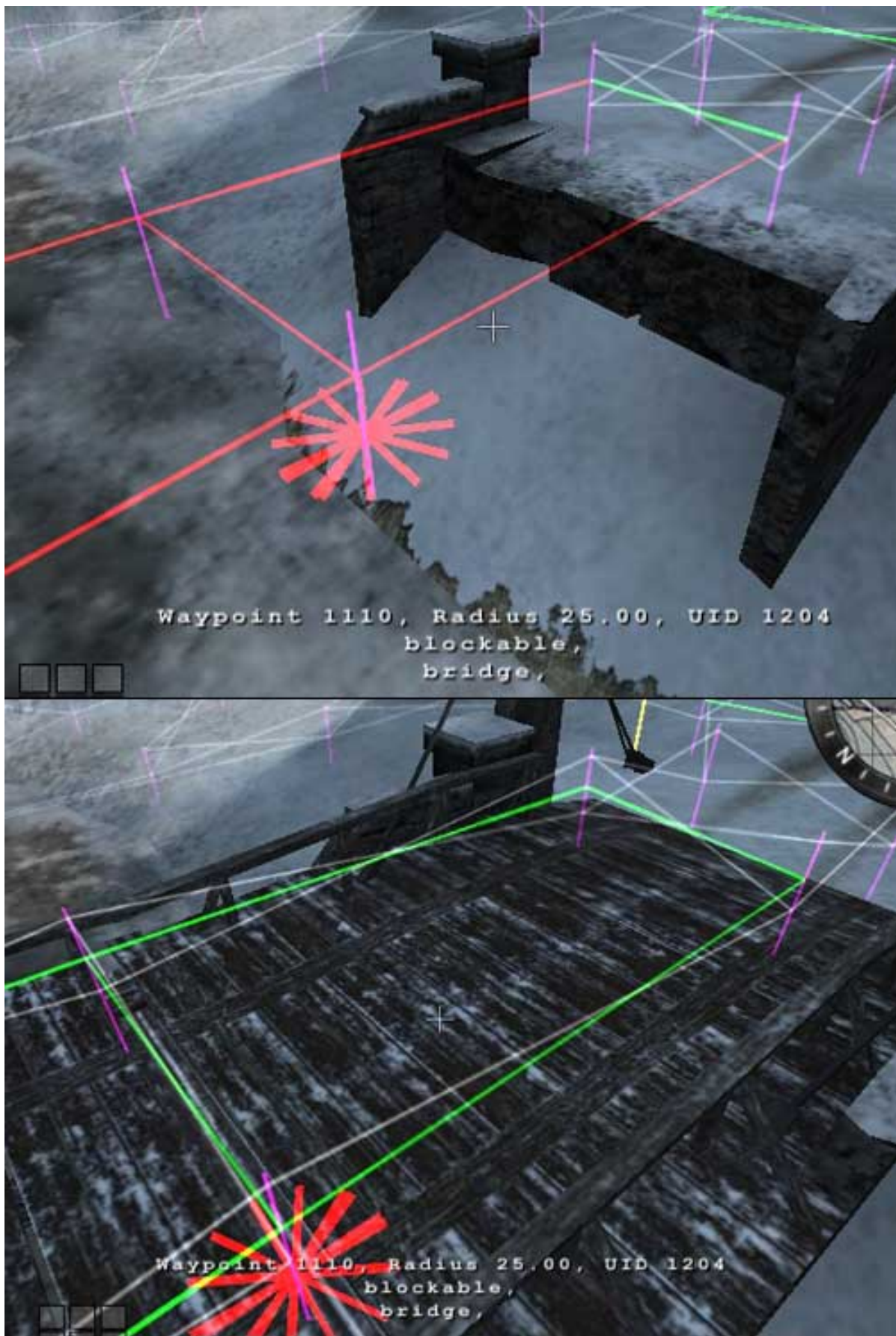
Blockable waypoints are a special case flag that deserves special mentioning. Similar to doors, the blockable flag needs to be placed on both sides of the object. Additionally, the blockable flag must be used in conjunction with another flag that determines the type of blockable path that is desired. Blockable paths are a feature of Omni-bot that allow paths to update their own blocked status, saving some scripting effort. The type of flag used in addition to the blockable flag determines how they figure out their blocked status.

- Blockable Types
  - wall - If line of sight check passes between the 2 waypoints, the path is opened, otherwise it is closed.
  - bridge - If an object isn't detected half-way between the 2 waypoints(such as a bridge), the object is blocked, otherwise it is open.



In the picture above, we see the dynamite wall in the old city in the oasis map. To manage the connection between the waypoints on either side of the wall, it is using the "blockable" and "wall" flag. On the left side, when the wall is present, the path is blocked, which is shown by the red line as the link. On the right, when the wall is destroyed, the line of sight test passes and the path is opened up automatically.





This is an example of a use for "blockable" and "bridge". The check for "bridge" is basically a probe to see if there is a surface detected under the link between the waypoints. When there is no bridge built, the probe doesn't detect the surface, and results in a blocked path. When the bridge is built, the surface

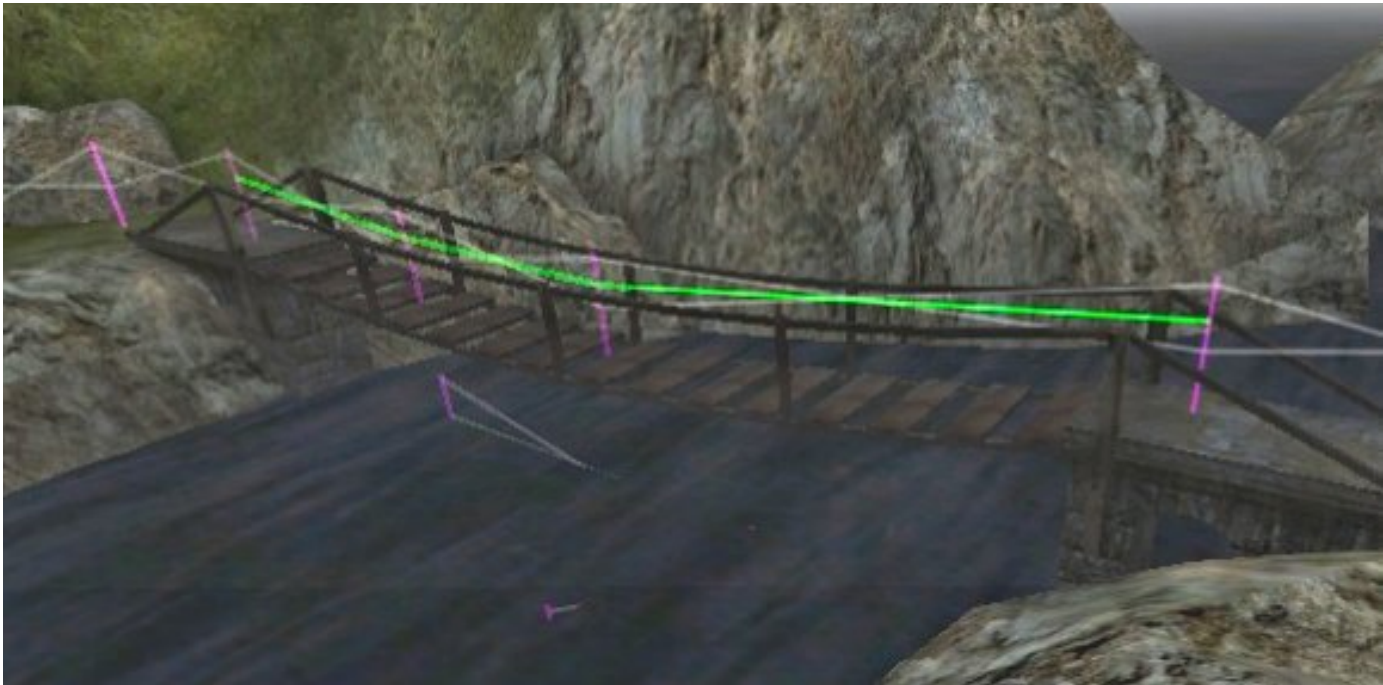


is detected, and the link is opened.

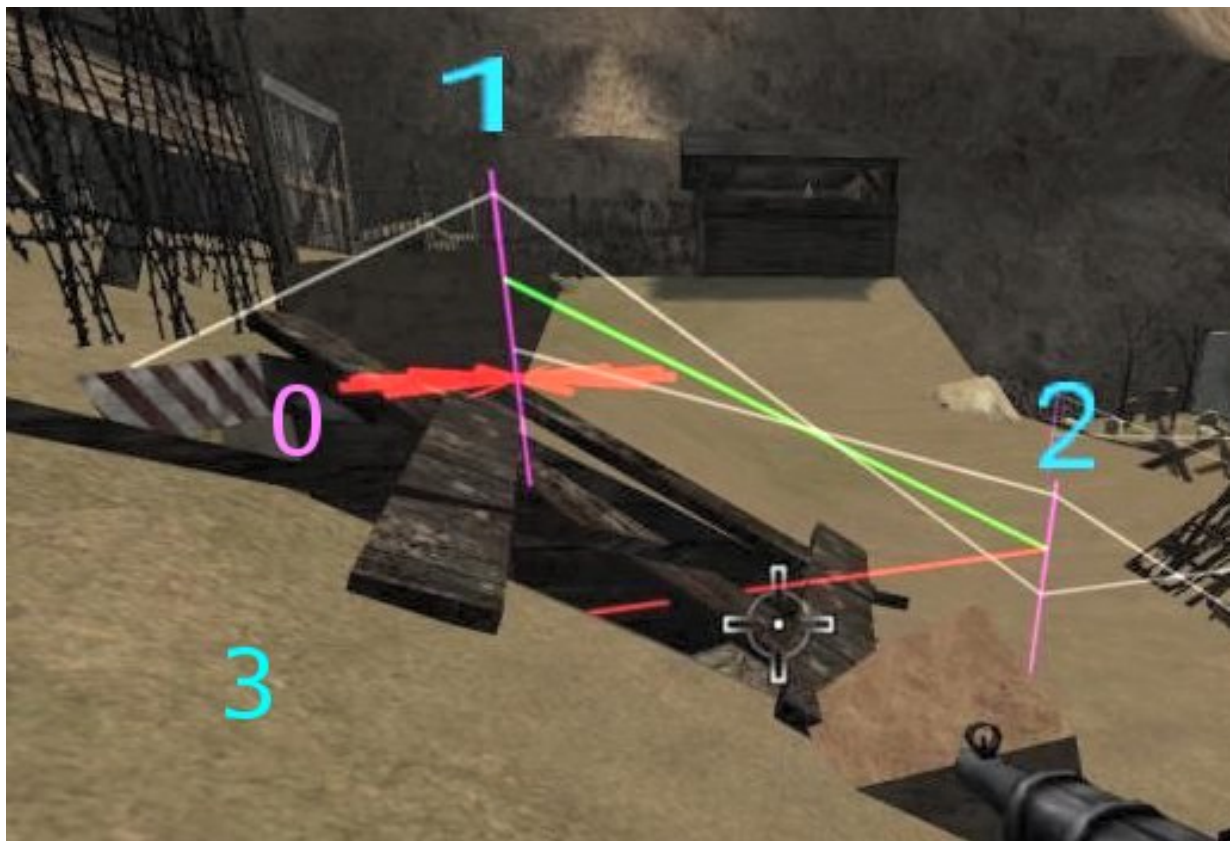
Due to this method of detecting the blocked status of a path across a bridge, rope bridges and similar constructions with a slightly curved shape may require three or more waypoints, all of which should have blockable and bridge flags. Otherwise, the detection of a surface beneath the path might fail, as can be seen in this example (taken from the Bergen map):



With more than two waypoints, the path is correctly recognized as usable by the bots:

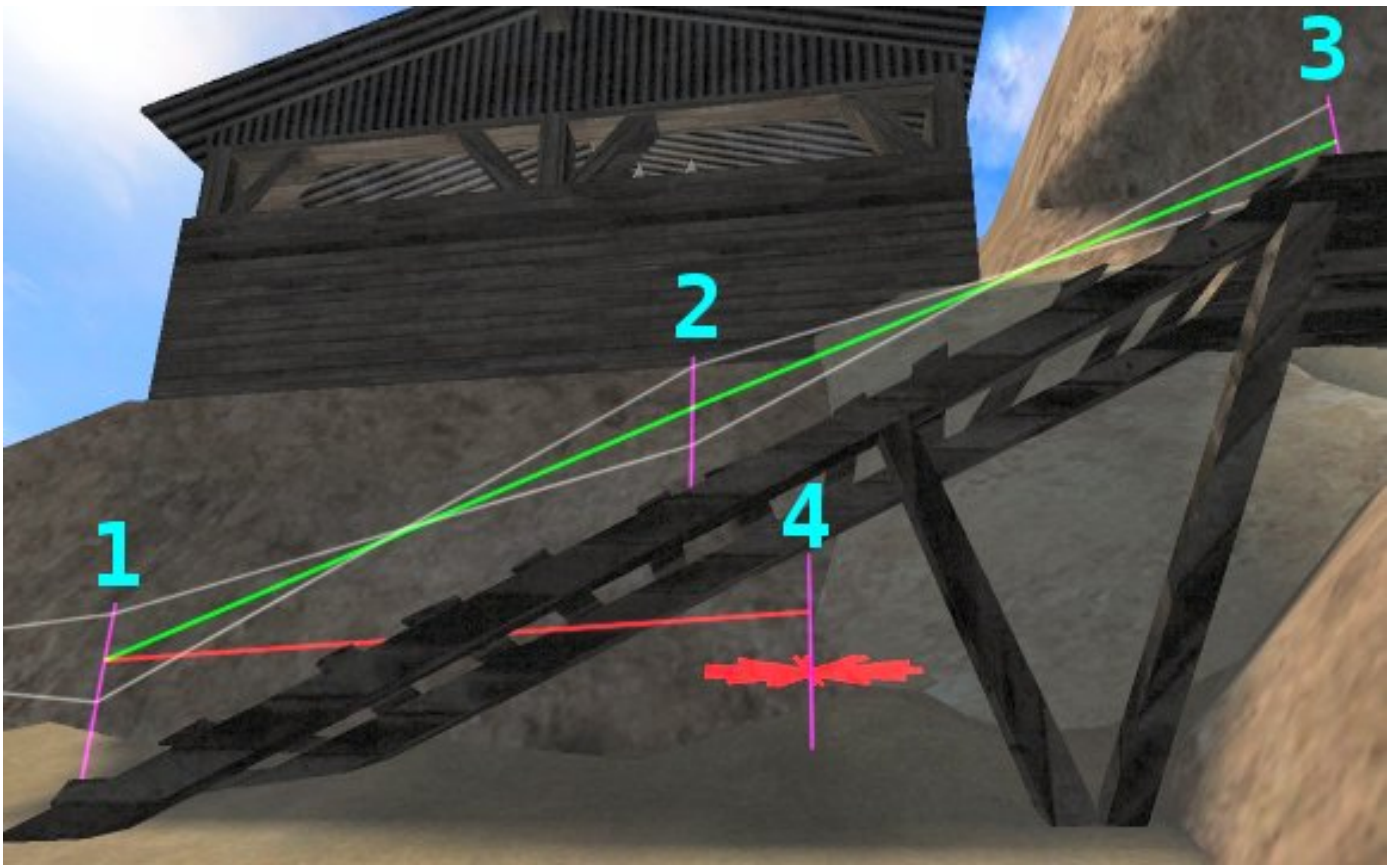


In a few cases, it makes sense to use both the wall and bridge flag on a single waypoint. The example below shows a bunker entrance with a destructible barrier in front of it (from the JFF Playground map). Waypoint 1 is marked blockable and bridge, waypoint 3 (inside the bunker) has the blockable and wall flag set, while waypoint 2 has all three flags: blockable, wall, and bridge. This is because it is connected to two different blockable waypoints. Depending on the status of the wooden barrier, the paths 2-3 and 2-1 will toggle between blocked and unblocked. (The reason for this complicated construction is that the bots should jump through the small opening marked 0 in the image if the barrier is intact.)

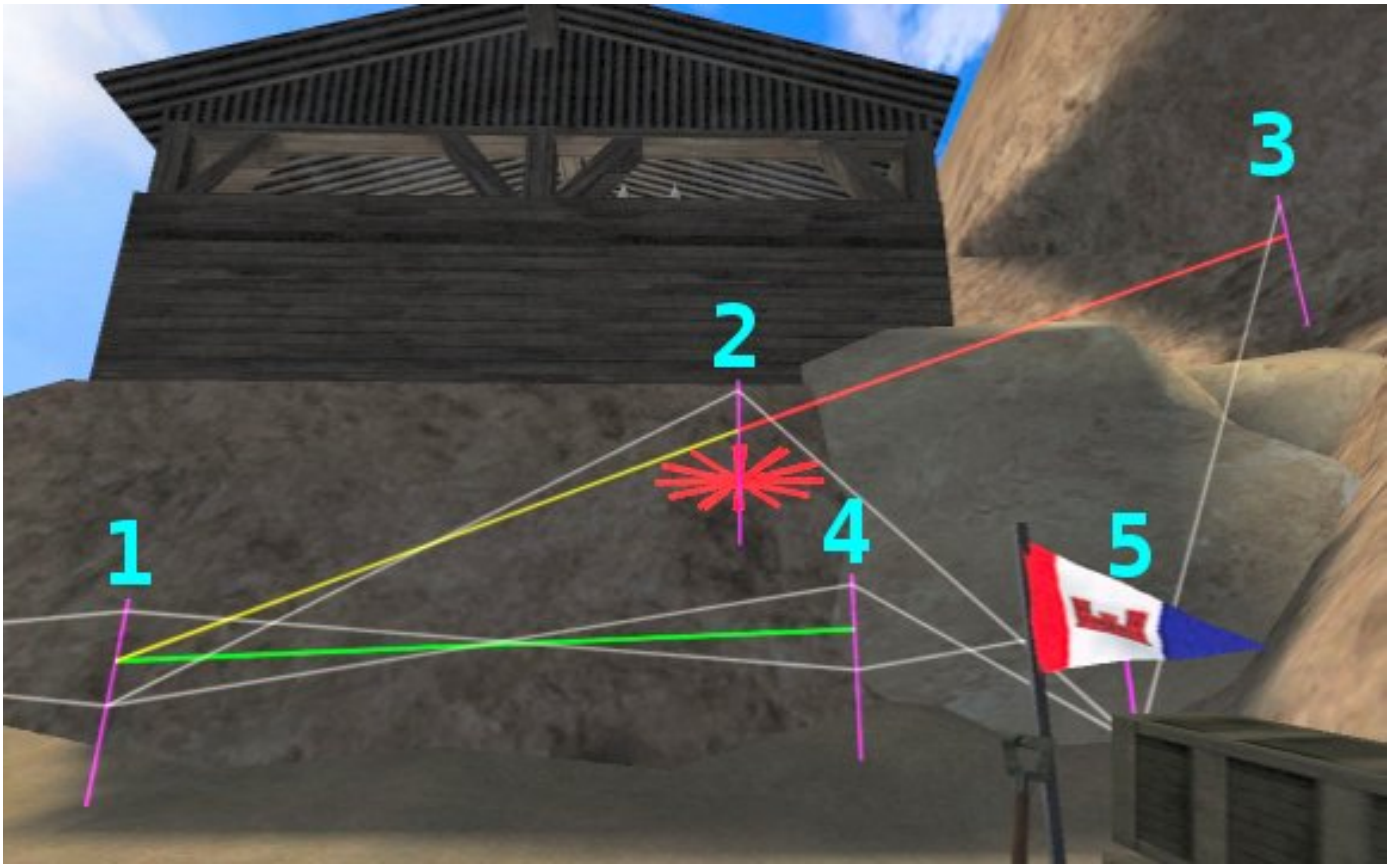


A similar example: The following screenshot shows a destructible assault ramp, again taken from JFF Playground. All the waypoints shown here, except 4, have the blockable and bridge flags set, waypoints 1 and 4 blockable and wall. When the ramp has been built, the path 1-4 is blocked, while 1-2-3 is free:





When the barrier is destroyed, the status of both paths is toggled:



Additionally, a new waypoint 5 (partly hidden by the crates) was added between the shots. The monodirectional connections 2-5 and 3-5 tell the bots that they can move from 2 or 3 to 5, but not vice versa.

You may wonder what waypoint 4 in the above example is good for. While it is not strictly necessary, it is not a bad idea to have it, mainly for two reasons: Without waypoint 4, a bot that got pushed under the barrier due to some circumstances would try to move towards waypoint 2 (because it's the closest one) and get stuck there. The connection between 4 and 5 enables the bot to find a working path back. Furthermore, waypoint 4 is probably a good place for engineer bots to plant dynamite, slightly better than 2, because it is less exposed.

Keep in mind that paths can be blocked by vehicles too. All the waypoints along the path of a vehicle should have the blockable and wall flag set. There is a page dedicated to vehicle pathing.

## Waypointing Tips

- Save often. Although the bot is pretty stable, it is a good habit to save often.
- Disable the time limit of the map or set it to something very high if possible. In ET, use the command `/ref timelimit 0` to have unlimited time on a map. If a map restarts while you are editing waypoints, your unsaved changes will be lost.
- When learning to waypoint, spend some time looking at existing waypoints, in particular the ones handled by the main Omni-bot developers.
- Have a decent understanding of the objectives of the map. It should help you create more direct paths between spawn areas and the objectives each team will likely be heading for during play.
- Understand the radius. The radius is effectively the tolerance for moving toward a waypoint. If you need the bot to get closer, use a smaller radius, if the waypoint represents a large room, use a large radius.
- Don't use radius values of less than 10-20 or so. You may need to experiment with this.
- TEST. There is no substitution for testing when it comes to verifying the bots can accomplish the objectives and that any problem spots for navigation can be recovered from. Play through full matches on both sides as well as spectator to watch the bots as they navigate.
- Make sure to place waypoints in areas where the bots can accidentally get off the main path.
- Try and limit usage of one way paths. Keep in mind that the bots use pathing for short term goals as well, so any area that a player or bot may need to be revived should have two way connections.
- If you see a bot get stuck during waypoint testing, add waypoints to the area while the bot is still stuck. The waypoints are updated dynamically so you should see the bot use any new waypoints you add to unstick them.
- Use key bindings. This does not only save a lot of typing. In some situations (e.g. with underwater waypoints) it may even be close to impossible to edit waypoints without shortcut keys. In ET for example, a bot command is bound to a key using the following command in the game console: `/bind <key> "bot <command>".` For example: `/bind f5 "bot waypoint_setradius 25"`
- You can easily save lots of these bind commands to a plain text file, say `waypointing.cfg`, and later execute this file by issuing the following in the console: `/exec waypointing.cfg`
- Use the game's `devmap` command to load the map, e.g. `/devmap Venice`. This enables a 'cheat mode' that enables you to walk through walls, warp to a certain position, etc. Read more about `devmap`.

- If the map is based on an "escort the vehicle" mission, read about the Vehicle Pathing utility before you start. It may save you a lot of trouble.

## Waypoint Commands

Waypoint commands are accepted through the games console, and can commonly be assigned to key bindings in the supported games.

### Executing a command

- In ET, ETF, or other Quake 3 engine based games - **/bot waypoint\_view 1**
- In Quake4, Fortress Forever, and other Half-life 2 or Doom 3 engine based games, the / isn't required, so simply: **bot waypoint\_view 1**

**NOTE:** ALL of these commands, except waypoint\_view can only be executed when waypoint\_view is enabled.

### waypoint\_add

*Usage:* waypoint\_add

*Example:* waypoint\_add

- Adds a waypoint at the location where you are currently standing. When executing waypoint\_add, a new waypoint will be created at the location you are currently standing. Some waypoint flags may be automatically placed on the waypoint.
  - If the waypoint is underwater, the underwater flag should be automatically added.
  - If you are crouching when you place the waypoint, the crouch flag should be automatically added.
  - If you are prone when you place the waypoint, the crouch flag should be automatically added instead of adding prone flag.

### waypoint\_addflag

*Usage:* waypoint\_addflag flags[string] ...

*Example:* waypoint\_addflag crouch

- Flags the nearest waypoint with named properties, or clears the flag from the waypoint if it already exists. This command can take any number of waypoint flags. See the list of flags below.

### waypoint\_addflagx

*Usage:* waypoint\_addflagx flags[string] ...

*Example:* waypoint\_addflagx crouch

- Same as `waypoint_addflag`, only instead of the nearest waypoint, uses the waypoint you are currently aimed at.

## **waypoint\_autobuild**

*Usage:* `waypoint_autobuild dc[1/0] bbox[1/0] limitheight[#] limitdist[#] maxconnections[#]`

*Example:* `waypoint_autobuild 1 0 32 1024 4`

- This command automatically creates connections between waypoints based on the parameters used. It is a time saving function that can generate a pretty good starting point for paths between waypoints. It is not meant to do all the work for you, and will likely need additional cleanup after using where you may need to add additional connections it missed, or removed bad connections in may have added.
  - `dc` - Disconnect all current connections before auto connecting.
  - `bbox` - Use a small bounding box when casting the ray between waypoints. Useful for filtering out some connections that clip closely to a wall.
  - `limitheight` - Only make connections to waypoints that are only within this height difference. Useful to prevent connections between multiple levels in a wide open room, even if there is line of sight between them.
  - `limitdist` - Only make connections to other waypoints within this distance.
  - `maxconnections` - Only make a maximum of this many connections. 3-5 tends to be a good range to start with.

## **waypoint\_autoradius**

*Usage:* `waypoint_autoradius all/cur[string] height[#] minradius[#] maxradius[#]`

*Example:* `waypoint_autoradius all 32 20 200`

- Automatically detects a waypoint radius on the nearest or all waypoints(depending on 1st parameter).
  - `all/cur` - `all` performs autoradius on all waypoints, `cur` performs it on the nearest.
  - `height` - A vertical offset from the waypoint position to perform the collision tests with. This is useful due to differences in waypoint position between games. In Doom3 & HL2 engine games, the position is at the ground, while in ET the position is about half-way up a players height.
  - `minradius` - The minimum radius to use.
  - `maxradius` - The maximum radius to use.

## **waypoint\_benchmark**

*Usage:* `waypoint_benchmark`

*Example:* `waypoint_benchmark`

- Development tool. Executes a path search between every waypoint and every other waypoint.

Used to test the speed of the path finding system.

## **waypoint\_benchmarkgc**

*Usage:* waypoint\_benchmarkgc iterations[#]

*Example:* waypoint\_benchmarkgc 1

- Development tool. Performs a GetClosestWaypoint test with every waypoint. Used to test the speed of nearest waypoint lookups.

## **waypoint\_benchtrace**

*Usage:* waypoint\_benchtrace iterations[#]

*Example:* waypoint\_benchtrace 1

- Development tool. Performs a traceline collision test between every waypoint. Useful for internal testing of traceline speed.

## **waypoint\_biconnect**

*Usage:* waypoint\_biconnect

*Example:* waypoint\_biconnect

- Same as waypoint\_connect, but results in a 2 way connection between the waypoints.

## **waypoint\_biconnectx**

*Usage:* waypoint\_biconnectx

*Example:* waypoint\_biconnectx

- Same as waypoint\_connectx, but results in a 2 way connection between the waypoints.

## **waypoint\_changeradius**

*Usage:* waypoint\_changeradius change[#]

*Example:* waypoint\_changeradius -10

- Change the radius of the nearest waypoint by the change amount. Useful if you want to bind a increase and decrease radius key, such as a mouse wheel.

## **waypoint\_clearallflags**

*Usage:* waypoint\_clearallflags flags[string] ...

*Example:* waypoint\_clearallflags crouch

- Clears a list of flags from all waypoints in the map. Like waypoint\_addflag, it can take any number of flag names, and clears the flags from all waypoints in the map. Useful if you want to remove all of a particular flag from a map.

## **waypoint\_clearcon**

*Usage:* waypoint\_clearcon

*Example:* waypoint\_clearcon

- Clears all the connections from the nearest waypoint or all selected waypoints(if any).

## **waypoint\_clearproperty**

*Usage:* waypoint\_clearproperty name[string]

*Example:* waypoint\_clearproperty bias

- Clears a property from a waypoint by its name.

## **waypoint\_color**

*Usage:* waypoint\_color

*Example:* waypoint\_color type[string] red[#] green[#] blue[#]

- Allows colors to be configured for certain types of waypoints.
  - waypoint\_color - Default color of waypoints.
  - waypoint\_selected - Selected waypoints.
  - link\_closedcolor - If the waypoint has a closed flag.
  - link\_teleport - If the waypoint has a teleport flag.
  - link\_1way - One-way connection links.
  - link\_2way - Two-way connection link.
  - blockable\_blocked - Blockable line indicators when it is blocked.
  - blockable\_open - Blockable line indicators when it is open(unblocked).
  - aimentity - Color of the box drawn around entities when you aim at them with waypoint\_view enabled.
  - radius - Color of the radius indicator.
  - team1 - Team 1 flagged waypoints
  - team2 - Team 2 flagged waypoints
  - team3 - Team 3 flagged waypoints (not for ET)
  - team4 - Team 4 flagged waypoints (not for ET)



## **waypoint\_connect**

*Usage:* waypoint\_connect

*Example:* waypoint\_connect

- Flags the nearest waypoint for a connection. If a waypoint is already flagged for connection that waypoint will be connected to this waypoint with a 1-way connection.

## **waypoint\_connectx**

*Usage:* waypoint\_connectx

*Example:* waypoint\_connectx

- Same as waypoint\_connect, only uses the waypoint you are aiming at.

## **waypoint\_dcall**

*Usage:* waypoint\_dcall

*Example:* waypoint\_dcall

- Disconnect all waypoints in the map. This removes all connections, while leaving only the waypoints.

## **waypoint\_del**

*Usage:* waypoint\_del

*Example:* waypoint\_del

- Deletes a nearest waypoint that you are standing within 100 units of. All connections from and to the waypoint will get automatically removed as well.

## **waypoint\_deleteaxis**

*Usage:* waypoint\_deleteaxis axis[string]

*Example:* waypoint\_deleteaxis

- Deletes all waypoints on a specified side of a specified axis.
  - axis - x, y, z, -x, -y, -z, which side of which axis to delete all waypoints from. Mainly useful for symmetrical maps when you wish to delete a side, make adjustments, and re-mirror the waypoints.

## waypoint\_info

*Usage:* waypoint\_info

*Example:* waypoint\_info

- Prints some basic information about the nearest waypoint.

## waypoint\_load

*Usage:* waypoint\_load name[string, optional]

*Example:* waypoint\_load

- name is optional and if used, loads the waypoints for the currently loaded map with the suffix name. For example, if the map d3ctf1 is loaded, waypoint\_load half would load from the nav directory the file d3ctf1half.way
- Loads the waypoints for the currently loaded map. Waypoints are loaded from the nav directory under the mod currently running. Warning: Any changes you may have made to the waypoints since the last load will be lost. This command is typically used for undoing changes made after the last save.

## waypoint\_lockselected

*Usage:* waypoint\_lockselected

*Example:* waypoint\_lockselected

- Locks all currently selected waypoints. Locked waypoints aren't effected by waypoint\_translate.

## waypoint\_mirror

*Usage:* waypoint\_mirror axis[string] p[optional]

- axis can be defined as x, y, or z
- p is optional, and if defined will use your current position to define an axis

*Example:* waypoint\_mirror x

- This command mirrors all current waypoints and rotates the mirrored waypoints across the origin of a provided axis. This function is useful for mirroring the waypoints in a symmetrical map, where each team has an identical base. The optional p parameter is useful for when the map isn't centered around the 0 axis. Stand as closely to the center of the map as possible and execute the command including the optional p parameter and the waypoints will be mirrored and offset based on the player position.

## waypoint\_move

*Usage:* waypoint\_move

*Example:* waypoint\_move

- Grabs the nearest waypoint for moving. If a waypoint has already been grabbed by a previous call to waypoint\_move, it is dropped at your current position. Connections are maintained throughout waypoint moving.

## **waypoint\_save**

*Usage:* waypoint\_save

*Example:* waypoint\_save

- name is optional and if used, saves the waypoints for the currently loaded map with the suffix name. For example, if the map d3ctf1 is loaded, waypoint\_save half would save the waypoints to the nav directory as the file d3ctf1half.way
- Saves the waypoints for the currently loaded map. Waypoints are saved to a file called mapname.way in the nav directory under the mod currently running. Warning: This will overwrite an existing waypoint file of the same name without asking. Rename or backup your original waypoint files when you want to experiment.

## **waypoint\_select**

*Usage:* waypoint\_select radius[#]

*Example:* waypoint\_select 500

- Selects all waypoints in the provided radius. If no radius is provided, the selection is cleared. Some of the above functions work on selected waypoints.

## **waypoint\_setdefaultradius**

*Usage:* waypoint\_setdefaultradius radius[#]

*Example:* waypoint\_setdefaultradius 100

- Sets the default radius that is used with all waypoints placed. Useful if you plan to waypoint\_add a bunch of waypoints that will share a radius different from the current default to save time going back and changing radius manually.

## **waypoint\_setfacing**

*Usage:* waypoint\_setfacing

*Example:* waypoint\_setfacing

- Sets the facing for the nearest waypoint to your current facing. A waypoint's facing is basically a vector associated with a waypoint that tells the bot in which direction to look when the bot is camping on that waypoint (due to a sniper or defend flag in most cases). Waypoint facing can also be used for various purposes in scripts.

## **waypoint\_setname**

*Usage:* waypoint\_setname name[string]

*Example:* waypoint\_setname spawnarea

- Sets the name of the nearest waypoint. Waypoint names are used as part of the map goal name for any goals that are created as a result of the flags on a waypoint. For example, if you set a waypoint's name to "barrier\_axis1" and the waypoint has a sniper flag set, it will appear as MAP\_SNIPER\_SPOT\_barrier\_axis1 in the bots' goal list. It's much easier and less error-prone to manipulate goals in a mapscript based on meaningful names rather than UIDs.

## **waypoint\_setproperty**

*Usage:* waypoint\_setproperty name[string] value[string]

*Example:* waypoint\_setproperty bias 0.6

- Sets an arbitrary property value by name. This is the current method of allowing a waypointer to set the bias of a goal by setting a bias as a property on goals that are then used to create maps goals.

## **waypoint\_setradius**

*Usage:* waypoint\_setradius radius[#]

*Example:* waypoint\_setradius 100

- Sets the radius of the nearest waypoint.

## **waypoint\_shownames**

*Usage:* waypoint\_shownames expression[string, optional]

*Example:* waypoint\_shownames ATTACK.\*

- Prints all waypoint id's and names that optionally match an expression.

## **waypoint\_stats**

*Usage:* waypoint\_stats

*Example:* waypoint\_stats

- Prints out some basic information for the waypoint pathing system. Primarily a debug tool, but useful if you want to see the total number of waypoints currently placed.

## **waypoint\_translate**

*Usage:* waypoint\_translate x[#] y[#] z[#]

*Example:* waypoint\_translate 10 0 0

- Translates(moves) all waypoints, or the currently selected waypoints by Vector3(x,y,z)

## **waypoint\_unlockall**

*Usage:* waypoint\_unlockall

*Example:* waypoint\_unlockall

- Unlocks all waypoints.

## **waypoint\_view**

*Usage:* waypoint\_view enable[1,0,true,false,on,off]

*Example:* waypoint\_view 1

- Enables or disables waypoint rendering.

## **waypoint\_viewfacing**

*Usage:* waypoint\_viewfacing enable[1,0,true,false,on,off]

*Example:* waypoint\_viewfacing 1

- Enables or disables rendering of the facing vector for waypoints if they have a facing vector.

# **Waypoint Flag List**

## **allies (ET specific)**

- An ET alias for the team2 flag. Often used with team doors, or at spawn points.

## **ammo**

- Flag this waypoint as having ammo nearby.

## **armor**

- Flag this waypoint as having armor nearby.

## **arty\_spot (ET specific)**

- A location for the bot to go to to look for arty\_target\_s or arty\_target\_d points

## **arty\_target\_d (ET specific)**

- Dynamic artillery strike target. An artillery using bot will watch this position and attempt to call artillery strikes ahead of an enemy that approaches this target location.

## **arty\_target\_s (ET specific)**

- Static artillery strike target. An artillery using bot will call an artillery on this position, whether there is anything there or not.

## **attack**

- A location a bot will go to and attack. Currently implemented as a simple goto location. No camping or anything else. To be improved in the future.

## **axis (ET specific)**

- An ET alias for the team1 flag. Often used with team doors, or at spawn points.

## **blockable**

- Flags the path between two waypoints as blockable.

Note: This flag has to be used in conjunction with additional flags to determine the type of checking that will be done for the blockable path! See above.

## **bridge (ET specific)**

- Flags a path between two waypoints that leads over a destructible bridge or ramp. Use with 'blockable' flag.

## **cappoint**

- A capture point for stolen items like documents, radar parts or whatever map makers come up with. This is where the bot carrying the stolen item should head to. Example: the boat in Venice.

## **climb**

- Flag this waypoint as climb, for ladders or other climbable objects.

## **closed**

- Flag this waypoint as closed, and therefor un-usable in path searches.

This flag is best added and removed from a script. Syntax example:

```
Wp.SetWaypointFlag( "waypoint_name1" , "closed" , false );  
Wp.SetWaypointFlag( "waypoint_name2" , "closed" , true );
```

## **crouch**

- Flag this waypoint as crouch, causing bots to crouch when they are within its radius.

## **defend**

- A location a bot will go to and defend. Currently implemented as a simple camp spot. To be improved in the future. Tip: If you add this flag don't forget to do the waypoint\_setfacing command for this waypoint. Otherwise bots might stare into walls or don't even look at objects to defend while camping.

## **detpack (ETF specific)**

- Flag this waypoint as a location for a Grenadier to lay a detpack.

## **door**

- Flag this waypoint as door, causing bots to be aware of special case door behaviors.

## **elevator**

- Flag this waypoint as elevator, causing the bot to use special elevator logic.

## **goto**

- Flag this waypoint as a GoTo point. This causes a GoTo goal to be created. It is scriptable like other goals, and is simply a goal that will cause the bots to move to the location.

## **health**

- Flag this waypoint as having health nearby.

## **inwater**

- Flag this waypoint as being in water. This should eventually be auto-detected on placement. Will eventually be used for smarter swimming logic.

## **jump**

- Flag this waypoint as jump, causing the bot to jump.

## **jumpgap**

- Flag this waypoint as jumpgap, causing the bot to check the ground ahead and jump near before gaps in the ground.

## **jumplow**

- Flag this waypoint as jumplow, causing the bot to check a short distance in front of the bot and jump over low obstacles such as small walls, crates, boxes...

## **mg42 (ET specific)**

- Flags this waypoint as an MG42 camping spot for soldier bots with mg42s. Not yet used in 0.66.

## **mine**

- Flags this waypoint as a spot to lay mines. Normally the bots will plant only one mine per mine spot.

## **mortar**

- (under wiki construction)

## **mortar\_target\_d**

- (under wiki construction)

## **mortar\_target\_s**

- (under wiki construction)

## **movable**

- Flag this waypoint as being movable. Not yet implemented, but in the future this will be used to implement waypoints that move along with the objects they are placed on, such as trains, platforms, etc...



## **pipetrapped (ETF specific)**

- Flag this waypoint as a location for a Grenadier to lay a pipe trap FROM.

## **pipetrapped2 (ETF specific)**

- Flag this waypoint as a location for a Grenadier to shoot his pipe trap TOWARDS.

## **prone**

- Flag this waypoint as prone, causing the bot to go prone and crawl along the path. Prone is also used in some circumstances as a stance to be in, such as with sniper points and such.

## **script**

- (under wiki construction)

## **sentry (ETF specific)**

- Flag this waypoint as a location for an Engineer to build a sentry.

## **sneak**

- Flag this waypoint as sneak, causing the bot to walk or sneak while in its radius.

## **snipe**

- Flag this waypoint as a snipe point, causing sniper bots to use it as a snipe point. Requires a valid facing for the waypoint. Often used in conjunction with the prone or crouch flags, and/or team flags. Read more about this flag on the sniper spots page.

## **sprint**

- Flag this waypoint as sprint, causing the bot sprinting along the path.

## **supplystation (ETF specific)**

- Flag this waypoint as a location for an Engineer to build a supplystation.

## **team1**

- Flag this waypoint as usable for team1 only.

## **team2**

- Flag this waypoint as usable for team2 only.

### **team3 (Not ET related)**

- Flag this waypoint as usable for team3 only.

### **team4 (Not ET related)**

- Flag this waypoint as usable for team4 only.

### **teamonly**

- Internal flag. Don't use this directly. Automatically set and un-set based on using other team flags.

### **teleport**

- Flag this waypoint as a teleport. The paths between 2 teleport waypoints is considered 0 length since navigation is considered instant.

### **underwater**

- Flag this waypoint as being under water. This should eventually be auto-detected on placement. Will eventually be used for smarter swimming logic.

### **wall (ET specific)**

- Flags a path between two waypoints that is blocked by a wall or similar obstacle. Use with 'blockable' flag.

### **waterblockable**

- Flags the path between two waypoints as blockable by water. Example: the caves in Oasis.

## **Community Waypoint Guidelines**

### **Note that this information may be partially outdated by the new waypoint database!**

Until there is the waypoint database running it would be nice if waypoint file / map script contributions could follow a few guidelines to make it easier for everybody to find the appropriate waypoint version.

- 1.) When submitting waypoint files please put them in a zip together with any additional information you want to include, e.g. readme.txt or whatever.
- 2.) While waypoint packs may have their uses I think it is generally better to submit single waypoints, cause in this way users will more easily find them without having to look at all the details of the package to find out if the needed waypoint file is in there. If you want to submit a package, please at

least include the map list in the download description.

3.) When uploading please name the download following this pattern: MAPNAME-BOTVERSION-WAYPOINTREVISION Example: oasis-0.51-1.0

4.) Please try to limit the number of contained subfolders.

5.) When contributing map scripts it is important that the waypoint file matches the map script or they may not work as intended. Therefore please either include them directly in the zip together with the matching waypoint file or if you want to submit them independently please reference the waypoint file needed in the download description.

6.) Don't upload vis files. They will be automatically generated on map start and should not be submitted together with the waypoint files.

A couple of concluding notes: We hope to get a real waypoint database going in the future that will ease up things a lot but I can give no date to when that'll be the case. If someone is interested we could use an "Official Waypoint Tester".

Hope those notes don't sound offensive. We're very thankful for all the user contributions we get and without you this whole project certainly wouldn't be as successful as it is.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Waypointing](http://omni-bot.com/wiki/index.php?title=Omni-bot_Waypointing)"

This page has been accessed 3,357 times. This page was last modified 11:45, 31 May 2008.

# Omni-bot Map Scripting

From Omni-bot Wiki

## Contents

- 1 Omni-bot Map Scripting
- 2 Map Scripts
  - 2.1 Examples
- 3 Global Functions
- 4 Scripting
- 5 Game Specific Map Scripting

## Omni-bot Map Scripting

Scripting plays an important and powerful part of Omni-bot. Scripting serves many uses in Omni-bot.

- Map Scripts - Help the bots play maps. Focus their attention on the goals that mean the most at a given time.
- Bot Scripts - Add bits of functionality to the bot. Something as simple as some basic chat communication up to a full blown feature can be scripted.

## Map Scripts

Map scripts are scripts that run after the navigation file for a map is loaded. Map scripts sole purpose is providing gameplay hints or direction to the bots to maximize the effectiveness of the bots playing a map.

Map scripts typically set up script callbacks for game events that will allow the script to adjust the bots focus to different goals, depending on the context of the event.

## Examples

- If a map required a team to assault and construct a bridge to proceed to the rest of the map, the map script would set up the initial state of the bot goals to focus around the bridge. Once the bridge was built, the map script might then shift focus on the next map objective.
- If a map required multiple points to be captured, like a capture and hold map, the

map script would control which control points are available for each team. When a team doesn't control one of the points, it might enable goals that deal with attacking and overtaking the point. If a team controls a point, those points will be made unavailable and in its place other goals that might deal with defense or holding the point.

In addition to setting up script trigger callback functions, map scripts are an ideal place to put any map specific special logic that needs special support or attention in the map.

## Global Functions

These functions will be called automatically by Omni-bot.

### OnMapLoad

Called after the waypoint file loads and goals are initialized.

```
global OnMapLoad = function() { // Do stuff };
```

### OnBotJoin

Called when a bot joins the game. A reference to the bot is passed into the function so that it is available for doing any map specific stuff to. For example, the map script may maintain a list of attacker and defender bots in separate tables. The bot could be added to one of the tables in the OnBotJoin function and removed in the OnBotLeave function.

```
global OnBotJoin = function( bot ) { // Do stuff };
```

### OnBotLeave

Called when a bot leaves the game.

```
global OnBotLeave = function( bot ) { // Do stuff };
```

## Scripting

For map scripting, the OnMapLoad function is the mostly used function. This function is the place to set up the initial state of the goals in the map, as well as trigger callback functions that should be called when events happen in the game.

## Game Specific Map Scripting

## Enemy Territory Map Scripting

### Quake 4

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Map\\_Scripting](http://omni-bot.com/wiki/index.php?title=Omni-bot_Map_Scripting)"

This page has been accessed 1,312 times. This page was last modified 04:09, 28 April 2008.

# Omni-bot Map Goals

From Omni-bot Wiki

## Contents

- 1 What are map goals?
- 2 How do they work?
  - 2.1 Common Map Goals
    - 2.1.1 Attack
    - 2.1.2 Defend
    - 2.1.3 CaptureTheFlag
    - 2.1.4 ReturnTheFlag
    - 2.1.5 Snipe
  - 2.2 Enemy Territory Map Goals
    - 2.2.1 BuildConstruction
    - 2.2.2 MountMg42
    - 2.2.3 RepairMg42
    - 2.2.4 PlantExplosive
    - 2.2.5 TakeCheckpoint
    - 2.2.6 ReviveTeammate
    - 2.2.7 MobileMg42
    - 2.2.8 MobileMortar
    - 2.2.9 PlantLandmine
    - 2.2.10 CallArtillery
    - 2.2.11 Flamethrower
    - 2.2.12 Panzer

## What are map goals?

Map goals are locations in the map that represent something for the bot to do. Capturing the flag is a map goal, an attack point is a map goal, a defend point is a map goal. They are called map goals because they are specific locations specific to the map. They may be generated from specific flags placed on waypoints, or they may be generated automatically by being detected by the bot during initialization.

Regardless of where they come from, they will all end up as map goals, and their availability will be dependent on how they are detected.

You can see a list of map goals by executing 'bot show\_goals' command in the game console. This will display all the goals and their availability status.

# How do they work?

Map goals are either detected automatically, or are a result of flagging certain waypoints with flags. Flagging a waypoint with the 'attack' flag for example will result in an attack map goal being created. This allows the waypointer to influence bot behavior by setting up some objectives manually. Alternately, many goal types are detected automatically whenever possible, depending on the game. Examples of this include the constructables, mg42s, and checkpoints in Enemy Territory, and flags and capture points in most supported games.

In most cases, the goal have all the information it needs without further input, and can operate correctly. Some goals have additional options that can be set that are specific to that goal type. For map goals that originate from waypoints, that means additional named properties being placed on the waypoints. For goals automatically detected, these properties must be set via script.

The lists below will outline each map goal and detail any additional properties that can be set on them.

## Common Map Goals

---

### Attack

Attack goals allow a bot to move to a position and attempt to camp it for a brief period of time before moving on.

- Properties
    - Stance - "Stand", "Crouch", or "Prone"
    - MinCampTime - minimum time, in seconds, to camp here - default 2 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 5 seconds
- 

### Defend

Defend goals allow a bot to move to a position and attempt to camp it for an extended period of time before moving on.

- Properties
  - Stance - "Stand", "Crouch", or "Prone"
  - MinCampTime - minimum time, in seconds, to camp here - default 5 seconds



- MaxCampTime - maximum time, in seconds, to camp here - default 10 seconds
- 

## CaptureTheFlag

The CaptureTheFlag goal handles various aspects of going to pick up an enemy flag if available, and take it to a capture or hold point.

- No Properties
- 

## ReturnTheFlag

This goal is generally dynamically generated when a flag is dropped if the owning team is capable of touching the flag to return it.

- No Properties
- 

## Snipe

- Properties
    - Stance - "Stand", "Crouch", or "Prone"
    - MinCampTime - minimum time, in seconds, to camp here - default 6 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 10 seconds
- 

## Enemy Territory Map Goals

---

### BuildConstruction

This goal handles the bot constructing various map features using the engineer pliers.

- Properties
  - Crouch - crouch while building, default false
  - Prone - prone while building, default false
  - IgnoreTargets - ignore targets while building, default false

---

## MountMg42

This goal allows the bot to mount a stationary Mg42 emplacement and fire on enemies.

- Properties
    - IgnoreTargets - ignore target to mount, default false
    - MinCampTime - minimum time, in seconds, to camp here - default 20 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 30 seconds
- 

## RepairMg42

This goal allows engineer bots to repair broken Mg42 emplacements.

- Properties
    - IgnoreTargets - ignore targets while repairing, default false
- 

## PlantExplosive

This goal handles both setting dynamite for engineers and setting satchel charges for covert ops on various targets around the map.

- Properties
    - IgnoreTargets - ignore targets while building, default false
- 

## TakeCheckpoint

This goal allows a bot to move to a checkpoint to attempt to capture it for their team.

- No Properties
- 

## ReviveTeammate

This goal allows a medic to revive a fallen teammate on the battlefield.

- No Properties
- 

### **MobileMg42**

- Properties
    - MinCampTime - minimum time, in seconds, to camp here - default 20 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 30 seconds
- 

### **MobileMortar**

This goal represents a position on a map to set up a mobile mortar, and to fire mortar rounds randomly along up to 12 pre-defined aim vectors. Useful for setting up a mortar barrage on an unseen area of the map.

- Properties
  - MortarAim0 - aim vector for a shot of the mortar
  - MortarAim1 - aim vector for a shot of the mortar
  - MortarAim2 - aim vector for a shot of the mortar
  - MortarAim3 - aim vector for a shot of the mortar
  - MortarAim4 - aim vector for a shot of the mortar
  - MortarAim5 - aim vector for a shot of the mortar
  - MortarAim6 - aim vector for a shot of the mortar
  - MortarAim7 - aim vector for a shot of the mortar
  - MortarAim8 - aim vector for a shot of the mortar
  - MortarAim9 - aim vector for a shot of the mortar
  - MortarAim10 - aim vector for a shot of the mortar
  - MortarAim11 - aim vector for a shot of the mortar

Only 1 aim vector is required, though up to 12 is supported. That means that however many that are available, the mortar goal will choose a random aim vector for each shot.

---

### **PlantLandmine**

This goal allows landmines to be built at certain areas of the map.

- No Properties

---

## CallArtillery

This goal represents a position to wait at in order to call in an artillery strike. Artillery strikes can be called on static artillery target goals, or dynamic ones. Dynamic artillery targets are areas the bot will watch for enemy activity before calling it. Static targets are called in immediately.

- Properties
    - MinCampTime - minimum time, in seconds, to camp here - default 1 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 2 seconds
- 

## Flamethrower

Specialized version of an attack goal, only allows the user to approach with the flamethrower equipped. This goal will likely be deprecated in the future in favor of additional weapon properties on the Attack and Defend goals.

- Properties
    - MinCampTime - minimum time, in seconds, to camp here - default 2 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 8 seconds
- 

## Panzer

Specialized version of an attack goal, only allows the user to approach with the panzerfaust equipped. This goal will likely be deprecated in the future in favor of additional weapon properties on the Attack and Defend goals.

- Properties
    - MinCampTime - minimum time, in seconds, to camp here - default 2 seconds
    - MaxCampTime - maximum time, in seconds, to camp here - default 8 seconds
- 
- 

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Map\\_Goals](http://omni-bot.com/wiki/index.php?title=Omni-bot_Map_Goals)"

This page has been accessed 293 times. This page was last modified 22:08, 24 August

# Omni-bot Scripting

From Omni-bot Wiki

Scripting is an important and powerful part of Omni-bot. Scripting serves many uses in Omni-bot. The two most important types of scripts are:

- Map Scripts - Help the bots play maps. Focus their attention on the goals that mean the most at a given time.
- Bot Scripts - Add bits of functionality to the bot. Something as simple as some basic chat communication up to a full blown feature can be scripted.

## The GameMonkey Scripting Language

Omni-bot uses an embedded scripting language called GameMonkey, which has a syntax very similar to that of C, but has many high-level features such as built-in associative maps and thread support. To get familiar with GM, it's a good idea to download the stand-alone distribution (gmsrc\_<version>.zip) from the official download page and try out some basic GM scripts without having to deal with the omni-bot environment. In the distribution, there is a Windows executable gmsrc\bin\gme.exe which you can use to run scripts.

For example, you can create a file `hello.gm` with the contents:

```
print("hello cruel world");
```

then run the script with `cmd.exe` (supposing that you extracted the distribution into C: and created the script in the same directory as `gme.exe`):

```
C:\gmsrc\bin>gme hello.gm
hello cruel world
```

This way, you can make sure your algorithms are working correctly before integrating them into omni-bots, making debugging much easier. There are also useful documentation files under `gmsrc\doc`, especially the language reference (`GameMonkeyScriptReference.pdf`) and a built-in functions reference (`gmdoc.html`).

**Sorry, this page is a stub at the moment.** For the time being, we refer you to the following pages:

- GameMonkey reference

- Omni-bot Script Reference
- Omni-bot Map Scripting
- Bot Library (mainly for bot scripts)
- Advanced Scripting Guide
- Community provided scripts

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Scripting](http://omni-bot.com/wiki/index.php?title=Omni-bot_Scripting)"

This page has been accessed 1,112 times. This page was last modified 04:02, 28 April 2008.

# Omni-bot Weapon Scripting

From Omni-bot Wiki

## Contents

- 1 Weapon Scripting
- 2 Intro
- 3 Weapon Scripts
- 4 Custom Functions

## Weapon Scripting

It is recommended you open **weapon sample.gm** in a separate window to follow along. This is a sample weapon script that contains all the properties and functions of a script defined weapon.

## Intro

Prior to Omni-bot 0.6, all the information that the bot used to know how to use the weapons from a game was contained within the bot dll. This was very inflexible not only for tweaking, but for 3rd party mods to customize to support additional weapons that weren't included as part of the default Omni-bot.

## Weapon Scripts

Weapon scripts are located in the *scripts/weapons* folder of the Omni-bot installation folder, under the relevant game. For example, for Enemy Territory it is *et/scripts/weapons*.

All weapon scripts must have a prefix of *weapon\_* in the filename. This prefix is what the bot uses to load all weapon scripts that exist in that folder.

During the initialization of the bot, all files in the *scripts/weapons* folder starting with *weapon\_* will be loaded to create a template for a weapon. For each weapon that it loaded, *weapon\_defaults.gm* is first executed so that common parameters can be put in there for ease of use. After *weapon\_defaults.gm*, the weapon script is then executed in order to define the rest of the weapon.

*weapon\_defaults.gm* is the best reference for available weapon properties. It will have all available properties defined as well as explanations for them in comments in the script.

Weapons currently support 2 fire 'modes'. Primary, and Secondary. This is so games that support alt-firing can in the future be easily supported. For most games, such as ET and Quake 4, Primary fire is normally all that's used. Most weapon properties are actually properties of their fire modes, as you will see in the scripts.

When each weapon script is executed, the weapon is passed to the script as the 'this' parameter, so all properties will be set on 'this' in one way or the other. Again, the best way to understand this is to look at `weapon_defaults.gm`

We'll go over each one in this article from the top down, using the `weapon_defaults.gm` script as our reference, and pasting each line.

```
this.Name = "Shotgun";
```

The name of a weapon is used mainly for human readable printouts and debugging.

```
this.WeaponId = -1;
```

This is one of the most important settings in the whole weapon script. This value is the number that maps the script defined weapon with a number within the game. If these don't match up, you will likely see the bot trying to equip the wrong weapons, or nothing at all if the value doesn't represent an existing weapon. The only people that would normally have to deal with this are those that are adding new weapon scripts to support custom weapons, such as those in Jaymod and etpub.

```
this.MinUseTime = 2.0;
```

This value is a minimum usage time for the weapon. This is the minimum time the bot is allowed to equip a weapon. Primarily useful to smooth out rapid weapon switching that can occur when 2 weapons overlap in desirability calculations.

```
this.PrimaryFire.WeaponType = "instant";
```

The weapon type can be one of a few different types. "melee", "instant", "projectile". This type helps determine mainly how the bot should aim the weapon. For example, projectile weapons will take into account projectile gravity and projectile speed to calculate a valid trajectory. Projectile weapons will also attempt to lead the target to account for the travel time for the projectile. Accurate trajectories are calculated from several of the values defined in script, as well as the current gravity the game is using, so it's important that they are accurate.

```
this.PrimaryFire.MaxEquipMoveMode = "run";
```



This represents the fastest the bot can move while equipping the weapon. An example of use for this is the zoomed sniper rifle in ET. If you move faster than a walk, the weapon will automatically un-scope. Setting this property to "walk" will cap the bot to walking as long as that weapon is equipped.

```
this.PrimaryFire.ShootButton = BTN.ATTACK1;
```

This value is simply the button that is pressed to fire the appropriate fire mode.

```
this.PrimaryFire.ProjectileSpeed = 1000;
```

This is a property for projectile weapons. This value only matters when the WeaponType is set to "projectile". This is the speed at which the projectile fires from the weapon. It is used for both trajectory and leading calculations.

```
this.PrimaryFire.RequiresAmmo = true;
```

Some weapons don't require ammo, such as melee weapons. This value determines if the bot cares about checking the ammo for the weapon.

```
this.PrimaryFire.WaterProof = false;
```

This property determines if the gun is usable under water. If a weapon is not waterproof it will not be considered for use when the bot is underwater.

```
this.PrimaryFire.SplashDamage = false;
```

This property means the weapon produces splash damage, so that the bot can be a little more aware of the potential for friendly fire.

```
this.PrimaryFire.InheritsVelocity = false;
```

This property means the weapon projectile will inherit the velocity of the shooter when it is fired. Very few games actually do this, though some do extensively, like tribes. This property is there to support it should the need arise.

```
this.PrimaryFire.ProjectileGravity = 0.0;
```

This property is a gravity multiplier for the weapons projectile, and is only used for "projectile" WeaponType. This is one of the key parts of calculating trajectory. Gravity is calculated by multiplying the games gravity value by this projectile gravity. Normally this

would be 1.0 or 0.0, but some games don't use the full gravity for projectiles. Projectile weapons that aren't effected by gravity would simply set this to 0.0, while weapons that are effected by gravity, like grenades might have 1.0, meaning they get the full effect of gravity applied to them. 0.5 would mean only half the effect of gravity is used.

```
this.PrimaryFire.NeedsInRange
```

This property causes the MinRange and MaxRange properties to be treated as a rigid range for use, and in the future the bot should attempt to actively stay within this range. Currently doesn't do this though.

```
this.PrimaryFire.MinRange
```

```
this.PrimaryFire.MaxRange
```

The Min and Max Range to use for the NeedsInRange check.

```
this.PrimaryFire.MinChargeTime
```

```
this.PrimaryFire.MaxChargeTime
```

These properties define a range that the weapon should be charged. When firing the weapon, a random value between these Min and Max properties will be generated to charge the weapon. Some examples of a charged weapon, Half-life Gauss gun, Team Fortress Sniper Rifle.

```
this.PrimaryFire.DelayAfterFiring
```

This is a time value(in seconds) which the bot will ignore this weapon as a weapon choice after firing it once.

```
this.PrimaryFire.IgnoreReload
```

This property tells the bot to not consider this fire mode when looking for weapons that need reloading.

```
this.PrimaryFire.FireOnRelease
```

This property alters the bots usage of the weapon so that the actual firing of the weapon occurs on the release of the button as opposed to the pressing of the button. This sort of behavior is normally on weapons that are charged, such as the previously mentioned Half-life Gauss gun and Team Fortress Sniper Rifle.

```
this.PrimaryFire.MaxAimError = Vector2(0, 0);
```

This property is carried over from previous bot versions. This value represents the maximum aiming error that should be applied to a weapon when aiming at a target. It is a 2 dimensional vector, which represents horizontal(x), and vertical(y) aim error. A random value between -MaxAimError, and MaxAimError will be generated to determine the error at any given time, with periodic re-calculations(every 2-3 seconds or so).

```
this.PrimaryFire.AimOffset = Vector3(0, 0, 0);
```

This property is an offset that is applied to the targets position when calculating the aim position. It gets applied before the aim error. It is a 3 dimensional vector that represents an offset in world space. Normally this is used to offset the default aim position due to differences in the origin of players across different games. For example, in ET, the player origin is around his waist area, whereas some other games commonly have the player origin at their feet.

```
this.PrimaryFire.SetTargetBias(CLASS.VEHICLE_HVY, 0);  
this.PrimaryFire.SetTargetBias(CLASS.BREAKABLE, 0);
```

This property allows the weapon to be configured with a bias towards certain target classes. Sometimes, some weapons may not be capable of hurting certain target types, like a pistol versus a tank. This property can be used to set this up. This property can be set independently for each target class. In this example, the weapon is set to a bias of 0 for CLASS.VEHICLE\_HVY, effectively disabling the weapon for use against CLASS.VEHICLE\_HVY. The 2nd line also disables the weapon for use against breakable targets(windows, fences, etc in ET). Since this is the default weapon script, all weapons will initially be set up like this, allowing individual weapons to be set up differently, so we could for example allow only pistols to shoot at CLASS.BREAKABLE.

```
this.PrimaryFire.DefaultDesirability = 0.0;
```

This property is the default desirability for the weapon mode. Default desirability is used whenever the bot doesn't have a target. This is primarily what determines which weapon the bot will have out while it's running around with no target. In Enemy Territory, we might set pistols with a higher DefaultDesirability than heavy weapons due to the dramatic player slowdown that equipping heavy weapons has. This would let the bot run around faster with the pistol out, but most likely switch to his heavy weapons when he see's a target.

```
this.PrimaryFire.SetDesirabilityRange(0, 100000, 0.0);
```

This is one of the most important functions for setting up when the bot should use this weapon. This function maps a distance range to a priority for the weapon. Here's a few examples to clarify this.

```
this.PrimaryFire.SetDesirabilityRange(0, 500, 0.5);
```

For a target 0-500 distance, the desirability for this weapon is 0.5

```
this.PrimaryFire.SetDesirabilityRange(500, 5000, 0.75);
```

For a target 500-5000 distance, the desirability for this weapon is 0.75.

Clearly in this example this weapon is preferred at longer range. Care must be taken when choosing these values. Often you will need to look at the desirability ranges of other weapons in order to tweak the values so that weapons don't fight over preference by overlapping too much.

When the bot is calculating the desirability to use a weapon, the first thing that happens are the quick checks that can disqualify a weapon for selection, such as the bot being in water but the weapon not being waterproof, after those simpler checks, he gets the range desirability from the above functions based on the range of his target, so if a target were 750 units away, the range desirability would be 0.75. After that, any further biases are calculated. Target bias and the bots own bias towards a weapon are multiplied with the 0.75 to give the final desirability for the weapon.

**Note:** In the future I hope to have an easier way to set the range desirability, preferably by graphically editing a curve that maps a smooth desirability to a range.

## Custom Functions

Some weapons may need more control over their desirability calculation and/or how the bot fires them. This is where script callbacks can be useful. As seen in the weapon\_defaults.gm, there are 3 script functions that are commented out. These represent 3 available callbacks that can be used to modify how the weapon is used above and beyond what the built in parameters cover.

```
this.PrimaryFire.CalculateDefaultDesirability = function(bot)
{
    return 0;
};
```

```
this.PrimaryFire.CalculateDesirability = function(bot, targetInfo)
{
    return 0;
};
```

This function simply calculates the default desirability and normal desirability of the weapon. The 2nd is passed the target info for the bots current target, should you need to use information about the target. Rarely should these functions need to be scripted unless you absolutely need to perform some special logic as part of the process. Be aware that all internal calculations still apply, as a way to reduce the frequency of making the script callbacks.

```
this.PrimaryFire.CalculateAimPoint = function(bot, targetInfo)
{
    return targetInfo.LastPosition;
};
```

This callback lets the script calculate an aim point for the weapon. Defining this function will likely have significant impact on the cpu usage due to frequency of calls, and likewise should be avoided when possible.

More callbacks will likely be defined in the future as the need arises, though generally you should be able to do nearly all you need to do without the callbacks, and should attempt to do so for efficiency.

If you come up with common weapon functionality that isn't covered in the properties above, feel free to let me know. If it's a common occurrence I will likely add a property for it.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Weapon\\_Scripting](http://omni-bot.com/wiki/index.php?title=Omni-bot_Weapon_Scripting)"

This page has been accessed 747 times. This page was last modified 04:04, 28 April 2008.

# Omni-bot Routing

From Omni-bot Wiki

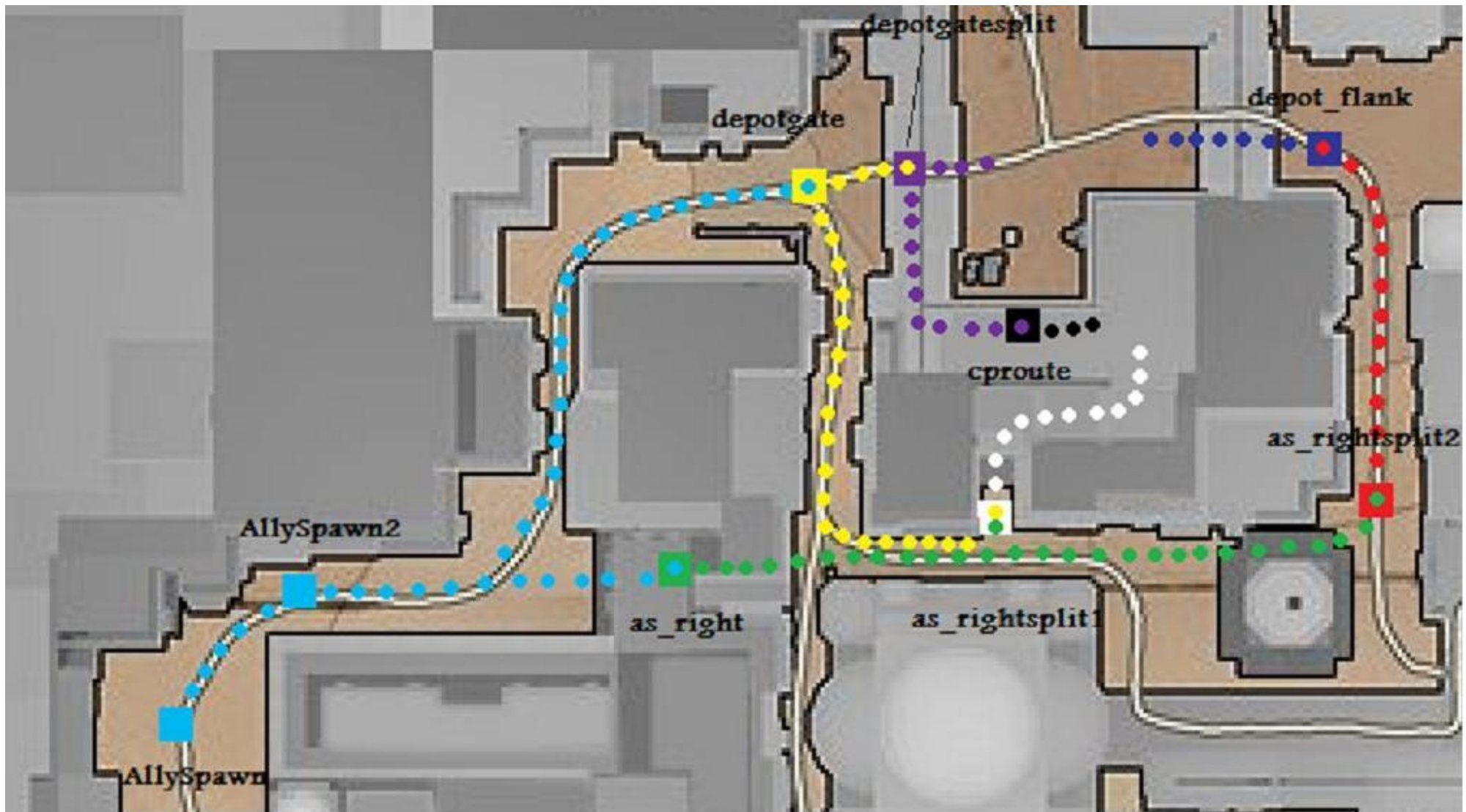
## Contents

- 1 Routing Overview
- 2 Setting It Up
  - 2.1 Step 1: Setting up the script
  - 2.2 Step 2: Adding route nodes
  - 2.3 Step 3: Scripting the routes
  - 2.4 Step 4: Testing the routes
  - 2.5 Step 5: Adding priorities to routes
  - 2.6 Step 6: Copying routes
  - 2.7 Step 7: Toggling routes
  - 2.8 Step 8: Ask for help

## Routing Overview

Until version 0.65, omni-bot path selection relied on the shortest path to a goal. While this typically provided for very focused attacks, game play became very predictable. The solution is Routing. Routing provides alternate goals for the bots to go to before heading to their main goal. When positioned correctly, these alternate goals provide the effect of bots choosing realistic and varied paths to their main goals.

Routing requires some setup in script and in the waypoints, but the results are worth the effort. The objective of this article is to provide an overview of how the routing system works and to clearly define how to set it up by walking through a working example on Goldrush.



In this picture, Squares are route nodes and the Dots are potential paths. The color of the Dots correspond to the color of the Route nodes. Once they reach a route node, they will travel along the path(s) of the same color IF the route node is set up for their current main goal.

Walking through the picture, let's say the tank construction goal is set up for an allied engineer within the AllySpawn route nodes, and that the engineer has the tank construct as its goal. The engineer will randomly choose whether to go to the green route node (as\_right) or the yellow route node (depotgate). For the purpose of this example, let's say the bot chose to go to the green route node. From there, he will randomly choose to go to the white or red route node. The bot will continue to make these decisions until there are no route nodes left to go to. From that point, the bot will take the shortest path to the tank construct goal. As you can see, this opens up several different points of attack for the bots; which creates a sense of realism in terms of predictability. Defenses will now need to be set up to be able to react to all of



the entry points rather than focusing on easily predicted choke points if the shortest path is always used.

## Setting It Up

Setting up Routes involves waypointing and scripting. This section will take you step by step through the process of setting up the routes in the Goldrush example.

### Step 1: Setting up the script

The first requirement for routing is that the routes are defined in the map script. The method to do this consists of building a table for the routes and calling a utility function that will register the routes.

If a map script does not exist yet, open up your favorite text editor and add the following to a new file:

```
global OnMapLoad = function()  
{  
};
```

Save the file as <mapname>.gm in your Omni-Bot\et\nav folder. You are now ready to start building the routes. Inside the OnMapLoad function is where the route definitions are placed and they all start with a basic layout:

```
global OnMapLoad = function()  
{  
    Maproutes =  
    {  
    };  
  
    Util.Routes(Maproutes);  
};
```

You now have a Maproutes table defined and are registering everything within that table with the Util.Routes function.

### Step 2: Adding route nodes

Route nodes are fairly straight forward. They are placed in positions where a bot might receive the goal you want them to use alternate paths for. The bot MUST be inside the route nodes radius when receiving the main goal for it to work. Spawn points are good spots for route nodes because bots will always receive a goal when they spawn.

Route nodes have three important properties; route flag, waypoint name, and radius. Setting up the first route node in the Goldrush example, choose a waypoint in the allied spawn area that won't effect the bots navigation if it has a large radius. Once the node is selected, issue the following commands in console:

```
/bot waypoint_addflag route  
/bot waypoint_setname AllySpawn
```



```
/bot waypoint_setradius 350
```

The radius of the route nodes at spawn positions are very critical because the bots must be inside its radius for it to work. Be sure that the radius covers the entire spawn area. You can use any name you want for the route nodes, but they should be somewhat intuitive.

For routing to be effective, you will need at least two defined because once they reach the route node, they will take the shortest path to their main goal UNLESS there is another route node set up for them to go to. Continuing with our example, create a route node where the green square is in the picture and name it as\_right:

```
/bot waypoint_addflag route  
/bot waypoint_setname as_right
```

You now have two route nodes; meeting the requirement for working routes. In our example, we want the bots to use two paths from the spawn, so add the route node depicted by the yellow square at the depot gate:

```
/bot waypoint_addflag route  
/bot waypoint_setname depotgate
```

Be sure to save the waypoints at this point. The next step is to set up the goals and route node selection in the script.

### Step 3: Scripting the routes

The Maproutes table is set up with the following format:

```
Maproutes =  
{  
    GOALNAME =  
    {  
        ROUTENAME = {},  
    },  
    GOALNAME2 =  
    {  
        ROUTENAME = {},  
    },  
};
```

If you are new to map scripting, this may appear a bit intimidating. The key is to follow the steps and syntax exactly, then let the understanding of the syntax come naturally. If you are comfortable with gm scripting, the routing set up consists of nested tables.

The goal name must match the output from show\_goals and the route name is the waypoint name prepended with ROUTE\_. For the Goldrush example, set up the tank construct routing as follows:

```
global OnMapLoad = function()  
{
```

```

    Maproutes =
    {
        MAP_CONSTRUCTION_tank_construct =
        {
        },
    };

    Util.Routes(Maproutes);
};

```

You have just added a map goal to the Maproutes table. The syntax of the goal name is VERY important. Be sure to double check the goal name with /bot show\_goals. Once you are sure the goal name is correct, you can add the route nodes:

```

global OnMapLoad = function()
{
    Maproutes =
    {
        MAP_CONSTRUCTION_tank_construct =
        {
            ROUTE_AllySpawn = {},
        },
    };

    Util.Routes(Maproutes);
};

```

The AllySpawn route node is now a valid route node for the tank construct goal. If a bot spawns within its radius and receives the tank construct as a goal, it will use that route node. In and of itself, this route node will not do anything because at spawn the bot is already at it. So let's add the two path options we want to give the bots from the Allied Spawn:

```

global OnMapLoad = function()
{
    Maproutes =
    {
        MAP_CONSTRUCTION_tank_construct =
        {
            ROUTE_AllySpawn =
            {
                ROUTE_as_right = {},
                ROUTE_depotgate = {},
            },
        },
    };

    Util.Routes(Maproutes);
};

```

It's important to understand what we just did with this:

```
ROUTE_AllySpawn =  
{  
    ROUTE_as_right = {},  
    ROUTE_depotgate = {},  
},
```

The bot reaches the AllySpawn route node by default when it spawns; and because we have added the two routes inside the brackets, the bots will randomly choose between them. At this point, you have randomized routes for Allied engineers with the tank construct goal. Half the time they will choose to go to route as\_right and half the time they will choose to go to route depotgate.

Let's make them even less predictable. Looking at the example picture, you can see that we want them to have a couple different options once they reach the route nodes we just set up. Repeating step 2, add the rest of the route nodes depicted by the colored squares.

The scripting for the path options in the picture is as follows:

```
global OnMapLoad = function()  
{  
    Maproutes =  
    {  
        MAP_CONSTRUCTION_tank_construct =  
        {  
            ROUTE_AllySpawn =  
            {  
                ROUTE_as_right =  
                {  
                    ROUTE_as_rightsplit1 = {},  
                    ROUTE_as_rightsplit2 =  
                    {  
                        ROUTE_depotflank = {},  
                    },  
                },  
            },  
            ROUTE_depotgate =  
            {  
                ROUTE_cproute = {},  
                ROUTE_depotgatesplit = {},  
                ROUTE_as_rightsplit1 = {},  
            },  
        },  
    },  
};  
  
Util.Routes(Maproutes);  
};
```

The AllySpawn2 route node was placed because engineers do not always receive construct goals immediately

at spawn. It can be added to the script as follows:

```
global OnMapLoad = function()
{
    Maproutes =
    {
        MAP_CONSTRUCTION_tank_construct =
        {
            ROUTE_AllySpawn =
            {
                ROUTE_as_right =
                {
                    ROUTE_as_rightsplit1 = {},
                    ROUTE_as_rightsplit2 =
                    {
                        ROUTE_depotflank = {},
                    },
                },
                ROUTE_depotgate =
                {
                    ROUTE_cproute = {},
                    ROUTE_depotgatesplit = {},
                    ROUTE_as_rightsplit1 = {},
                },
            },
            ROUTE_AllySpawn2 =
            {
                ROUTE_as_right =
                {
                    ROUTE_as_rightsplit1 = {},
                    ROUTE_as_rightsplit2 =
                    {
                        ROUTE_depotflank = {},
                    },
                },
                ROUTE_depotgate =
                {
                    ROUTE_cproute = {},
                    ROUTE_depotgatesplit = {},
                    ROUTE_as_rightsplit1 = {},
                },
            },
        },
    };

    Util.Routes(Maproutes);
};
```

Notice that it is just a copy of the AllySpawn route, but positioned so any engineer that may receive

the construct goal a little later than right at spawn will use the routing as well.

## Step 4: Testing the routes

Testing the routes is by far the most important step. It is recommended that you test routes individually as you start to learn the routing system. Once you are comfortable with the setup, you may want to add routes for an entire phase and then test them to make the process a bit faster.

If the routing is not working as expected, there are some basic troubleshooting options. It's important to remember the criteria for the routes; the bot must be within the radius of the route node when receiving the goal, and the route node must be set up to support the goal. The basic troubleshooting steps are:

1. Check in console when the map loads for any error messages. It should list any goal names not found when initializing the routes.
2. Check the radius of the route where the bot receives the goal
3. Make sure the bot can reach the goal
4. Make sure the bot is getting the expected goal

For numbers 2 and 3, consider using the command /bot debugbot all goals with only one bot connected.

In console, it should give success / failure messages for short term and long term goals. An example of a known problem is if there are priority type goals (like radar parts stealing) that are active when they aren't reachable. The solution is to set them to be not active in OnMapLoad if they are unreachable.

## Step 5: Adding priorities to routes

In some cases, you may want bots to use a certain route more often than others. This can be done by adding a "weight" to the route table:

```
ROUTE_as_right =
{
    Weight = 2,
    ROUTE_as_rightsplit1 = {},
    ROUTE_as_rightsplit2 =
    {
        Weight = 2,
        ROUTE_depotflank = {},
    },
},
```

In this example, the bots will randomly choose to go to route as\_right twice as much as route depotgate. And then they will choose to go to route as\_rightsplit2 twice as much as route as\_rightsplit1.

## Step 6: Copying routes

If you have more than one goal that you want to share the same routing with, you can copy the routing. In the Goldrush example, say we have some attack goals in the yard that we want to route to. Rather than writing out the tables for each attack goal, we can copy them like this:

```
Maproutes.ATTACK_Depot_1 = Maproutes.MAP_CONSTRUCTION_tank_construct;  
Maproutes.ATTACK_Depot_2 = Maproutes.MAP_CONSTRUCTION_tank_construct;
```

The key point to remember when doing this is that the route table that exists must be on the right side of the equation. In this case, we have created routes for two attack goals that are the same as our example tank construct routing.

These should be placed in OnMapLoad as well, below the Maproutes table and above the Utility function that registers the routes:

```
global OnMapLoad = function()  
{  
    Maproutes =  
    {  
        MAP_CONSTRUCTION_tank_construct =  
        {  
  
        },  
    },  
};  
  
Maproutes.ATTACK_Depot_1 = Maproutes.MAP_CONSTRUCTION_tank_construct;  
Maproutes.ATTACK_Depot_2 = Maproutes.MAP_CONSTRUCTION_tank_construct;  
  
    Util.Routes(Maproutes);  
};
```

## Step 7: Toggling routes

Routes are treated very similar to map goals. They will show up in your list if you do a /bot show\_goals. They can be turned on and off via scripting with the SetAvailableMapGoals function. As an example, say we have route that we want opened up when barrier is destroyed, then closed when that barrier is constructed. This can be done inside the OnTriggers for those events:

```
Barrier_construct = function (trigger)  
{  
    SetAvailableMapGoals( TEAM.AXIS, false, "ROUTE_routname" );  
};  
  
Barrier_destroyed = function (trigger)  
{  
    SetAvailableMapGoals( TEAM.AXIS, true, "ROUTE_routname" );  
};
```

## Step 8: Ask for help

If you are having trouble with the syntax, set up, or understanding the routing system, ask for help. This guide is by no means comprehensive and complex maps will offer different challenges. Please post in the forums if you have questions / problems / bug reports.

# Omni-bot Script Goals

From Omni-bot Wiki

## Contents

- 1 Script Goals
- 2 Intro
- 3 Script Goals
- 4 Script Goal Functions
- 5 Game Specific Script Goals

## Script Goals

Script goals are a feature of 0.7 and so are not yet available in the latest public release.

## Intro

New to Omni-bot 0.7, scripted goals have gotten a complete makeover, along with the internal goal system.

Several large changes are the result of this new system

- **Drop-in goals** - In 0.66 scripted goals had to be added to the bot profile(def\_bot.gm). This sucked, so it's gone. Script goals are now independently dropped in to a folder and they are loaded automatically. This also makes it much easier to share custom bot goals from users.
- **Smaller, faster, easier, more reliable** - More helper functions, events, and callbacks have been added, resulting in easier to write scripts that are smaller and run faster. What previously needed yield loops to accomplish is now done with events, making it much faster. Reliability is much improved. No more being unsure of missed cases where your goal can leave the bot in an unknown state(the dreaded standing around bug of  $\leq 0.66$ ).
- **Easier Debugging** - The new Debug Window contains a high level picture of the behavior tree of the bot, using icons and colors to represent the activity that is going on in the bot. At a glance, you can see what behavior the bot is running, disable or tweak properties of them in real time, and enable debug drawing of specific states.

# Script Goals

Script goals are located in the *scripts/goals* folder of the Omni-bot installation folder, under the relevant game. For example, for Enemy Territory it is *et/scripts/goals*. They can alternatively be placed in *global\_scripts/goals* where they will be loaded in any game.

All script goals must have a prefix of *goal\_* in the filename. This prefix is what the bot uses to load all script goals that exist in that folder.

During the initialization of the bot, all files in the *global\_scripts/goals* and *scripts/goals* folder starting with *goal\_* will be loaded **once** to create a template for a goal. When a bot is added to the game, all script goals that were previously loaded are cloned and added to the bots behavior tree, which can be seen in its entirety in the Debug Window.

When the *goal\_\** scripts are run, the value of *this* in the scripts is the script goal object itself, and it is up to the script to define the necessary functions for the goal.

Anyone familiar with Finite State Machines (FSM), will recognize the very state machine like setup of script goals.

## Script Goal Functions

Script Goal Functions are documented in the ScriptGoal script reference page.

## Game Specific Script Goals

Certain games may come with script goals already implemented. They implement additional functionality for the game they were written for. They may also be used as reference for writing your own.

- Doom 3 Script Goals
- Enemy Territory Script Goals
- Fortress Forever Script Goals
- Quake 4 Script Goals
- RTCW Script Goals
- Common Script Goals

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Script\\_Goals](http://omni-bot.com/wiki/index.php?title=Omni-bot_Script_Goals)"

This page has been accessed 517 times. This page was last modified 16:26, 1 May 2008.



# Omni-bot Command Reference

From Omni-bot Wiki

## Contents

- 1 Console Commands
  - 1.1 addbot
  - 1.2 balanceteams
  - 1.3 debugbot
  - 1.4 debugtriggers
  - 1.5 dontmove
  - 1.6 dontshoot
  - 1.7 draw\_goals
  - 1.8 drawblocktests
  - 1.9 drawthreats
  - 1.10 help
  - 1.11 kickall
  - 1.12 kickbot
  - 1.13 maxbots
  - 1.14 minbots
  - 1.15 nav\_logfailedpath
  - 1.16 nav\_showfailedpath
  - 1.17 navsystem
  - 1.18 reload\_weapons
  - 1.19 revision
  - 1.20 script\_collect
  - 1.21 script\_debug
  - 1.22 script\_run
  - 1.23 script\_runfile
  - 1.24 script\_stats
  - 1.25 show\_bb
  - 1.26 show\_goals
  - 1.27 showprocesses
  - 1.28 stopprocess
  - 1.29 update\_all\_nav
  - 1.30 update\_nav
  - 1.31 version

## Console Commands

Omni-bot comes with a wide range of utilities in the form of console commands for general purpose useage as well as debugging tools for waypointing.

## Executing a command

- In ET, ETF, or other Quake 3 engine based games - /bot <command>
- In Quake4, Fortress Forever, and other Half-life 2 or Doom 3 engine based games, the / isn't required

## addbot

Adds a bot to the game.

syntax: bot addbot <team> <class> <name>

example: /bot addbot 2 1 bob will add a soldier named bob to the allied team in ET

## balanceteams

Forces bots to keep teams balanced.

syntax: bot balanceteams 1/0

example: bot balanceteams 1 turns on bot balancing

notes: by default, omni-bot will keep teams balanced when adding bots unless bots are added specifically to a team

## debugbot

Enables debugging output on a specific bot.

syntax: bot debugbot botname debugtype

example: bot debugbot all goals will give debug output for all connected bots goals

debug types: log, move, aim, goals, sensory, brain, weapon, script, events, fpinfo

## debugtriggers

Prints triggers to console.

syntax: bot debugtriggers

usage: used for checking or identifying trigger names to be used with map scripting. when turned on, you will see

messages about events as they occur in the map:

<+++> Trigger: TagName: The Tank has been repaired! Action: announce Entity: 0x7016dd8c  
Activator: 0

<+++> means that a callback (trigger) is set up and has been called, while <---> means no callback is associated

TagName is the string to use for the OnTrigger function in the map script

Action is the category in which the event is recognized

notes: this is a toggle. issuing the command a second time will turn it off.

## dontmove

Enables/disables all bot movement ability.

syntax: bot dontmove true/false/1/0/yes/no

example: bot dontmove 1 will stop all bots from moving.

## dontshoot

Enables/disables all bot shooting ability.

syntax: bot dontshoot true/false/1/0/yes/no

example: bot dontshoot 1 will stop all bots from shooting.

## **draw\_goals**

Draws debug information for all mapgoals.

syntax: bot draw\_goals on/off goalname

example: bot draw\_goals on will draw squares around all objective goals and highlight the radius of attack / defend goals

notes: the goalname parameter is optional and supports expressions. i.e. bot draw\_goals on ATTACK.\* will highlight all of the attack goals in the map.

## **drawblocktests**

Enables drawing of blockable line tests.

syntax: bot drawblocktests 1/0

example: bot drawblocktests 1 turns on drawing of block tests

usage: will draw block test lines in addition to the ones drawn in waypoint mode.

## **drawthreats**

Enables drawing of detected threats.

syntax: bot drawthreats 1/0

example: bot drawthreats 1 turns on drawing of threats

usage: when enabled, potential threats are outlined (i.e. player entities)

## **help**

List all bot commands available in console.

syntax: bot help

## **kickall**

Kick all bots from the game by name.

syntax: bot kickall

## **kickbot**

Removes a bot from the game.

```
syntax: bot kickbot <botname>
example: bot kickbot [BOT]Aimless
```

## **maxbots**

The maximum players to keep in play.

```
syntax: bot maxbots #
example: bot maxbots 20
```

## **minbots**

The minimum players to keep in play.

```
syntax: bot minbots #
example: bot minbots 2
```

## **nav\_logfailedpath**

Saves info about failed path attempts for debugging.

## **nav\_showfailedpath**

Render a failed path by its index.

## **navsystem**

Creates a navigation system of a specified type.

## **reload\_weapons**

Reloads the weapon database from script files on disc.

## **revision**

Shows the revision the bot dll was built from.

```
syntax: bot revision
```

## **script\_collect**

Performs a garbage collection.

```
syntax: bot script_collect
usage: for debugging use only
```

## **script\_debug**

Enables/disables debug messages in the scripting system.

syntax: bot script\_debug 1/0/on/off

example: bot script\_debug 1 turns on debugging information in the script system.

usage: when enabled, if an error occurs in script execution, it will be printed to the console.

## script\_run

Executes a string as a script snippet.

syntax: bot script\_run <string>

example: bot script\_run "print(GetGameTimeLeft())"

## script\_runfile

Executes a specified script file.

syntax: bot script\_runfile <filename>

example: bot script\_runfile testing.gm

## script\_stats

Shows scripting system memory usage/stats.

syntax: bot script\_stats

usage: displays memory use and garbage collection stats in console

## show\_bb

Shows the contents of the global blackboard.

syntax: bot show\_bb

## show\_goals

Prints out the names of each goal.

syntax: bot show\_goals <optional> <optional p>

example: bot show\_goals DEFEND.\* will list all defend goals in console

example: bot show\_goals .\* p will list all goals and their sub priorities (for >= 0.7)

notes: this command supports expressions for the optional parameter

Explanation of goal-list-entry by example:

```
33: FLAG_the_War_Documents -> 0000 serial 3 priority 0.55
    ALLIES AXIS SOLDIER MEDIC ENGINEER LIEUTENANT 1.80
```

<goal-no>: <goal-name> -> <team-availability> serial <serial> bias <bias>

goal-no: number of goal

goal-name: name of goal  
team-availability: availability team1 team2 team3 team4  
serial: serial  
bias: current bias value

The optional p parameter displays properties set with the SetGoalPriority function which enables Team and Class properties to be set.

notes: team-availability shows boolean value for each team  
team3 and team4 are obsoleted in ET, since ET only has 2 teams:  
TEAM.AXIS and TEAM.ALLIES or TEAM.TEAM1 and TEAM.TEAM2.

## **showprocesses**

Shows a process by its name.

syntax: bot showprocesses

usage: for debug use only

## **stopprocess**

Stops a process by its name.

syntax: bot stopprocess

usage: for debug use only

## **update\_all\_nav**

Attempts to download all nav files from the database, including updating existing files.

syntax: bot update\_all\_nav

notes: Be sure to have copies of all custom waypoints and scripts that you may be working on!

## **update\_nav**

Checks the remote waypoint database for updated navigation.

syntax: bot update\_nav

notes: Be sure to have copies of all custom waypoints and scripts that you may be working on!

## **version**

Prints out the bot version number.

syntax: bot version

# Omni-bot Script Reference

From Omni-bot Wiki

## Omni-bot Script Reference.

### Global Script Constants

AMMO • BB • BONE  
BTN • BUY • CAT  
CLASS • CONTENT •  
DEBUG  
ENTFLAG • EVENT • ITEM  
POWERUP • PROFILE •  
SKILL  
TEAM • TRACE • VOICE  
WEAPON

### Global Script Functions

Bot Library  
Math Library  
System Library

### Scripting Types

AABB  
Bot  
Blackboard  
MapGoal  
Matrix3  
ScriptGoal  
TargetInfo  
Timer  
TriggerInfo  
Vector3  
Weapon

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_Script\\_Reference](http://omni-bot.com/wiki/index.php?title=Omni-bot_Script_Reference)"

This page has been accessed 1,995 times. This page was last modified 09:39, 16 July 2008.

# Omni-bot F.A.Q.

From Omni-bot Wiki

## Contents

- 1 Omni-Bot F.A.Q.
  - 1.1 General Questions
    - 1.1.1 How do I customize the bots names ?
  - 1.2 ET Related
    - 1.2.1 How To Adjust The Difficulty
    - 1.2.2 How To Adjust The MG42 Aiming
    - 1.2.3 How To Edit Waypoints
    - 1.2.4 How To use Custom Waypoints
    - 1.2.5 How To Run Omni-bot On Windows(tm) Home Server
    - 1.2.6 How To Run Omni-bot On Linux Home Server
    - 1.2.7 How To Quickly Add The Omni-bots To Your server
    - 1.2.8 Omni-bot and Other Mods
    - 1.2.9 Mod Folders & Mod Versions
    - 1.2.10 Mod Specific Cvars
  - 1.3 Doom 3 Related
  - 1.4 Quake 4 Related
  - 1.5 General Server Setup Tips

## Omni-Bot F.A.Q.

### General Questions

#### How do I customize the bots names ?

Open the *autoexec.gm* script for the game you wish to customize names for. For ET, the file is called **et\_autoexec.gm**, for Quake4 it is called **q4\_autoexec.gm**, and so on.

Inside you will see a table being constructed that associate names with a profile script.

Here's what the **q4\_autoexec.gm** looks like at the time of this writing.

```
Names["[BOT]Walter"] = "def_bot.gm";
Names["[BOT]Fred"] = "def_bot.gm";
Names["[BOT]Morgan"] = "def_bot.gm";
Names["[BOT]Lawrence"] = "def_bot.gm";
Names["[BOT]Richard"] = "def_bot.gm";
Names["[BOT]Michael"] = "def_bot.gm";
Names["[BOT]Brad"] = "def_bot.gm";
Names["[BOT]George"] = "def_bot.gm";
Names["[BOT]Anton"] = "def_bot.gm";
Names["[BOT]Monty"] = "def_bot.gm";
Names["[BOT]Bean"] = "def_bot.gm";
Names["[BOT]Sean"] = "def_bot.gm";
Names["[BOT]Backfire"] = "def_bot.gm";
```



```
Names["[BOT]Halfwit"] = "def_bot.gm";
Names["[BOT]Halfbaked"] = "def_bot.gm";
Names["[BOT]Fullmonty"] = "def_bot.gm";
Names["[BOT]Nohope"] = "def_bot.gm";
Names["[BOT]Hitnrun"] = "def_bot.gm";
Names["[BOT]Missnrun"] = "def_bot.gm";
Names["[BOT]Oysterhead"] = "def_bot.gm";
Names["[BOT]Fullthrottle"] = "def_bot.gm";
Names["[BOT]Noammo"] = "def_bot.gm";
Names["[BOT]Bullseye"] = "def_bot.gm";
Names["[BOT]Aimless"] = "def_bot.gm";
Names["[BOT]Blackadder"] = "def_bot.gm";
```

When a bot is added to the game without a name specified, it will choose a random available name from the ones listed. Each name has a profile associated with it, in this case all of these names have def\_bot.gm, which is typically common for most games. The profile script is a way to customize a particular bot. You can script additional features or modifications to existing features into a script and associate the script with a specific bot name in order to have varying yet consistent behavior. Look here for more on customizing your Omni-bots.

**To add additional names, simply add another line with a new name.**

Normally there should be at least as many names as there are player slots available in the game, though you can add more and Omni-bot will randomly choose between all of them.

Small note : you don't have to add the prefix [BOT] to the name of the bot, but it would be fair or at least informative to let the players on your server know in a glance that Omni-bot is enabled and running.

## ET Related

### How To Adjust The Difficulty

Here's a page with a : Difficulty sample.gm

### How To Adjust The MG42 Aiming

Here's a page that describes just that : MG42 Aim Adjustments

Here you'll find more : Community Scripts

### How To Edit Waypoints

Here you have the Omni-bot Waypointing general info page. There's also a page with a nice : waypointing tutorial for Enemy Territory.

On this Omni-bot wiki page you will find a table which lists the currently available and being worked on waypoints for Enemy Territory, also check the waypoint download section on the regular website of Omni-bot for more files.

### How To use Custom Waypoints

When you've downloaded a waypoint file for a custom ET map, you simply put the \*.way file inside your : ... \et

\nav directory.

Although the files that are delivered with Omni-bot are zipped, it is not necessary to put the new waypoint files inside the existing archives.

## How To Run Omni-bot On Windows(tm) Home Server

Here's a small description on how to make a quick -vanilla ET + Omni-bot server run on your Win32 home computer. You should be able to let some Omni-bots run and also view and edit or create waypoints for the maps with it.

If not already :

- Install Wolfenstein: Enemy Territory
- Run the patch 2.60b
- Install Omni-bot (latest version as of aug-2-2007 : 0.66 Stable)

When all the above is installed you'll see a new shortcut in your Start Menu : Omni-Bot ET. When you click on this shortcut you will start Enemy Territory with the mod : Omni-bot. In this way you will still have to go through the ingame menu to host and or play on your own server.

To avoid this hassle every time here below there's the double click way to start your own server. Make a shortcut like this on your desktop or wherever you want :

```
"C:\Program Files\Wolfenstein - Enemy Territory\ET.exe" +set dedicated 0 +set fs_game omnibot +set com_hunkmegs 64 +exec server.cfg
```

- *+set dedicated 0 \*a server, to be played on as well by the server host (=you)*
- *+set fs\_game omnibot \*set the gametype, make sure to set this if you want to start waypointing*
- *+set com\_hunkmegs 64 \*allocates a certain amount of memory for map and item loading*
- *+exec server.cfg \*server settings, your server configuration file which gets executed, in directory 'etmain'*

After you made the shortcut, you should be able to start and run the Omni-bot server with it, and start for example waypointing.

More and in depth info : [Advanced Wolfenstein: Enemy Territory Server Setup Guide](#)

There's also a pdf file for download.

## How To Run Omni-bot On Linux Home Server

```
#et +set fs_game omnibot +set com_hunkmegs 64
```

Make sure you've read the information on install Omni-bot on Linux

Some more links on Server Info

## How To Quickly Add The Omni-bots To Your server

There are -in short- 3 ways :

- Via the : MinBots & MaxBots Omni-bot commands *\*set those in your et\_autoexec.gm. file, in the directory 'scripts'*
- Via the 'bot addbot' console command, f.e to add a Axis Covert type : "\bot addbot 1 5 Covert-Axis"
- Via a omni-bot.cfg, a file in which you put the above 'bot addbot' command as many times you want a bot with a selected class.

Example 1 omni-bot.cfg :

```
// start
// bot addbot <team> <class> <name>
// save f.e as : omni-bot.cfg -in same folder where your server.cfg is-
// put the line : exec omni-bot.cfg in your server.cfg to add bots automatic on map start
bot addbot 1 1 Soldat-Axis
bot addbot 1 2 Sani-Axis
bot addbot 1 3 Engi-Axis
bot addbot 1 4 Leut-Axis
bot addbot 1 5 Covert-Axis
bot addbot 2 1 Soldier-Allied
bot addbot 2 2 Medic-Allied
bot addbot 2 3 Engi-Allied
bot addbot 2 4 Leut-Allied
bot addbot 2 5 Covert-Allied
// eof
```

Example 2 omni-bot2.cfg :

```
// start
bot minbots 0
bot maxbots 10
// eof
```

## Omni-bot and Other Mods

Running Omni-bot with Wolfenstein:Enemy Territory is also possible in combination with several other mods

Setting up a mod server to run with Omni-bot isn't much different from running Omni-bot with vanilla etmain.

- Reassure yourself that the Omni-bot & mod version you are trying to run are fully compatible. (See below Table)
- Use the included readme file in the mod folder or check their website and or forum, f.e. to get more info on Omni-bot specific cvars for your mod.
- **Important Note** : you cannot edit or create waypoints in the gametypes etpub, jaymod, or NQ. See also : Omni-bot Waypointing

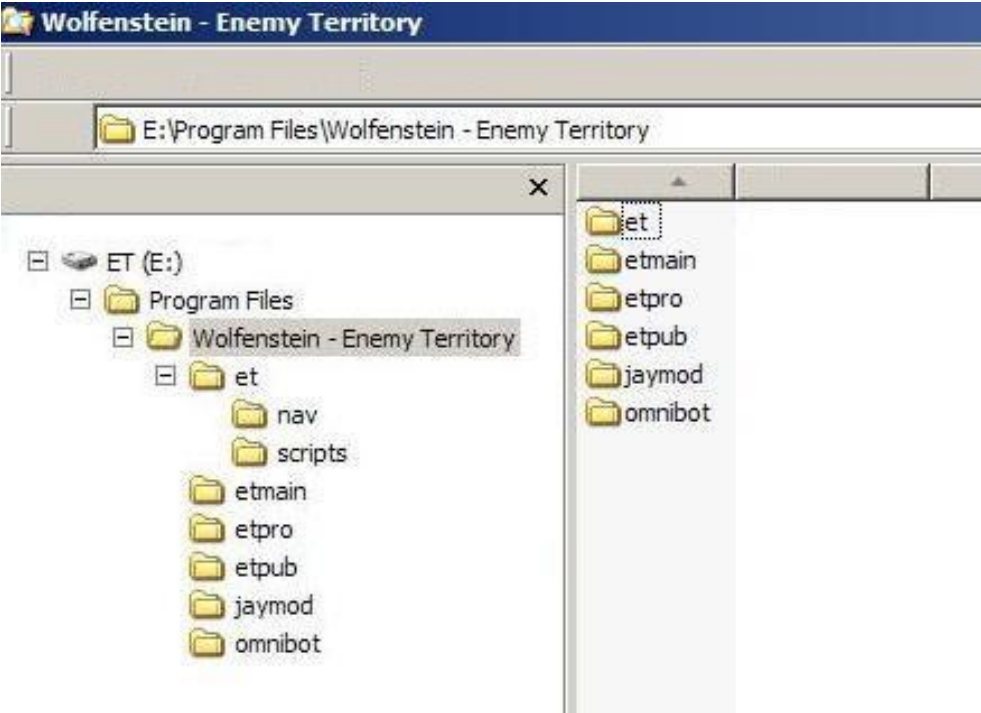
## Mod Folders & Mod Versions

The modfolders called omnibot, jaymod, etpub & noquarter should all (if installed) be at the same level as the etmain folder.

Like this :

So after you install the mod you can create a new shortcut that points to the specific mod (folder) like :

- ETPub  
  
"C:\...\ET.exe" +set fs\_game etpub +set com\_hunkmegs 64  
+exec pubserver.cfg
- Jaymod



Omni-bot &Mod Versions Compatible Table			
Omni-bot	Jaymod	ET pub	NQ
0.5.2	2.0	0.7.0	tbd
0.5.3.2	tbd	0.7.1	tbd
0.5.3.2	tbd	0.7.2	tbd
0.60	2.1.2	0.7.3	1.1.1
0.61	2.1.2	tbd	1.1.1
0.61	2.1.4	tbd	1.1.1
0.65	2.1.5	tbd	tbd
0.66	2.1.5	0.8.1	1.2.0
0.66	2.1.6	0.8.1	1.2.0
0.66	2.1.7	0.8.1	1.2.0

"C:\...\ET.exe" +set fs\_game jaymod  
+exec jayserver.cfg

- NQ  
  
"C:\...\ET.exe" +set fs\_game  
noquarter +set com\_hunkmegs 64  
+exec nqserver.cfg

You'll notice that the only thing in the shortcut line changing is the setting of fs\_game.

Also you can set some Cvars which are specific to your mod into your (mod) server.cfg.

Mod Specific Cvars

Mod	Omni-bot Specific Cvars URL	Mod Version Doc URL
ET Pub	Omni-bot ET Pub 0.8.1	0.8.1
Jaymod	Omni-bot Jaymod	2.1.6
NQ	Omni-bot NQ	1.1.1

## **Doom 3 Related**

TBD

Until then check out these links for Doom 3

## **Quake 4 Related**

TBD

Until then check out these links for Quake 4

## **General Server Setup Tips**

- Use a 'shortcut' to startup your server.
- Start your server with a 'basic settings' server.cfg. (f.e. for ET use the server.cfg in your etmain folder)
- Make back-ups of (all) your server configuration files!
- Add the cvars you want to run the server with gradually, so you can always revert to a previous version when something is wrong.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_F.A.Q.](http://omni-bot.com/wiki/index.php?title=Omni-bot_F.A.Q.)"

This page has been accessed 8,483 times. This page was last modified 13:30, 5 July 2008.

# General Bot F.A.Q.

From Omni-bot Wiki

## What is a bot ?

Anyone that has played FPS(First Person Shooter) games is probably familiar with bots.

Bots are computer controlled opponents, typically designed to play a game in a similar manner as what you might encounter when playing online multiplayer with other players. Although it is very difficult to get bots to actually play like a human would, most bot authors invest considerable time giving the bots the ability to competently play the game, and be a fun opponent.

Bots normally reside on the game server, meaning it is the game server that actually runs the bots, processes their navigation and AI, and ultimately controls the overall actions of the bots. Client side bots are commonly considered cheats, because they too are computer controlled and are often capable of giving individual clients unfair advantages. Aimbots could be considered a form of client side bot.

## What are they bots for ?

- Sometimes, for some people, playing against a bot can be more fun than playing a human. Humans are commonly idiots in multi-player games. They may cheat, complain, camp vehicles/spawns, teamkill, or generally do other things that ruin the enjoyment of other players. A well coded bot will play the game as designed. They don't complain. They don't cheat.
- They can be good practice. Playing against some bots, or even a hoard of bots, can be a good way to practice your death-matching skills, dodging, avoidance, sniping, etc.
- Bots are a good way for a server administrator to attract real players to their server. In a server browser, there are often many hundreds of servers listed. Few people will choose to join an empty server, so bots are often a good way to keep some gameplay going on a server even when no humans are around. Most bots have options that allow bots to be removed from the game as human players join, eventually resulting in a game of all or mostly humans.
- Playtesting maps. As a mapper building a map, it is often a good idea to playtest it. Sometimes it can be difficult to get enough humans together to properly test a map. If the mapper so desires, they can build bot navigation for their map as they develop it and test it with a full game of bots, or humans and bots, to test the balance, fun, and overall flow of the map. Foxbot for TFC was often used in this way.

# Do bots cheat ?

The answer to this varies from bot to bot. To be perfectly honest, technically some of the way bots have to be coded might be considered cheating by some, however most bot authors go through great lengths to reduce cheating to a level where the fun is not ruined.

Some examples of things that might be considered cheating

- Bots 'see' by doing collision tests with lines or volumes. This means that in order to determine if a bot can 'see' another player, a ray is shot from the bots eye to the target. If this ray doesn't collide with anything, such as walls, other players, props, etc, we would consider the object in the bots view. If the ray does hit something we consider the view blocked.

This concept is both a strength and a weakness of bot AI.

The strength, and the potential for cheat accusations to come about, are often due to the reliance on collision to determine view. For example, many games have foliage in their outdoor maps. Grass, bushes, trees, etc. Often these sort of things do not have any kind of collision on them, so they don't obstruct the view of a bot at all. To counter this, some games go through extra efforts to simulate obstruction through foliage. Many bots don't make such compensations, so in maps that contain objects that to humans might be considered cover because they block your view with transparent, often the bot is able to see right through them. This also comes into play with game elements such as flashbangs, smoke grenades, tear gas, or other things that might be based around altering the rendering of a human players screen. These sort of effects are often faked with bots, in one way or another.

A weakness of this method, is that it gives the bot a very very rough view of the environment. At the simplest level, a bot might cast a single ray from his eye to the enemy eye or enemy position to see if it can 'see' them. If the ray hits something, he can't see it, if it doesn't hit anything, he can see it. The problem with relying on a single raycast to make this determination, is that in reality that one spot on the target might be obstructed, while the rest of the target might be viewable. Due to this, sometimes bot authors do several raycasts at different positions on the target, to account for the possibility of partially obstructed targets. This results in a better view system, at the expense of requiring more processing. In extreme cases, a bot might cast a ray to every body part or bone on the target. This would give excellent resolution to the bot, but at a very high increase in cpu cost. Ultimately it depends on how accurate the bot author wants them to be. Most games typically don't notice slight inaccuracies in the resolution of the bots view system, while a slower paced tactical game might benefit more from it.

Also under the category of cheating, it is true that as a server side mod, the bot author

technically has full access to where everything and everyone is in the game. It is fully within the capabilities of the bot coder to make the bots fully aware of everyone and everything. As a matter of fact, that would be the easiest way to make a bot. Fortunately, most bot authors don't like their bot to cheat that badly, which is why a bots knowledge of the world is heavily filtered, usually by parameters that mimic the view capabilities of a human player as close as possible, like the field of view, view distance, and whether the view is obstructed, usually by doing ray cast collision checks.

Finally, the subject most often accused of cheating is a bots aiming. It is very easy to make a bot shoot exactly on target every time. Some bots have implemented aiming systems that have resulted in an almost superhuman accuracy. These bots are rarely fun to play, and most bot coders also put alot of effort into a more human-like aiming system, with properties such as a max turn rate, and adding acceleration/deceleration to aiming, both for smoothness and for less robotic behavior.

In the end, bot AI, and game AI for that matter almost always cheats to some degree. It isn't feasible, both in time and processing power, to make an AI work exactly like a human. The key to good AI is limiting the cheating of the bot to an acceptable level. Some cheating is inevitable, but a good bot will not make it obvious that they are cheating. It's all about having fun, and sometimes the code has to be written to work around the fact that bots don't see a rendered world at all like players, and sometimes have very limited or non existent hearing capabilities too. Ultimately it doesn't matter, as long as the gaming experience is fun.

You will find some more links on AI in the Programming & Scripting section of the Community Portal page.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=General\\_Bot\\_F.A.Q.](http://omni-bot.com/wiki/index.php?title=General_Bot_F.A.Q.)"

This page has been accessed 691 times. This page was last modified 16:27, 15 September 2007.



# Omni-bot Wiki:Community Portal

From Omni-bot Wiki

Here you will find some links for easy reference to the supported games of Omni-bot and links to related topics.

## Contents

- 1 Enemy Territory
  - 1.1 Files
  - 1.2 Fora or Forums
  - 1.3 Funny
  - 1.4 General Info
  - 1.5 Leagues & Competitions
  - 1.6 Mods
  - 1.7 Programming & Scripting
  - 1.8 Server Info
- 2 Fortress Forever
  - 2.1 General Info
- 3 Quake 4
  - 3.1 Files
  - 3.2 General Info
  - 3.3 Mods
  - 3.4 Server Info
- 4 Doom 3
  - 4.1 Files
  - 4.2 General Info
  - 4.3 Mods
  - 4.4 Server Info
- 5 Pages Omni-bot & Wiki Users

## Enemy Territory

### Files

- Escapedturkey A Big List Of ET Files

- Strategy Informer ET Files
- Wildstar ET maps
- Wolfenstein: Enemy Territory -Filefront
- Wolfmap ET Files
- etpro mapscripts

## Fora or Forums

- ET Pub
- Jaymod
- NQ
- Omni-bot
- Splash Damage
- Teamwarfare ET

## Funny

- ET Comic
- Online Party Panzer Pattern Maker

## General Info

- Enemy Territory (Final) - A Quick Guide 4 Newbies
- ET-Center
- ET Fun Community Kallisti
- ET Linux (Ubuntu Install)
- ET-Nation.co.uk
- ET-Resource Site
- ET-Scene.de (German)
- ET Unofficial Strategy Guide
- ET + RTCW Jolt Co.
- GamesTV.org
- Planet Wolfenstein
- Quake & RTCW MDC & MDS File Format
- SplashDamage
- Team Fortress Xtreme
- Wiki Wolfenstein: Enemy Territory
- Wolf ET MDM & MDX File Format
- Wolf Wiki

## Leagues & Competitions

- Clanbase
- ET-Cup
- GameArena - Australian Community
- Stronger Than All
- TeamWarfare League
- Warleagues

## Mods

- ET Admin Wiki
- ET Pub
- Jaymod
- Jaymod Wiki
- NQ Wiki
- Omni-Bot

## Programming & Scripting

- AI Depot
- ET-Scripting Reference
- Finite State Machines (FSM)
- Game AI Development
- Gamemonkey Script
- GameMonkey Script Downloads
- Introduction to GameMonkey Script
- Code3Arena Tutorials
- C# Programming
- Learning C++
- Logic Software - CSLI
- Mathematical Software Resources
- OutCast\*\*NL's Scripts & Binds
- Planet Source Code
- Qfsm
- Simple finite state machine in C
- SMC - Finite State Machine Compiler (Java)
- The basics of C++
- The Codeproject, pick your syntax
- Turing Machines
- Turing Machines (TM)
- UML Tutorial: Finite State Machines

- Visual Turing -Freeware
- Wiki Finite State Machine

## Server Info

- Advanced Server Setup
- Console Commands
- General Server Setup
- Rcon Commands
- World Server Ranks
- Wiki General Game Servers Setup

## Fortress Forever

### General Info

- Fortress Forever Forum
- Fortress Forever Home
- Dust Bowl Valley demo (vid)
- Wiki Page

## Quake 4

### Files

- Quake 4 Filefront

### General Info

- Planet Quake 4
- Wiki Quake 4

### Mods

- Quake 4 Mods List

### Server Info

- [Quake 4 Linux Server Setup GameAdmins](#)
- [Quake 4 Linux Server Setup Wiki](#)

## Doom 3

### Files

- [Doom 3 Filefront](#)
- [Doom 3 \(mini\) Mods](#)

### General Info

- [Doom 3](#)
- [Doom 3 Linux FAQ](#)
- [Doom 3 Wiki](#)

### Mods

- *co-operative gameplay* *You play with your teammates through the singleplayer levels and fight against the demons.*

### Server Info

- [Server Guides for Doom 3](#)

## Pages Omni-bot & Wiki Users

- [762](#)
- [Alice](#)
- [Crapshoot](#)
- [D00d](#)
- [DrEvil](#)
- [Hank officer](#)
- [IRATA](#)
- [Magik](#)
- [Stoned-Aimlezz](#)
- [Vypir](#)

# SciTE

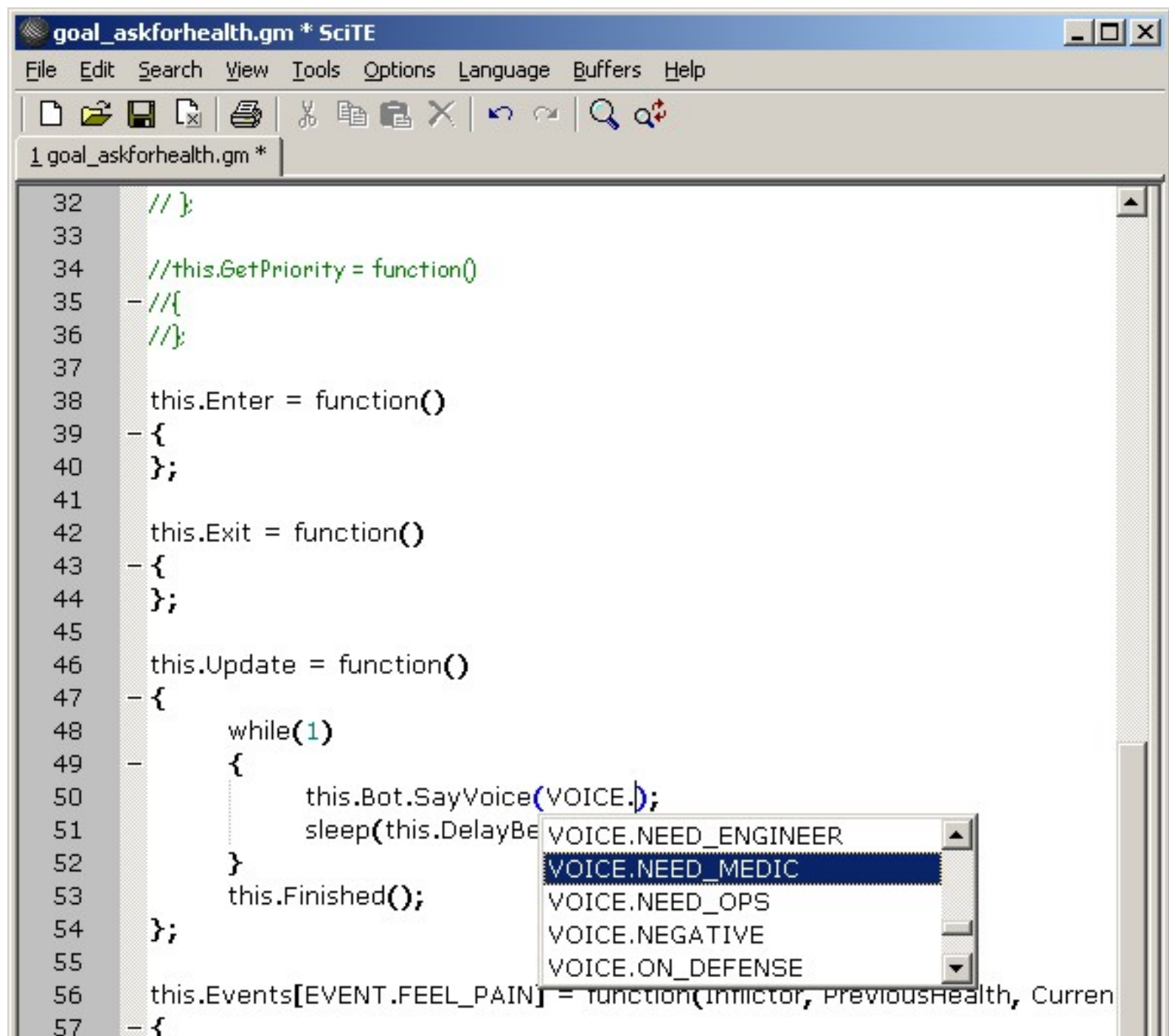
From Omni-bot Wiki

## SciTE

SciTE is a useful and extendable text editor of choice for the Omni-bot developers.

The version of SciTE available here comes pre-configured with syntax highlighting for Game Monkey script, as well as keywords and auto completion rules for Omni-bot.

- [Download SciTE 1.75](#)



The screenshot shows the SciTE text editor window titled "goal\_askforhealth.gm \* SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The toolbar contains icons for file operations and editing. The editor displays a Game Monkey script with syntax highlighting. The script includes comments and function definitions for a bot's behavior. An auto-completion dropdown menu is visible, showing a list of voice types: VOICE.NEED\_ENGINEER, VOICE.NEED\_MEDIC (highlighted), VOICE.NEED\_OPS, VOICE.NEGATIVE, and VOICE.ON\_DEFENSE.

```
32 // }
33
34 //this.GetPriority = function()
35 - //{
36 //}
37
38 this.Enter = function()
39 - {
40 };
41
42 this.Exit = function()
43 - {
44 };
45
46 this.Update = function()
47 - {
48     while(1)
49     - {
50         this.Bot.SayVoice(VOICE.);
51         sleep(this.DelayBe
52     }
53     this.Finished();
54 };
55
56 this.Events[EVENT.FEEL_PAIN] = function(Injictor, PreviousHealth, Curren
57 - {
```

```
54 };
55 VOICE.NEGATIVE
56 VOICE.ON_DEFENSE
57 this.Events[EVENT.FEEL_PAIN] = function(Infligator, PreviousHealth, CurrentHealth)
58 - {
59     whoDoneIt = GetEntName(Infligator);
60     //print( format("%s: %s Hurt me: Was %d, now %d", this.Bot.Name, whoDoneIt, PreviousHealth, CurrentHealth));
61     // commented out for now since goal doesnt get passed to callback. wip
62     if(CurrentHealth < this.HealthThreshold)
63     - {
64         this.Priority = 1;
65     }
66 };
```

li=50 co=41 INS (CR+LF)

**Feb 9 2008**

Initial upload of SciTE for Omni-bot. Contains Most autocomplete and highlighting for Enemy Territory and core Omni-bot.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=SciTE>"

This page has been accessed 217 times. This page was last modified 18:03, 24 February 2008.

# PSPad

From Omni-bot Wiki

**PSPad** is a useful and extensible text editor of choice for the Omni-bot developers. It can be easily set up to work with many different programming and markup languages. Get the syntax highlighter archive below and copy the subfolders of the archive to your PSPad folder for a quick start.

## Contents

- 1 Basic Setup
  - 1.1 Which files go where?
  - 1.2 Setting up the Code Explorer
  - 1.3 Using an interpreter or compiler to find syntax errors
- 2 Using PSPad with Omni-bot scripts
  - 2.1 Code Completion: Turbo-charge your work
- 3 Download syntax highlighter archive
- 4 Page History
  - 4.1 Feb 14 2008
  - 4.2 Feb 15 2008
  - 4.3 Feb 27 2008

## Basic Setup

### Which files go where?

Get the syntax highlighter zip archive below and copy the folders contained in the archive to your PSPad folder, or extract the archive and copy the files you want one by one.

The archive contains:

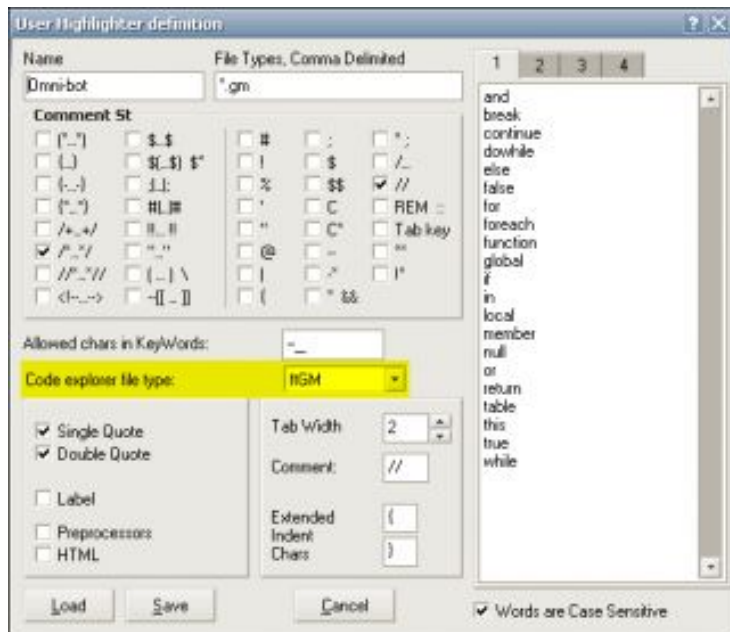
- An .ini file that list all the GameMonkey keywords, as well as most Omni-bot constants and functions; this belongs in your PSPad\Syntax folder.
- A .def file that provides code snippets and some syntax hints; this belongs in your PSPad\Context folder.
- A template for map scripts in standard format, a template for weapon scripts, and one for bot scripts; these should go to your PSPad\Template folder. Most users will hardly ever use them, but some might find them useful. Note that you can have any number of templates for a given file type.
- A script that can help to properly indent your code. It is intended for use with JavaScript code, but due to the similarity of the syntax, it can be used with GM files too. Useful if you have a piece of code



whose indentation got messed up somehow, e.g. through copy and paste.

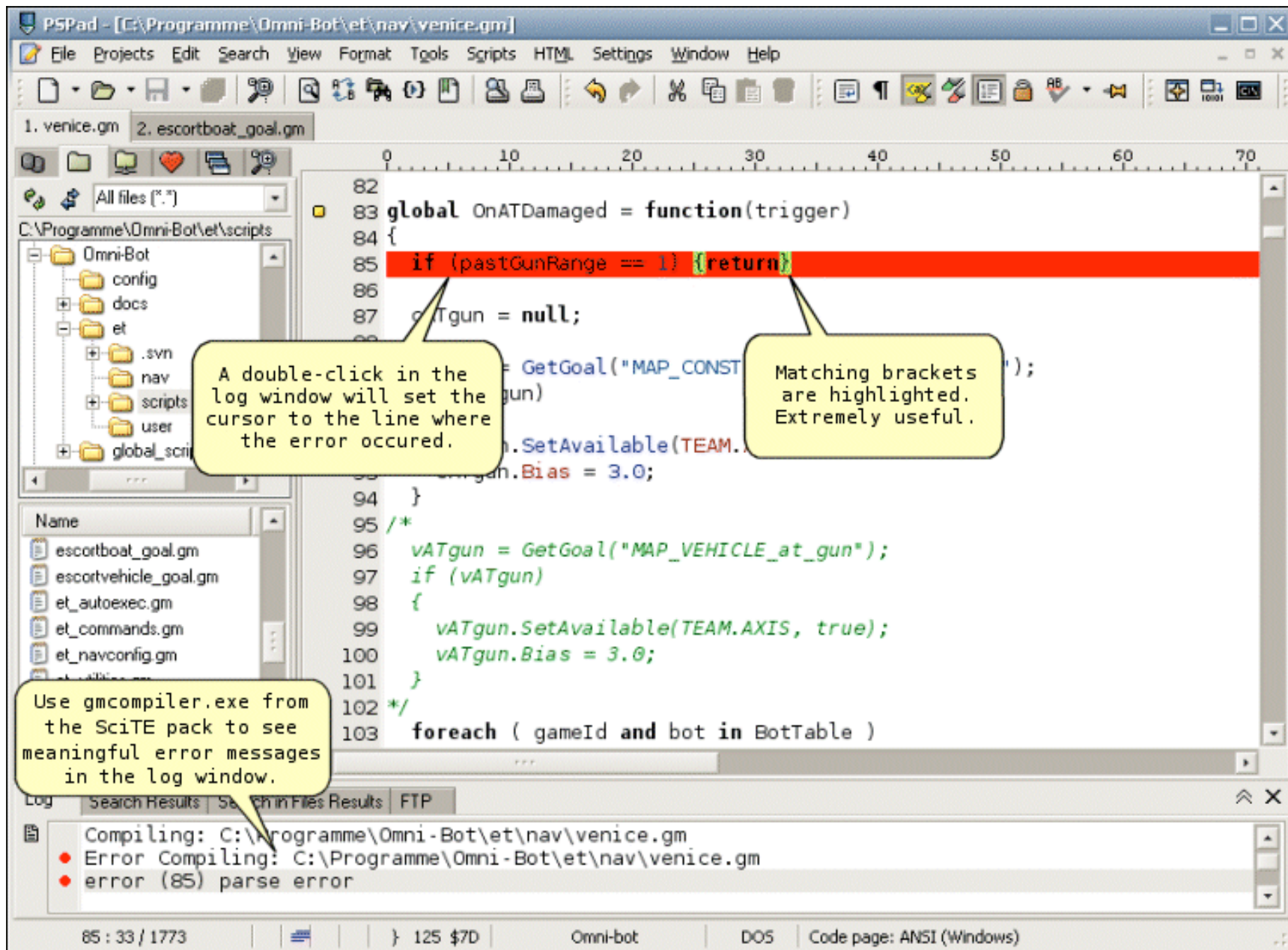
## Setting up the Code Explorer

Unlike SciTE, PSPad has no code folding (i.e. no plus or minus signs on the left that can be used to collapse or expand blocks of code). This is partly compensated for by the code explorer, a dockable window that shows the functions defined in your code. If the code explorer doesn't work in your setup, choose ftGM as code explorer file type:

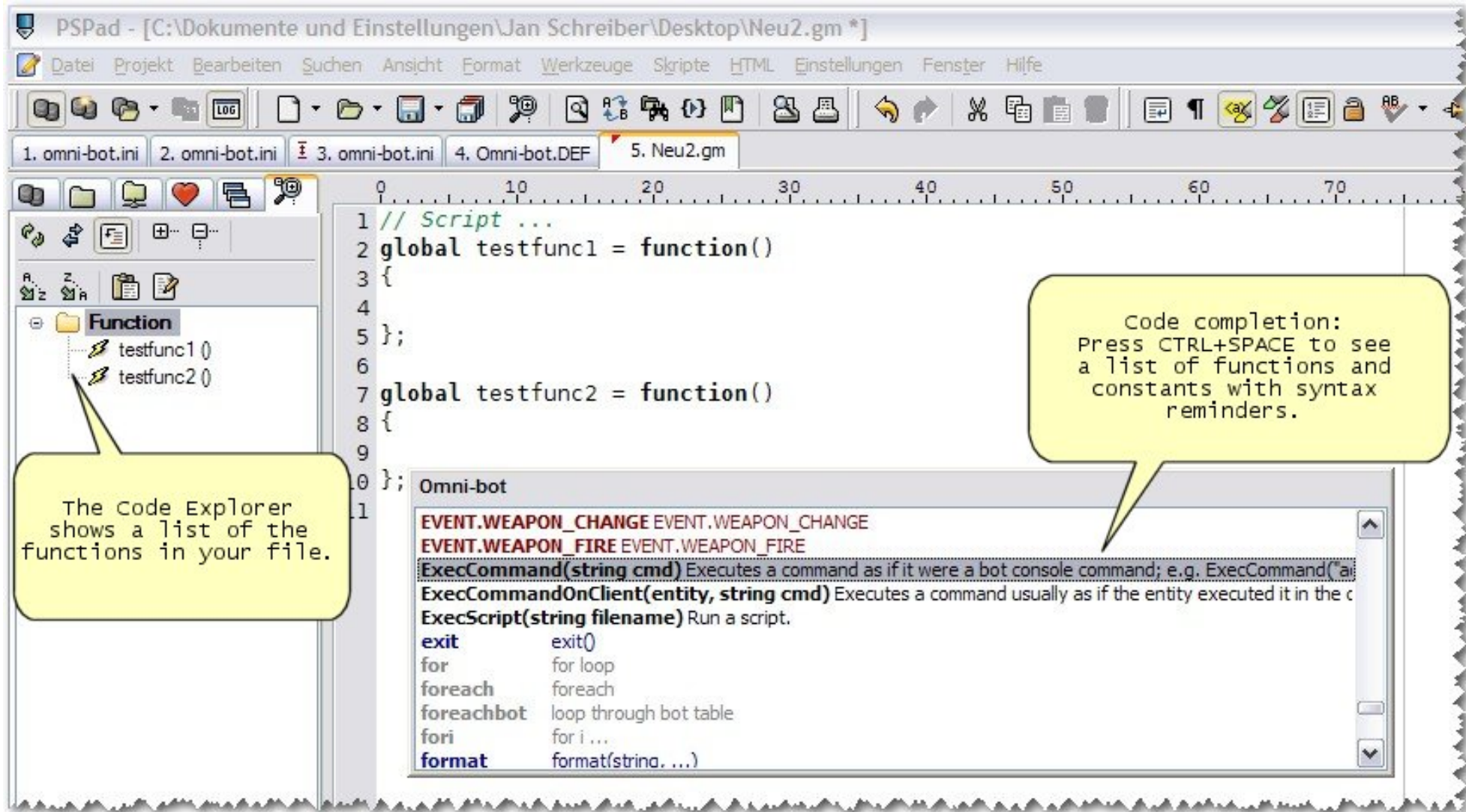


## Using an interpreter or compiler to find syntax errors

Ideally, you should get the SciTE Editor pack and use gmcompiler.exe as compiler for .gm files, or get the GameMonkey download and use the gme.exe contained therein. This will help to find syntax errors.



## Using PSPad with Omni-bot scripts



## Code Completion: Turbo-charge your work

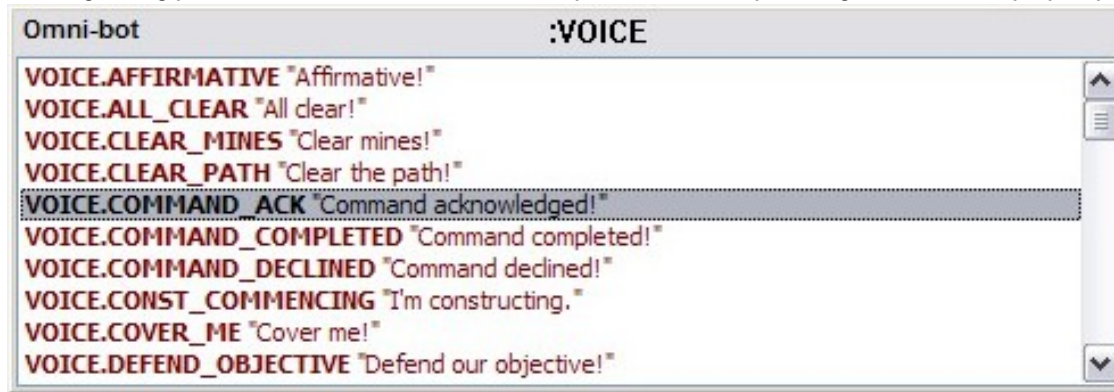
Most IDEs have a feature that is most commonly known by the Microsoft trademark IntelliSense. As a generic editor, PSPad can't have all the benefits of a dedicated IDE with IntelliSense, but it has two similar features: *Clips* or *snippets* and *Auto Completion*, by default invoked via Ctrl+Space and Ctrl+J, respectively. The pop-ups are never triggered automatically in PSPad, you have to use the keyboard shortcuts mentioned above.

- The clip window (Ctrl+Space) shows everything that is configured in the omni-bot.def context file. You can type while it is open in order to narrow down the list.
- The auto completion window (Ctrl+J) scans

- your currently edited file,
- the syntax definition omni-bot.ini

for entries that match the characters left of the cursor. This is useful for variable names and keywords that haven't got a clip definition entry.

For example, when you type "voi" into the editor and press Ctrl+Space, you'll see a pop-up similar to



the following:

## Download syntax highlighter archive

This archive contains syntax highlighting and auto completion data for PSPad. For more info see the ReadMe.txt within the archive.

- PSPad-highlighter.zip

---

## Page History

### Feb 14 2008

Initial upload of PSPad highlighting for Omni-bot. Contains most autocomplete and highlighting for Enemy Territory and core Omni-bot.

### Feb 15 2008

Minor improvements and corrections. -- d00d 06:49, 15 February 2008 (MST)

### Feb 27 2008



# Notepad++

## From Omni-bot Wiki

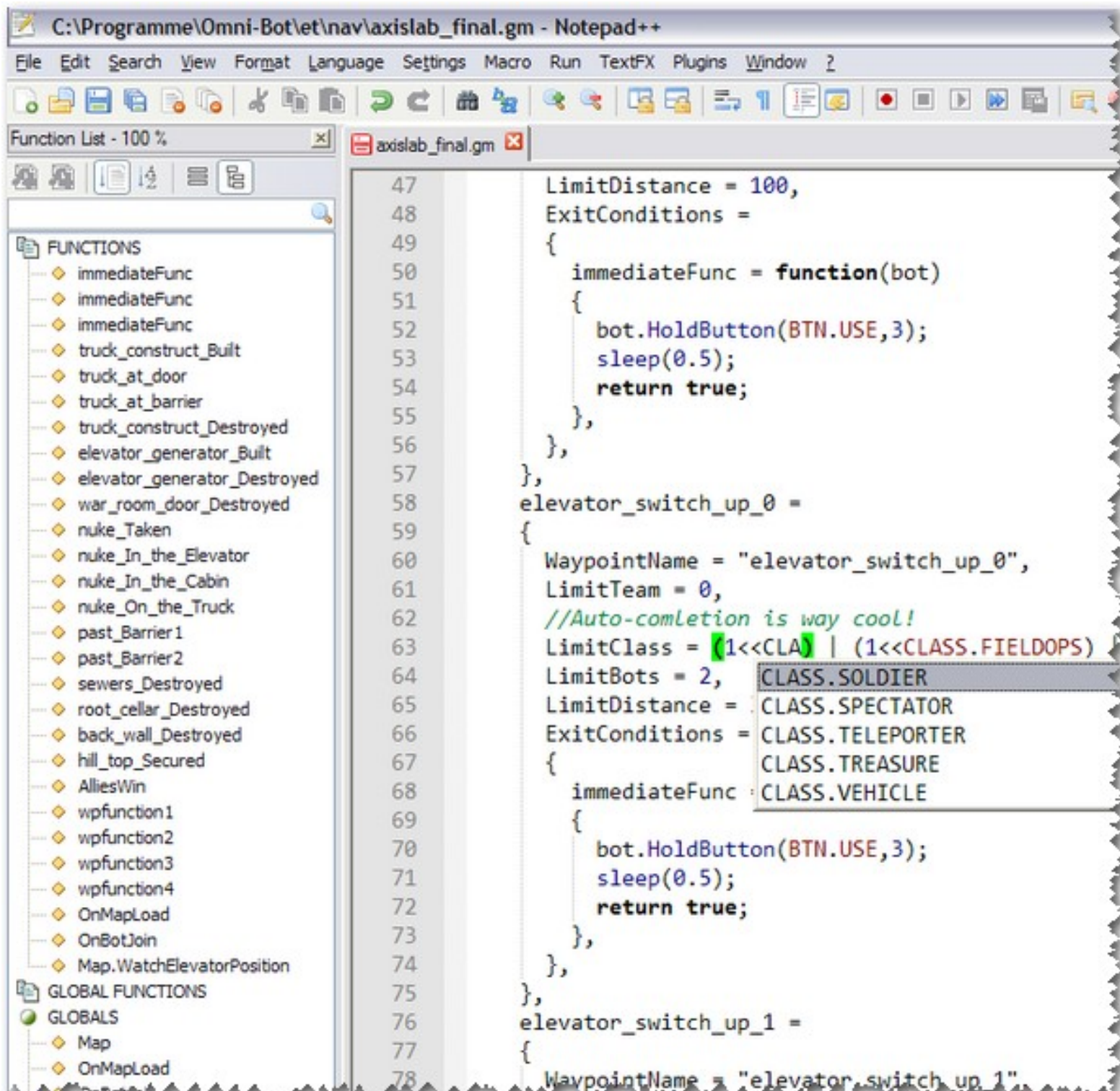
**Notepad++** is a useful and highly configurable text editor. It is primarily developed by Don Ho and distributed under the GPL. In July 2008, it was voted "Best Tool or Utility for Developers" by the SourceForge community.

The version of Notepad++ available here comes pre-configured with syntax highlighting for Game Monkey script, as well as keywords and auto completion rules for Omni-bot. Select Omni-bot from the languages menu.

The archive contains a gmcompiler binary for Windows. Select gmcompiler from the Run menu or press Ctrl+F9. If there are syntax errors in your file, you will see an error message.

You can download a localization for your native language and additional plug-ins from the Notepad++ download page.

- [Download Notepad++ 5.0.2](#)



## Contents

- 1 August 1 2008
- 2 July 19 2008
- 3 July 6 2008
- 4 March 12 2008
- 5 March 2 2008

### **August 1 2008**

Upgraded to N++ 5.0.2.

### **July 19 2008**

Upgraded to N++ 5.0.1.

### **July 6 2008**

Upgraded to N++ 5.0.

### **March 12 2008**

Updated function list expressions and Notepad++ executable.

### **March 2 2008**

Initial upload of Notepad++ for Omni-bot. Contains Most autocomplete and highlighting for Enemy Territory and core Omni-bot.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Notepad%2B%2B>"

This page has been accessed 260 times. This page was last modified 11:19, 1 August 2008.

# ET Minimizer

From Omni-bot Wiki

The ET Minimizer is very useful if you want to run ET in fullscreen mode and still want to be able to edit your script(s) in between.

Note that the tool doesn't restore your system's gamma correction to a normal value, so your display will appear much too bright. If you have a waypointing config file with key bindings, you might want to add the following to it:

```
bind KP_DEL "vstr togglegamma"
set gamma1 "seta r_gamma 1.00; set togglegamma vstr gamma2;"
set gamma2 "seta r_gamma 1.95; set togglegamma vstr gamma1;"
set togglegamma "vstr gamma1"
```

This makes the DEL key on the numeric keypad toggle between the ET default gamma value of 1.95 and a value of 1.00 that is suitable for virtually everything outside ET. Before minimizing and after returning to the game, just press KP\_DEL.

- [Download ET Minimizer](#)

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET\\_Minimizer](http://omni-bot.com/wiki/index.php?title=ET_Minimizer)"

This page has been accessed 260 times. This page was last modified 20:06, 26 February 2008.



# Bot Library

From Omni-bot Wiki

## Contents

- 1 Global Bot Functions
  - 1.1 AddBot
  - 1.2 CalcTrajectory
  - 1.3 CheatsEnabled
  - 1.4 Trigger Regions
    - 1.4.1 OnTriggerRegion
    - 1.4.2 DeleteTriggerRegion
  - 1.5 DistanceBetween
  - 1.6 Debug Functions
    - 1.6.1 DrawDebugLine
    - 1.6.2 DrawDebugAABB
    - 1.6.3 DrawDebugAABB
    - 1.6.4 EchoToScreen
    - 1.6.5 Error
    - 1.6.6 ShowPaths
    - 1.6.7 Log
  - 1.7 ExecCommand
  - 1.8 ExecScript
  - 1.9 RunScript
  - 1.10 Entity Functions
    - 1.10.1 GetEntBonePosition
    - 1.10.2 GetEntCategory
    - 1.10.3 GetEntClass
    - 1.10.4 GetEntEyePosition
    - 1.10.5 GetEntFacing
    - 1.10.6 GetEntFlags
    - 1.10.7 GetEntHealthAndArmor
    - 1.10.8 GetEntOwner
    - 1.10.9 GetEntPowerups
    - 1.10.10 GetEntTeam
    - 1.10.11 GetEntWorldAABB
    - 1.10.12 GetEntityName
    - 1.10.13 GetEntName
    - 1.10.14 GetEntityLocalSpace
    - 1.10.15 GetEntityWorldSpace
    - 1.10.16 GetEntPosition
    - 1.10.17 GetEntRotationMatrix
    - 1.10.18 GetEntVelocity
    - 1.10.19 GetEntityInSphere
    - 1.10.20 GetEntityStat
  - 1.11 EntityKill
  - 1.12 GetTeamStat
  - 1.13 GetGameEntityFromId
  - 1.14 GetGameIdFromEntity

- 1.15 GetGameName
- 1.16 GetModName
- 1.17 GetGameState
- 1.18 GetGameTimeLeft
- 1.19 GetTime
- 1.20 GetGoal
- 1.21 GetGoals
- 1.22 GetGravity
- 1.23 GetMapExtents
- 1.24 GetMapName
- 1.25 GetMaxPlayers
- 1.26 GetNumPlayers
- 1.27 GetNumBots
- 1.28 GetModVersion
- 1.29 GetPointContents
- 1.30 KickAll
- 1.31 KickBot
- 1.32 MinBots
- 1.33 MaxBots
- 1.34 OnTrigger
- 1.35 RegisterDefaultProfile
- 1.36 ServerCommand
- 1.37 SetAvailableMapGoals
- 1.38 SetMapGoalProperties
- 1.39 SetGoalGroup
- 1.40 TraceLine

## Global Bot Functions

### AddBot

Adds a bot to the game, and optionally specifies team, class, and name for the bot.

**Parameters:** ()

**Parameters:** (team)

**Parameters:** (team, class)

**Parameters:** (team, class, name)

**Parameters:** (table)

**Returns:** none

**Example:**

```
AddBot ( ) ;
```

```
// OR
AddBot(Team.Axis);
// OR
AddBot(Team.Axis, Class.Soldier);
// OR
AddBot(Team.Axis, Class.Soldier, "SomeDude");
// OR
// not all fields are valid for all games, here's a quake 4 example
// not all fields are required. any left out will use sensible default values
tbl =
{
    name="SomeDude",
    team=Team.StroGG,
    class=Class.Player,
    spawnpoint="someSpawnpointname",
    model="model_player_kane_stroGG",
    skin="base",
};
AddBot(tbl);
```

**See Also:** KickBot, KickAll

---

## CalcTrajectory

Calculates a projectile trajectory, and returns the results in a table, or null if no trajectory exists for the parameters given.

**Parameters:** (Vector3 start position, Vector3 target position, projectile speed, projectile gravity)

**Returns:** table or null

### Example:

```
mypos = Vector3(10,10,10); // get a valid position from somewhere
targpos = Vector3(20,20,20); // get a valid position from somewhere
projVel = 1000;
projGrav = 0.5;
traj = CalcTrajectory(mypos, targpos, projVel, projGrav);
if(traj)
{
    // if a table was returned, there can be 1 or 2 trajectories stored in it.
    // the 1st trajectory is the most direct trajectory
    // the 2nd trajectory is normally a mortar trajectory with a high degree of arc.
    // both trajectories are a unit length facing vector
    b.TurnToFacing(traj[0]);
    // OR
    b.TurnToFacing(traj[1]);
}
```

---

## CheatsEnabled

Checks if cheat mode is enabled in the game. Useful for debug or development scripts that use functions that are only available in cheat mode.

**Parameters:** none

**Returns:** true if cheats are enabled, false if not

**Example:**

```
if(CheatsEnabled())
{
    // do something
}
```

**See Also:** EntityKill, ServerCommand

---

## Trigger Regions

### OnTriggerRegion

Creates one of several types of trigger regions. A trigger region is an area that watches for entities of certain types to touch them and calls script callback functions for enter and exit. Useful if you want an easy way to be notified when certain entities go to certain areas of the map.

This function takes a table as the last parameter to define a number of flexible options for the trigger region. Here is a list of available key/value options for the table.

- Name - Optional name to give the region. Does not need to be unique.
- OnEnter - script function to call when an entity touches the region.
- OnExit - script function to call when an entity stops touching the region.
- TriggerOnClass - Define the class id you want the region to trigger on. This field may also be an indexed table with up to 8 class ids.
- TriggerOnCategory - Category of entities to watch for. This field may also be an indexed table with as many categories as you want to limit the trigger to.
- TriggerOnEntity - A specific entity to trigger on. This field may also be an indexed table with up to 8 entities you want to limit the trigger to.

The following region types are supported. The type is dependent on which parameters of the function are used.

- AABB - Axis aligned bounding box.
- Sphere - A radius around a position.

Triggers may be rendered for reference with */bot drawtriggers 1*

**Parameters:** (AABB, table)

**Parameters:** (position, radius, table)

**Returns:** unique serial number of region

**Example:**

```
// creates a sphere trigger 256 units in radius around 0,0,0
triggerInfo =
{
    Name="TestTrigger",
    TriggerOnClass=CLASS.ANYPLAYER,
    OnEnter = function(ent)
    {
        print(GetEntName(ent), "entered aabb trigger");
    },
    OnExit = function(ent)
    {
        print(GetEntName(ent), "exited aabb trigger");
    },
};
serial = OnTriggerRegion(Vector3(0,0,0),256,triggerInfo);
```

**See Also:** DeleteTriggerRegion

---

## DeleteTriggerRegion

Deletes a trigger region previously created with OnTriggerRegion. Because regions aren't required to have unique names, DeleteTriggerRegion will delete all regions that match the provided name. This allows grouping of different trigger regions by shared names.

**Parameters:** (name)

**Parameters:** (serial #)

**Returns:** none

**Example:**

```
// creates a sphere trigger 256 units in radius around 0,0,0, then delete it immediately.
triggerInfo =
{
    Name="TestTrigger",
    TriggerOnClass=CLASS.ANYPLAYER,
    OnEnter = function(ent)
    {
        print(GetEntName(ent), "entered aabb trigger");
    },
};
```

```

        OnExit = function(ent)
        {
            print(GetEntName(ent), "exited aabb trigger");
        },
    };
    serial = OnTriggerRegion(Vector3(0,0,0),256,triggerInfo);

DeleteTriggerRegion(serial);
// OR
DeleteTriggerRegion("TestTrigger");

```

**See Also:** OnTriggerRegion

---

## DistanceBetween

Utility function for checking distances between 2 objects. This function takes 2 parameters, but the parameter types can vary.

**Parameters:** (object1, object2)

For this function, each parameter can be one of the following types.

- Vector3
- GameEntity
- Gameld

This provides a flexible and fast function that allows a script to check the distance between a variety of source and destination object, and reduces the need for the script to convert between GameEntity or Gameld, or to get the position of the GameEntity itself.

**Returns:** distance between the objects

**Example:**

```

// assume entity1 is a GameEntity from another source, such as GetAllType
if(DistanceBetween(b.GetGameEntity(), entity1))
{
}
// OR
if(DistanceBetween(b.GetGameEntity(), Vector3(10,20,30))
{
}

```

---

## Debug Functions

## DrawDebugLine

Draws a line in the environment. Useful for debugging.

**Parameters:** (Vector3 start, Vector3 end, color, duration)

**Returns:** none

**Example:**

```
start = Vector3(0,0,0);
end = Vector3(20,20,20);
// draw it in red for 5 seconds
DrawDebugLine(start, end, COLOR.RED, 5);
```

**See Also:** DrawDebugAABB, GetEntWorldAABB

---

## DrawDebugAABB

Draws an [AABB](#).

**Parameters:** (AABB, color, duration)

**Returns:** none

**Example:**

```
AABB box; // initialize it with something
// draw it in red for 5 seconds
DrawDebugAABB(box, COLOR.RED, 5);
```

**See Also:** DrawDebugLine, GetEntWorldAABB

---

## DrawDebugAABB

Draws an [AABB](#).

**Parameters:** (AABB, color, duration)

**Returns:** none

**Example:**

```
AABB box; // initialize it with something
// draw it in red for 5 seconds
DrawDebugAABB(box, COLOR.RED, 5);
```

**See Also:** DrawDebugLine, GetEntWorldAABB

---

## EchoToScreen

Prints a message to the screen.

**Parameters:** (duration, message)

**Returns:** none

**Example:**

```
EchoToScreen(5, "Hello World!");
```

---

## Error

Prints an error to the games output console.

**Parameters:** (error message)

**Returns:** none

**Example:**

```
Error("Somethin bad happened");
```

---

## ShowPaths

Debug Information function. Prints Omni-bot version, revision, and revision date, along with file system paths.

**Parameters:** none

**Returns:** none

**Example:**

```
ShowPaths();
```

---

## Log



Writes a string to the omnibot log file.

**Parameters:** (string)

**Returns:** none

**Example:**

```
Log("Something cool happened");
```

---

## ExecCommand

This function executes a bot command as if it came from the games input console.

**Parameters:** (command string)

**Returns:** none

**Example:**

```
ExecCommand("addbot 1 2");
```

---

## ExecScript

Attempts to execute a script file.

**Parameters:** (filename)

**Returns:** true if successful, false if not

**Example:**

```
ExecScript("myscript.gm");
```

---

## RunScript

**See:** ExecScript

---

## Entity Functions

### GetEntBonePosition

Gets the world position of a specific bone on an entity.

**Note:** This function may not be implemented for all games.

**Parameters:** (GameEntity/GameId, BoneId)

**Returns:** Vector3 world bone position, or null if bone not found

#### Example:

```
headpos = GetEntBonePosition(someent, BONE.HEAD);  
if(headpos)  
{  
    // got head position  
}
```

---

### GetEntCategory

Checks if the entity belongs to one or more entity categories. This function takes one or more categories, and returns true if the entity belongs to all the provided categories.

**Parameters:** (GameEntity/GameId, Category, ...)

**Returns:** true if the entity belongs to all provided categories.

#### Example:

```
if(GetEntCategory(someent, CAT.PROJECTILE))  
{  
}  
if(GetEntCategory(someent, CAT.PLAYER, CAT.VEHICLE))  
{  
}
```

---

### GetEntClass

Gets the class of the entity.

**Parameters:** (GameEntity/GameId)

**Returns:** class of entity, or null if there was an error

## Example:

```
cls = GetEntClass(someent);  
if(cls == CLASS.SOLDIER)  
{  
}
```

---

## GetEntEyePosition

Gets the world eye position of an entity.

**Parameters:** (GameEntity/GameId)

**Returns:** Vector3 eye position, or null if there was an error

## Example:

```
eyepos = GetEntEyePosition(someent);  
if(eyepos)  
{  
    // got eye position  
}
```

---

## GetEntFacing

Gets the world facing vector of an entity

**Parameters:** (GameEntity/GameId)

**Returns:** Vector3 facing, or null if there was an error

## Example:

```
face = GetEntFacing(someent);  
if(face)  
{  
    // got face vector, as a direction vector  
}
```

---

## GetEntFlags

Checks if the entity has one or more entity flags

**Parameters:** (GameEntity/GameId, Entity Flag, ...)

**Returns:** true if the entity has all provided entity flags.

**Example:**

```
if(GetEntFlags(someent, ENTFLAG.CROUCHED))
{
}
if(GetEntFlags(someent, ENTFLAG.CROUCHED, ENTFLAG.RELOADING))
{
}
```

---

## GetEntHealthAndArmor

Gets an object that provides access to health and armor of an entity. The object returned by this function is unique, in that it updates itself and the user doesn't need to keep calling it repeatedly from a script that checks it often.

**Parameters:** (GameEntity/GameId)

**Returns:** Health/Armor object, or null if bad entity.

Properties available through the object.

- Health - Current Health
- MaxHealth - Max Health
- Armor - Current Armor
- MaxArmor - Max Armor
- IsValid - Whether the entity is still valid. Use when watching the health/armor for an arbitrary entity. IsValid should allow you to assess the lifetime of the entity.

**Example:**

```
healthArmor = GetEntHealthAndArmor(someentity);
if(healthArmor)
{
    while(1)
    {
        if(healthArmor.Health < 20)
        {
            b.Say("I'm Hurt!");
        }
        sleep(2);
    }
}
```

---

## GetEntOwner

Gets the owner of an entity. Typically used for entities that can be held or carried.

**Parameters:** (GameEntity/GameId)

**Returns:** GameId of owner, or null if none or error

**Example:**

```
owner = GetEntOwner(someent);
```

---

## GetEntPowerups

Checks if the entity has one or more powerups. This function takes one or more powerups, and returns true if the entity has all of them.

**Parameters:** (GameEntity/GameId, PowerUp, ...)

**Returns:** true if the entity has all powerups.

**Example:**

```
if(GetEntPowerups(someent, POWERUP.INVINCIBLE))
{
}
if(GetEntPowerups(someent, POWERUP.QUADDAMAGE, POWERUP.BERSERK))
{
}
```

---

## GetEntTeam

Gets the team id that the entity belongs to.

**Parameters:** (GameEntity/GameId)

**Returns:** Team Id of entity, or null if error

**Example:**

```
if(GetEntTeam(someent) == TEAM.RED)
{
}
```

---

## GetEntWorldAABB

Gets the world AABB for an entity.

**Parameters:** (GameEntity/GameId, AABB<optional>)

If you pass an AABB as the 2nd parameter, that object will be filled in instead of a new AABB returned. This can save memory allocations in a script that calls the function often by re-using the same object.

**Returns:** true if AABB passed as 2nd parameter and filled in successfully, if no AABB provided, returns AABB for entity. Both return null on an error.

### Example:

```
entAABB = GetEntWorldAABB(someent);  
if(entAABB)  
{  
    // do something  
}  
// re-use the same AABB later  
if(GetEntWorldAABB(someent, entAABB))  
{  
}
```

---

## GetEntityName

Gets the name of an entity.

**Parameters:** (GameEntity/GameId)

**Returns:** name of entity, or null if error

### Example:

```
entName = GetEntityName(someent);  
if(entName)  
{  
    if(entName == "SomeName")  
    {  
        // do something  
    }  
}
```

---

## GetEntName

**See:** [GetEntityName](#)

---

## GetEntityLocalSpace

Converts a world position into a local space position of a specified entity.

**Parameters:** (GameEntity/GameId, Vector3 world position)

**Returns:** Vector3 local space position, or null if error

**Example:**

```
worldpos = Vector3(10,10,10);  
localpos = GetEntityLocalSpace(someent, worldpos);
```

**See Also:** [GetEntityWorldSpace](#)

---

## GetEntityWorldSpace

Converts a local space position into a world space position of a specified entity.

**Parameters:** (GameEntity/GameId, Vector3 local position)

**Returns:** Vector3 world space position, or null if error

**Example:**

```
localpos = Vector3(10,10,10);  
worldpos = GetEntityWorldSpace(someent, localpos);
```

**See Also:** [GetEntityLocalSpace](#)

---

## GetEntPosition

Gets the world position of an entity.

**Parameters:** (GameEntity/GameId)

**Returns:** Vector3 world position, or null if error

**Example:**

```
p = GetEntPosition(someent);
```

---

## GetEntRotationMatrix

Gets the Matrix3 full transform of an entity.

**Parameters:** (GameEntity/GameId, Matrix3<optional>)

If you pass an Matrix3 as the 2nd parameter, that object will be filled in instead of a new Matrix3 returned. This can save memory allocations in a script that calls the function often by re-using the same object.

**Returns:** true if Matrix3 passed as 2nd parameter and filled in successfully, if no Matrix3 provided, returns Matrix3 for entity. Both return null on an error.

### Example:

```
entMat = GetEntRotationMatrix(someent);  
if(entMat)  
{  
    // do something  
}  
// re-use the same Matrix3 later  
if(GetEntRotationMatrix(someent, entMat))  
{  
}
```

---

## GetEntVelocity

Gets the world velocity of an entity.

**Parameters:** (GameEntity/GameId)

**Returns:** Vector3 world velocity, or null if error

### Example:

```
vel = GetEntVelocity(someent);
```

---

## GetEntityInSphere

Finds an entity within a radius around a point that matches a particular class Id. It is set up so that it can be used in a loop to find all entities.

**Parameters:** (Vector3 position, radius, classid, start entity<optional>)

**Returns:** GameEntity found, or null if none found



## Example:

```
p = Vector3(10,10,10);
radius = 20;
ent = GetEntityInSphere(p, radius, CLASS.ANYPLAYER);
dowhile(ent)
{
    // do something with it?
    // get the next one found
    ent = GetEntityInSphere(p, radius, CLASS.ANYPLAYER, ent);
}
```

---

## GetEntityStat

Generic function for getting information about an entity, by name.

**Parameters:** (GameEntity/GameId, stat name)

**Returns:** stat, or null if error or stat doesn't exist. The type depends on the stat.

## Example:

```
kills = GetEntityStat(someent, "kills");
deaths = GetEntityStat(someent, "deaths");
xp = GetEntityStat(bot.GetGameEntity(), "xp");
```

---

## EntityKill

Kills an entity. Requires cheats to be enabled.

**Parameters:** (GameEntity/GameId)

**Returns:** true if successful, false if not

## Example:

```
EntityKill(someentity);
```

**See Also:** CheatsEnabled

---

## GetTeamStat

Generic function for getting information about an team, by name.

**Parameters:** (Team Id, stat name)

**Returns:** stat, or null if error or stat doesn't exist. The type depends on the stat.

**Example:**

```
score = GetTeamStat(Team.RED, "score");
```

---

## GetGameEntityFromId

Converts a GameId to a GameEntity.

**Parameters:** (GameEntity)

**Returns:** GameId

**Example:**

```
ent = GetGameEntityFromId(2);
```

**See Also:** GetGameIdFromEntity

---

## GetGameIdFromEntity

Converts a GameEntity to a GameId.

**Parameters:** (GameEntity)

**Returns:** GameId

**Example:**

```
id = GetGameIdFromEntity(someent);
```

**See Also:** GetGameEntityFromId

---

## GetGameName

Gets the name of the currently running game.

**Parameters:** none

**Returns:** name of game

### Example:

```
if(GetGameName( ) == "Quake 4")
{
}
```

---

## GetModName

Gets the name of the currently running mod.

**Parameters:** none

**Returns:** name of mod

### Example:

```
if(GetModName( ) == "etpub")
{
}
```

---

## GetGameState

Gets the current state of the game.

**Parameters:** none

**Returns:** name of game state

### Example:

```
if(GetGameState( ) == "Playing")
{
}
```

---

## GetGameTimeLeft

Gets the current amount of time remaining in the game round.

**Parameters:** none

**Returns:** time left, in seconds

### Example:

```
if(GetGameTimeLeft() < 30)
{
}
```

---

## GetTime

Gets the current time elapsed in the game.

**Parameters:** none

**Returns:** time, in milliseconds

**Example:**

```
if(GetTime() > 30000)
{
}
```

---

## GetGoal

Gets a reference to a map goal by name.

**Parameters:** (name of map goal to get)

**Returns:** MapGoal if found, null if not

**Example:**

```
mg = GetGoal( "MAP_FLAG_redflag" );
```

**See Also:** GetGoals

---

## GetGoals

Gets any number of map goals that match a regular expression. Stores the matching reference in the first table parameter. Does **NOT** clear the table, so it may be called multiple times to accumulate results.

**Parameters:** (table, team, expression)

**Returns:** none

**Example:**

```
goals = table();  
mg = GetGoals(goals, TEAM.RED, "MAP_FLAG.*");
```

**See Also:** GetGoal

---

## GetGravity

Gets the current gravity of the game.

**Parameters:** none

**Returns:** gravity

**Example:**

```
grav = GetGravity();
```

---

## GetMapExtents

Gets the AABB map extents. Map extents are the bounds of the entire map.

**Parameters:** (AABB<optional>)

**Returns:** If no AABB passed as parameter, function will return the map AABB, otherwise the AABB parameter will be filled in with the results, and the function will return nothing.

**Example:**

```
aabb = GetMapExtents();  
// or  
AABB aabb;  
GetMapExtents(aabb);
```

---

## GetMapName

Get the name of the current map.

**Parameters:** none

**Returns:** name of map

**Example:**

```
if(GetMapName() == "oasis")
{
}
```

---

## GetMaxPlayers

Gets the current max players supported by the game.

**Parameters:** none

**Returns:** maxplayers

**Example:**

```
if(GetMaxPlayers() > 10)
{
}
```

---

## GetNumPlayers

Gets the current num players in the game.

**Parameters:** none

**Returns:** current players

**Example:**

```
if(GetNumPlayers() > 5)
{
}
```

---

## GetNumBots

Gets the current number of bots.

**Parameters:** none

**Returns:** current # bots

**Example:**

```
if(GetNumBots() > 10)
```

```
{  
}
```

---

## GetModVersion

Gets the version of the current mod.

**Parameters:** none

**Returns:** version of mod

**Example:**

```
if(GetModVersion() == "DOOM 1.3")  
{  
}
```

---

## GetPointContents

Gets the contents bits for a given position.

**Parameters:** (Vector3 position)

**Returns:** contents bits. Use bitwise manipulators along with constants from the global CONTENT table to check for specific content bits.

**Example:**

```
contents = GetPointContents(Vector3(10,10,10));  
if(contents & CONTENT.WATER)  
{  
}
```

---

## KickAll

Kicks all bots from the game.

**Parameters:** none

**Returns:** none

**Example:**

```
KickAll();
```

---

## KickBot

Kicks a bot by name.

**Parameters:** (name)

**Returns:** none

**Example:**

```
KickBot("Fred");
```

---

## MinBots

Sets the min bots.

**Parameters:** (# minbots)

**Returns:** none

**Example:**

```
MinBots(10);
```

**See Also:** MaxBots

---

## MaxBots

Sets the max bots.

**Parameters:** (# maxbots)

**Returns:** none

**Example:**

```
MaxBots(10);
```

**See Also:** MinBots

---



## OnTrigger

Registers a script function callback for a given trigger string.

**Parameters:** (trigger string, script function)

**Returns:** none

**Example:**

```
global myfunc = function()  
{  
};  
OnTrigger( "Allies have built the Old City Water Pump!", myfunc);
```

**See Also:** Omni-bot Map Scripting

---

## RegisterDefaultProfile

Registers a default profile that will be loaded for any bot that joins a specified class.

**Parameters:** (Class Id, script name)

**Returns:** none

**Example:**

```
RegisterDefaultProfile(CLASS.SOLDIER, "def_bot.gm");
```

---

## ServerCommand

Executes a server command. Requires cheats to be enabled.

**Parameters:** none

**Returns:** true if cheats are enabled, false if not

**Example:**

```
ServerCommand("map oasis");
```

**See Also:** CheatsEnabled

---

## SetAvailableMapGoals

Enables/Disables the available status of one or more map goals that match a regular expression, for a specific team.

**Parameters:** (Team Id, enable/disable, mapgoal expression string)

**Returns:** none

**Example:**

```
SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK.*" );
```

**See Also:** GetGoal, GetGoals

---

## SetMapGoalProperties

Passes a table full of properties along to all matching map goals to attempt to set their properties.

**Parameters:** (expression, table of properties)

**Returns:** none

**Example:**

```
// Set the camp time for all attack and defend goals
SetMapGoalProperties( "ATTACK_.*", {mincamptime=15, maxcamptime=30});
SetMapGoalProperties( "DEFEND_.*", {mincamptime=15, maxcamptime=30});
```

**See Also:** Omni-bot Map Goals for a list of properties for each map goal type.

---

## SetGoalGroup

Sets the goal or table of goals group name.

**Parameters:** (goalname or table, groupname)

**Returns:** none

**Example:**

```
SetGoalGroup( "SomeGoalName", "SomeGroupName" );
```

```
groupTable =
{
    "ROUTE_route1",
    "ROUTE_route2",
```

```
    "someothergoal",  
};  
  
SetGoalGroup( groupTable, "MyGroup" );
```

**See Also:** Util.DisableGroup, Util.EnableGroup, Util.GetGroup, Util.SetGroup, Util.ShowGroup

---

## TraceLine

Performs a traceline collision test, and returns the results in a table. TraceLine calls are useful for testing visibility, line of sight, or for getting a collision point along a line.

**Parameters:** (Vector3 start, Vector3 end, AABB, collision mask, ignore user GamelId, use PVS)

- Vector3 start - Start position of trace line.
- Vector3 end - End position of trace line.
- AABB - Pass an AABB if you with the line to have volume, otherwise pass null.
- Collision Mask - Mask of collision types to test. See global TRACE table.
- GamelId - Ignore this GamelId in the traceline. Usually the entity or bot it originates from.
- UsePVS - true to use PVS, false to ignore PVS

Always pass true to UsePVS if you need collision information from the call to TraceLine. UsePVS=false is useful when all you need to know is whether or not something is blocking the path from start to end, it will not give you the collision position. It is slightly faster in most cases to use PVS as an early out test, but doing so limits the amount of useful information you get back from the function. In many cases you don't need this information, so the extra speed is useful.

**Returns:** table of results

Results table contains the following information.

- fraction - 0.0 to 1.0 The ratio along the line the collision took place. Not always valid if UsePVS=true. Example: 0.5 means ray hit half way between the start and end.
- startsolid - true/false The traceline started inside a solid object. Not always valid if UsePVS=true

Additional properties are available if UsePVS=false and there was a collision.

- entity - May be null, the GameEntity that was hit in the collision.
- normal - The Vector3 normal of the collision point.
- end - The Vector3 world position of the collision.

## Example:

```
// Do a traceline along a bots facing 1024 units out  
start = b.GetEyePosition();  
end = start + b.GetFacing() * 1024;  
tr = TraceLine(start, end, null, TRACE.SHOT, b.GetGameId(), false);
```

```
// the fraction value tells us of a collision or not. If it's 1, there was no collision. If
less than 1 there was a collision.
if(tr.fraction < 1)
{
    // tr.startsolid
    // tr.entity
    // tr.normal
    // tr.end
}
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Bot\\_Library](http://omni-bot.com/wiki/index.php?title=Bot_Library)"

This page has been accessed 1,659 times. This page was last modified 06:33, 25 August 2008.

# Map Scripting Enemy Territory

From Omni-bot Wiki

## Contents

- 1 Standard Format
  - 1.1 Map Table
  - 1.2 OnMapLoad
  - 1.3 OnBotJoin
  - 1.4 Triggers
    - 1.4.1 Supported Triggers
- 2 Native Functions
  - 2.1 ChangeSpawnPoint
  - 2.2 GetGameTimeLeft
  - 2.3 GetGameType
  - 2.4 GetReinforceTime
  - 2.5 MaxViewDistance
  - 2.6 SetAvailableMapGoals
  - 2.7 SetGoalPriority
  - 2.8 SetMapGoalProperties
  - 2.9 TargetBreakableDist
- 3 Utility Functions
  - 3.1 ETUtilities
    - 3.1.1 ETUtil.ChangeClass
    - 3.1.2 ETUtil.ClearMainGoals
    - 3.1.3 ETUtil.ClearSecondaryGoals
    - 3.1.4 ETUtil.CountClass
    - 3.1.5 ETUtil.CountTeam
    - 3.1.6 ETUtil.DisableGoal
    - 3.1.7 ETUtil.EnableGoal
    - 3.1.8 ETUtil.LimitToClass
    - 3.1.9 ETUtil.NoSnipe
    - 3.1.10 ETUtil.RandomSpawn
    - 3.1.11 ETUtil.SelectWeapon
    - 3.1.12 ETUtil.SetPrimaryGoals
    - 3.1.13 ETUtil.ShowActiveGoals
    - 3.1.14 ETUtil.StopSniping
    - 3.1.15 ETUtil.SwitchWeapon
  - 3.2 Utilities
    - 3.2.1 Util.AddInvVehicle
    - 3.2.2 Util.AliveCount
    - 3.2.3 Util.DisableGroup
    - 3.2.4 Util.DistanceView
    - 3.2.5 Util.EnableGroup
    - 3.2.6 Util.GetGroup
    - 3.2.7 Util.OnTriggerPosition
    - 3.2.8 Util.RemoveGoal
    - 3.2.9 Util.SetGoalOffset
    - 3.2.10 Util.SetGoalPosition

- 3.2.11 Util.SetGroup
- 3.2.12 Util.ShowGroup
- 3.2.13 Util.SetMaxUsersInProgress
- 3.2.14 Util.SetPositionGoal
- 3.2.15 Util.ShowGoalInfo
- 3.2.16 Util.ShowGoalName
- 3.2.17 Util.ShowGoalOffset
- 3.2.18 Util.AddUsePoint
- 3.2.19 Util.AddUseWp
- 4 Conditionals
  - 4.1 Case Study
- 5 Scripting Tools
  - 5.1 makegm
- 6 Debugging Tools
  - 6.1 debugbot
  - 6.2 ScriptDebug
  - 6.3 GameMonkey interpreter
  - 6.4 print
  - 6.5 show\_goals
  - 6.6 show\_goalroutes
  - 6.7 draw\_goals
  - 6.8 draw\_goalroutes

## Standard Format

All official map scripts share a common layout. Having a standard for map scripts has several benefits; ease of documentation, debugging, and implementation of scripted goals are among those benefits. A typical ET map script will consist of one Map table and two functions that are automatically called by Omni-Bot:

```
global Map =
{
};

global OnMapLoad = function()
{
};

global OnBotJoin = function( bot )
{
};
```

Using makemapgm will ensure that a map script is in a standard format.

## Map Table

The Map Table has become an important part of an ET map script as some scripted goals rely on its existence. It also provides a 'safe' place to store map specific information. Typically it will consist of map variables, triggers / functions, and in some cases additional tables:

```
global Map =
```

```
{
    SomeGoal = "BUILD_some_construct",
    someVar = true,
    someTable =
    {
        someVar = false,
    },

    some_trigger = function(trigger)
    {
        print("some_trigger");
    },
};
```

It is important to note that when you add to a table you use a comma and not a semi-colon. Calling functions or referencing variables within the example table is done like this:

```
Map.someVar = false; //changes the value of someVar to false
print(Map.SomeGoal); //will print BUILD_some_construct
Map.someTable.someVar = true; //changes the value of someTable.someVar to true
```

The key thing to note is that you need to put Map. in front of whatever you want to call or reference within the Map table.

## OnMapLoad

The OnMapLoad function is used to initialize settings for the map. It is automatically called by Omni-Bot whenever a map loads (i.e. /map\_restart). Setting up triggers, goal availability, and goal bias' are the most common jobs performed by this function:

```
global OnMapLoad = function()
{
    OnTrigger( "Some Team did Something!", Map.some_trigger );
    SetAvailableMapGoals( TEAM.AXIS, false, Map.SomeGoal );
    SetGoalPriority( Map.SomeGoal, 1.1 );
};
```

## OnBotJoin

The OnBotJoin function is automatically called by Omni-Bot when a bot joins. Typical usage for OnBotJoin is for setting specific properties on the bot that you want to be map specific. View distance and breakable object distance are the two most commonly used:

```
global OnBotJoin = function( bot )
{
    bot.MaxViewDistance = 2500;
    bot.TargetBreakableDist = 150.0;
};
```

Note that it is not necessary to loop through the BotTable every time this is called as this function is called each time a bot joins.

## Triggers

A trigger is an event that is recognized by Omni-Bot in game. With map scripting, triggers can be set up to perform operations when a map event occurs. There are two steps to setting up triggers; defining the trigger and setting up the trigger function.

Triggers are most commonly defined in OnMapLoad using the OnTrigger function:

```
global OnMapLoad = function()  
{  
    OnTrigger("string", function);  
};
```

The "string" parameter is usually found using the wm\_announce messages seen during the game. This must be matched exactly including capitalization, punctuation, and color codes (if any). To see the wm\_announce message requires the waypointer to either play through the map and perform the objectives or look through the map script that is included inside a map's pk3 file.

If the wm\_announce messages appear in non-standard color, the latter method is often more convenient. The map's native script (not to be confused with the Omni-bot map script) is usually the file <mapname>.script in the /maps/ folder inside the pk3 file. Open the pk3 file with some archive utility, e.g. IZArc.

Another method of finding the string parameter is using the command **/bot debugtriggers**. This command will output to console everytime a recognized event occurs. The syntax will be similar to:

```
<---> Trigger: TagName: The Tank has been repaired! Action: announce Entity: 0x7016dd8c  
Activator: 0
```

The string just after TagName: is what the OnTrigger function will expect as the string parameter. For additional information about debugtriggers see the [Omni-bot\\_Command\\_Reference#debugtriggers](#) page. This method is required for maps that may not have wm\_announce messages that correspond to recognized map events.

Once you've collected the triggers you need, the contents of the console can be written to a plain text file by using the command **/condump mytextfile**. The file will be located in the omnibot folder under the game installation folder. By copying the trigger names from that file instead of typing them, the risk of typing errors is greatly reduced.

The function parameter of the OnTrigger function is the name of a function that the waypointer creates for the particular trigger. These can be named anything the waypointer wants, but should be somewhat intuitive.

Once the string parameter has been identified and a function name has been determined, the trigger definition will look something like this:

```
global OnMapLoad = function()  
{  
    OnTrigger("The Tank has been repaired!", Map.tank_repaired );  
};
```

The last part of the setup is to create the trigger function that will contain code for the game to process each time the event occurs. In all official map scripts, this is done inside the Map table:

```
global Map =  
{
```



```

tank_repaired = function( trigger )
{
    //things to do when this event occurs
    print("tank_repaired");
},
};

```

It is important to note is that the function is inside the map table and should have a comma outside the closing bracket and not a semi-colon. The print statement inside the trigger function is not necessary, but is a good way to test that the trigger is working. As an additional service to the users, you can comment these print statements out when you're done testing, so their console isn't flooded with information that doesn't mean much to them.

## Supported Triggers

Most events in ET will fall into the following Omni-Bot recognized categories:

- allied\_complete
- allied\_default
- allied\_failed
- announce
- announce\_icon
- axis\_complete
- axis\_default
- axis\_failed
- closed
- defused
- dynamited
- exploded
- mover\_goto x y z
- opened
- repair\_mg42
- returned
- round end
- stolen
- team\_announce
- thirty second
- two minute

If you are unsure if the event you want to set up a trigger for falls into one of these categories, use `/bot debugtriggers` in game.

**NOTE: For the returned trigger (i.e. documents returned) use `/bot debugtriggers` output for when the flag is returned by expiring (no player returns it). If the regular `wm_announce` message is used, the event that occurs when the flag expires will not be used. It is recommended to manually steal the flag, `/kill` and wait for them to be automatically returned. An example can be found in `radar.gm`. Most of the time it will be something like "Flag returned flag!"**

## Native Functions

This section consists of Omni-Bot functions specifically designed for use in map scripts.

### ChangeSpawnPoint

syntax: `bot.ChangeSpawnPoint(int spawnptid);`  
example: `bot.ChangeSpawnPoint(1);`

note: the number of the spawn point is map dependant and may require either searching for spawn selection configs or testing the numbers.

## GetGameTimeLeft

syntax: `GetGameTimeLeft();`  
returns: Time left in the game in seconds

example: can be used in maps where you may want the bots to focus on major objectives if there isn't much time left.

## GetGameType

syntax: `GetGameType();`  
returns: The current game type

## GetReinforceTime

syntax: `bot.GetReinforceTime();`  
returns: Time left before the bots next spawn

example: can be used in maps where you may want to exec a command if the bot is close to a new respawn.

## MaxViewDistance

syntax: `bot.MaxViewDistance = <distance>;`  
example: `bot.MaxViewDistance = 2500;`

note: typically set in OnBotJoin as this sets the property individually

## SetAvailableMapGoals

syntax: `SetAvailableMapGoals( Team, true / false, goalname );`  
example: `SetAvailableMapGoals( TEAM.AXIS, true, "MAP_FLAG_someflag" );`  
example: `SetAvailableMapGoals( TEAM.ALLIES, false, Map.Flag );`  
example: `SetAvailableMapGoals( TEAM.AXIS, false, "DEFEND.*" );`

note: the goalname parameter supports expressions

note: this command is the simpler alternative to getting a goal, then setting the availabilty.  
it's not necessary to do something like this:

```
somevar = GetGoal( "SOMEGOAL" )
if (somevar)
{
    somevar.SetAvailable( TEAM.ALLIES, true );
}
```

## SetGoalPriority

```
syntax: SetGoalPriority( goalname, priority, team, class, <optional persistent> );  
example: SetGoalPriority( Map.Flag, 1.0, 0, 0 ); //all teams and all classes  
example: SetGoalPriority( "DEFUSE_somedyno.*", 0.0, TEAM.AXIS, CLASS.ENGINEER, true ); //the  
persistent parameter is used for dynamic goals that may not have been created yet
```

## SetMapGoalProperties

```
syntax: SetMapGoalProperties( Goal, table )  
example: SetMapGoalProperties( "ATTACK_.*", {mincamptime=15, maxcamptime=30} );  
  
note: see OnMapLoad in goldrush.gm for example usage
```

## TargetBreakableDist

```
syntax: bot.TargetBreakableDist = <distance>;  
example: bot.TargetBreakableDist = 100;  
  
note: typically set in OnBotJoin as this sets the property individually  
note: used to allow bots to target breakables like windows. should be set to a relatively low  
number.
```

## Utility Functions

The functions listed in this section are custom functions created to support specific scenarios in game. They are located in ~/omni-bot/et/scripts and ~/omni-bot/global\_scripts respectively.

### ETUtilities

ET utility functions are located in the ~/omni-bot/et/scripts/et\_utilities.gm file.

### ETUtil.ChangeClass

```
syntax: ETUtil.ChangeClass( team, originalclass, newclass, revert, maxbots )  
example: ETUtil.ChangeClass( TEAM.ALLIES, CLASS.SOLDIER, CLASS.COVERTOPS, false, 1 );  
  
usage: used in braundorf_b4 to change one bot to covert ops for satcheling the side gate.  
Once satcheled, the function is called again with the revert flag set to true to  
have the bot change back to it's original class.
```

### ETUtil.ClearMainGoals

```
syntax: ETUtil.ClearMainGoals();  
usage: This function will deactivate all main goals for both teams.  
  
goals deactivated: AMMOCAB, CHECKPOINT, HEALTHCAB, BUILD, PLANT, FLAG, MOUNTMG42, MOVER
```

### ETUtil.ClearSecondaryGoals

```
syntax: ETUtil.ClearSecondaryGoals();
```

usage: This function will deactivate all secondary goals for both teams.

goals deactivated: AMMO, HEALTH, ARTILLERY, MOBILEMG42, REPAIRMG42, PLANTMINE

## ETUtil.CountClass

syntax: `ETUtil.CountClass( team, class )`

example: `ETUtil.CountClass( TEAM.ALLIES, CLASS.ENGINEER );`

usage: can be used to determine if a team has enough of a critical class for the map

## ETUtil.CountTeam

syntax: `ETUtil.CountTeam( team )`

example: `ETUtil.CountClass( TEAM.ALLIES );`

usage: used to count the number of bots on a give team. used in maps like et\_ice  
in a conditional statement for setting max users attacking / defending.

## ETUtil.DisableGoal

syntax: `ETUtil.DisableGoal(goalname, <optional true>);`

example: `ETUtil.DisableGoal("FLAG_someflag");`

usage: disables the goal for both teams. the optional true parameter is used to disable all  
goals except for ROUTE goals.

## ETUtil.EnableGoal

syntax: `ETUtil.EnableGoal(goalname);`

example: `ETUtil.EnableGoal("FLAG_someflag");`

usage: enables the goal for both teams.

## ETUtil.LimitToClass

syntax: `ETUtil.LimitToClass(goalname, team, class1, class2, class3);`

usage: used to limit specific goals to a specific class or classes

example: `ETUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER) //only soldiers on  
the allied team`

example: `ETUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER, CLASS.MEDIC) //only  
soldiers and medics`

## ETUtil.NoSnipe

syntax: `ETUtil.NoSnipe(bot);`

usage: covert ops bots will not choose a garand or k43

note: this is typically called in OnBotJoin

## ETUtil.RandomSpawn

syntax: `ETUtil.RandomSpawn(team, spawnpoint);`

usage: used to have bots randomly choose to spawn at the given spawnpoint

## ETUtil.SelectWeapon

syntax: `ETUtil.SelectWeapon(bot, weapon);`

usage: the given bot will switch to the given weapon if it is the correct class for the weapon

note: typically used in OnBotJoin to have soldiers choose a specific weapon

example: `ETUtil.SelectWeapon(bot, WEAPON.PANZERFAUST);`

## ETUtil.SetPrimaryGoals

syntax: `ETUtil.SetPrimaryGoals(priority);`

usage: used for setting priorities of common goals.

note: this is typically called in OnMapLoad and effects the following goals (in order of priority):

CAPPOINT FLAGRETURN PLANT CHECKPOINT FLAG

## ETUtil.ShowActiveGoals

syntax: `ETUtil.ShowActiveGoals();`

usage: shows active goals for both teams

note: should only be used for debugging

## ETUtil.StopSniping

syntax: `ETUtil.StopSniping();`

usage: all bots currently using a mauser will switch to a different weapon

note: this is typically called inside trigger functions

## ETUtil.SwitchWeapon

syntax: `ETUtil.SwitchWeapon(weapon);`

usage: all qualifying bots will switch to the given weapon

example: `ETUtil.SwitchWeapon(WEAPON.PANZERFAUST); //all soldiers will switch to panzer`

note: typically used in trigger functions

## Utilities

Utility functions are located in the `~/omni-bot/global_scripts/utilities.gm` file.

## Util.AddInvVehicle

syntax: `Util.AddInvVehicle( goalname )`

usage: adds a `vehicle` to the Map.InvVehicle `table`

used for script goals in cases where the vehicle is invulnerable, but shows as "dead"

## Util.AliveCount

syntax: `Util.AliveCount( team, class )`

usage: returns the number of bots alive on a team with a given class

## Util.DisableGroup

syntax: `Util.DisableGroup(groupname, team);`

example: `Util.DisableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to disable a set of goals in the given group for a given team

## Util.DistanceView

syntax: `Util.DistanceView(<optional off|0>);`

usage: `/bot dist`

`/bot dist off`

`/bot dist 0`

note: used for determining bot.MaxViewDistance settings. aim at a position as far as you can see and type `/bot dist`

## Util.EnableGroup

syntax: `Util.EnableGroup(groupname, team);`

example: `Util.EnableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to enable a set of goals in the given group for a given team

## Util.GetGroup

syntax: `Util.GetGroup(groupname);`

example: `Util.GetGroup("somegroupname");`

usage: this function will return a table of goals belonging to the given group name

## Util.OnTriggerPosition

syntax: `Util.OnTriggerPosition( goalname, wpname, tolerance, wpfunction )`

example: `Util.OnTriggerPosition( Map.Mover_train1, "depotyard", 200.0, Map.tug_depotyard );`

note: used for setting up positional triggers for movers.

## Util.RemoveGoal

syntax: `Util.RemoveGoal( goalname );`

example: `Util.RemoveGoal( "MOVER_truck" );`

note: this command will remove the goal from the map goal table

## Util.SetGoalOffset

syntax: `Util.SetGoalOffset( x, y, z, GoalName );`

example: `Util.SetGoalOffset( 0, 0, 30, "BUILD_construct" )`

usage: the x y and z parameters are added to the origin of the goal to move the location of where the bots will look for the goal.

## Util.SetGoalPosition

syntax: `Util.SetGoalPosition( x, y, z, GoalName );`

example: `Util.SetGoalPosition( 4534, 2168, -199, "BUILD_construct" )`

usage: used to give the goal a new origin.

note: using /devmap to load the map and issuing the /viewpos command in console will give your origin in the map

## Util.SetGroup

syntax: `Util.SetGroup(goalname, groupname);`

example: `Util.SetGroup("somegoalname", "somegroupname");`

usage: this function will add a given goal to a given groupname

## Util.ShowGroup

syntax: `Util.ShowGroup(groupname);`

example: `Util.ShowGroup("somegroupname");`

usage: this function will list all goals in the given group in the console.

## Util.SetMaxUsersInProgress

syntax: `Util.SetMaxUsersInProgress( Users, GoalNames );`

example: `Util.SetMaxUsersInProgress( 15, "CHECKPOINT.*" );`

usage: used to set the maximum number of bots going for a particular goal(s).

## Util.SetPositionGoal

syntax: `Util.SetPositionGoal( goalname1, goalname2 );`

example: `Util.SetPositionGoal( Map.Build_tank_construct, Map.Mover_tank );`

usage: sets the origin of one goal to match the origin of another. useful for centering construct goals on movers

## Util.ShowGoalInfo

syntax: `Util.ShowGoalInfo(goalname);`

command: `/bot sgi 'goalname'`

usage: used to print a goals current information in console including the goals health, status of the DEAD flag,  
and the goals position

## Util.ShowGoalName

syntax: `Util.ShowGoalName(radius, showOffset);`  
command: `/bot sgn <radius> <optional true>`

usage: issue the command `/bot sgn` to find the goalname of any goals within 100 units.  
Optionally expand the radius and  
find your current offset from the goal with `/bot sgn 500 true`

## Util.ShowGoalOffset

syntax: `Util.ShowGoalOffset(goalname);`  
command: `/bot sgo 'somegoalname'`

usage: used to determine a players current offset from a goal. the values given in console can be used for goals that  
require positional offsets

## Util.AddUsePoint

syntax: `Util.AddUsePoint(goalname, position);`  
example: `Util.AddUsePoint( "somegoalname", Vector3(123, 456, 789) );`

usage: used to add a Use Point for a goal at a given position.

## Util.AddUseWp

syntax: `Util.AddUseWp(goalname, waypointname);`  
example: `Util.AddUseWp( "somegoalname", "somewaypointname" );`

usage: used to add a Use Point for a goal at a given waypoints position.

## Conditionals

Enemy Territory is a complex game type. Maps are not linear and events do not necessarily happen in a specific order. Conditional statements are necessary in some cases to ensure that availability of goals is consistent with the waypoints idea of a map flow. A conditional statement begins with the 'if' function. In script, it is used to check given paramaters before executing some code:

```
if ( a == b )  
{  
    //do something  
}
```

In this example it checks if something is equal to something else before executing any code in the following brackets. If something does not equal something else, anything inside the brackets will be ignored. This conditional can be extended to make something happen if the first conditional was not met with the 'else'



command:

```
if ( a == b )
{
    //do something
}
else
{
    //do something else
}
```

And finally, this can be extended even further using the 'else if' command:

```
if ( a == b )
{
    //do something
}
else if ( a == c )
{
    //do something else
}
else
{
    //do something else
}
```

Conditionals can get more complex by adding additional checks. The following example checks if both are true before executing the code in the example:

```
if ( a == b && c == d )
{
    //do something
}
```

If a check is dependant on one of many checks being true, the 'or' operator is used:

```
if ( a == b || c == d )
{
    //do something
}
```

To test if something is true or false, the syntax used is:

```
if ( !something )
{
    //do something
}
```

The apostrophe in front of the parameter checks if something is false while a `!=` checks if something doesn't equal something else:

```
if ( a != b )
{
```

```
    a = b;
}
```

Checking if a value is greater or lesser than another value is done as follows:

```
if ( a > b || c < d )
{
    //do something
}
```

In ET maps where the map flow is not pre-determined, it is important to add conditionals to some triggers in which goals are activated. While the bots order of objective completion can be set by the waypointer, the waypointer can not predict when a human player may complete a particular objective. So if goals are activated with a specific order in mind and a human player activates a goal out of turn, you may have goals available to bots earlier or later than intended; potentially breaking the gameplay.

## Case Study

**Scenario:** Map is Fueldump and goals are being activated or deactivated based on the Main Bridge status. When the bridge is built, attack and defend goals are activated and when it is destroyed, goals are deactivated. To add to the complexity, some of the same goals are activated / deactivated based on the status of the footbridge.

The first thing that needs to be done is to create some variables for use in the conditional statements. Official map scripts add these variables to the Map table:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,
};
```

The next step is to set these variables in the appropriate triggers:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built
    },
}
```

```

    },

    bridge_Destroyed = function( trigger )
    {
        Map.BridgeStatus = 0; //not built
    },

};

```

Now that we have status' updating correctly, conditional statements can be added to help ensure goals are active at the appropriate time:

```

global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built
        SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built

        //if both bridges destroyed, shift the defense back
        if ( Map.BridgeStatus == 0 )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
        }
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built
        SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
        SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Bridge.*" );
    },

    bridge_Destroyed = function( trigger )
    {
        Map.BridgeStatus = 0; //not built

        //if both bridges destroyed, shift the defense back
        if ( Map.FootBridgeStatus == 0 )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
        }
    },

};

```

All looks good at this point. At the beginning of the map, the allied bots will be shifting correctly based on both bridges status. There is a potential problem here though. What happens if someone builds or destroys a bridge after the tank is through the tunnel? The goals back by the bridge will be activated again. The solution is to pick a major event to check against and add that to our conditionals. In this case we will check the status of the Tunnel door before activating those goals:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,
    Tunnel_Doors = true, //doors intact

    tunneldoors_Destroyed = function( trigger )
    {
        Map.Tunnel_Doors = false;
    },

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built

        if ( Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
        }
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built

        //if both bridges destroyed and doors intact, shift the defense back
        if ( Map.BridgeStatus == 0 && Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge_.*" );
        }
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built

        if ( Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Bridge_.*" );
        }
    },

    bridge_Destroyed = function( trigger )
    {
```

```

Map.BridgeStatus = 0; //not built

//if both bridges destroyed and doors intact, shift the defense back
if ( Map.FootBridgeStatus == 0 && Map.Tunnel_Doors )
{
    SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
    SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
}
},
};

```

For a complete example of conditional usage, look through any of the map scripts released with version 0.65.

## Scripting Tools

### makegm

makemapgm is a command that autogenerates a skeleton map script. Usage of this command will ensure a standard format and save time. Using the command is fairly straight forward:

Step 1: Load the map you want a script for

Step 2: Type /bot makegm in console. It will display a message if executed correctly

Step 3: Copy the mapname.gm file from ~/omni-bot/et/usr to ~/omni-bot/et/nav, Be sure to have backed up any mapname.gm file you may have been working on that exists in the nav folder.

The OnTriggers in OnMapLoad will need to be modified as makemap.gm does not find the information needed. Simply edit the "MISSING STRING" parameter of the OnTrigger function and the triggers should be working correctly.

## Debugging Tools

### debugbot

debugbot is a valuable tool to use if bots aren't behaving as expected. It will give success and failure messages in console for short and long term goals.

```

syntax: /bot debugbot all goals
or: /bot debugbot <bot-name> goals

```

It is recommended to have only one or two bots connected when issuing this command with the all parameter as it will give debug output for each bot.

### ScriptDebug

Script debugging is crucial as a high percentage of errors are syntax related. There are two ways to enable script debugging:

- /bot script\_debug 1
- [EnableScriptDebug\(true\)](#);

EnableScriptDebug(true); can be placed in et\_autoexec.gm while /bot script\_debug 1 will need to be executed in

console each time the map loads. If an error in script occurs while script debugging is enabled, the error will be listed in console in red text with the line number of the error.

note: in the omnibot.log, script errors will be written whether script debugging is enabled or not.

## GameMonkey interpreter

The GameMonkey download available at [www.somedude.net/gamemonkey/](http://www.somedude.net/gamemonkey/) contains an interpreter for gm files (gme.exe) that can be used as a syntax checker. With this, you won't have to start the game only to find you forgot a closing bracket or a semicolon.

Ideally, use this with an editor that can run command line applications and capture their output, such as SciTE or PSPad.

## print

print is a low level means of debugging a script. It can be used in cases where you want to test if a trigger is working correctly or if the map script has loaded:

```
global OnMapLoad = function()  
{  
    print("OnMapLoad");  
}
```

The print statement can also be used in-game via script\_run if you want to inspect the value of a variable:  
/bot script\_run "print(variable)"

## show\_goals

syntax: /bot show\_goals <optional>

usage: the optional parameter can be used to display one or several goals.

note: the optional parameter supports expressions. /bot show\_goals DEFEND.\* will give output for all DEFEND goals.

show\_goals can be useful to test whether or not goals are available when they are expected to be and that they have the expected bias setting. The output from show goals is similar to:

```
PLANT_sometarget -> 1000 serial 3 bias 1.00
```

The number just to the right of the arrow represents teams and status. For ET, only the first two matter. The first number is for Axis and the second is for Allies. 1 means it's available for that team while 0 means it is unavailable. In this case, the dynamite goal is currently available for the Axis team.

## show\_goalroutes

syntax: /bot show\_goalroutes <optional>

usage: used to list routes used for a give goal or goals

## draw\_goals

syntax: /bot draw\_goals [on|1|off|0] <optional>

usage: the optional parameter can be used to display one or several goals.

Note: the optional parameter supports regular expressions. /bot draw\_goals .\*BUILD.\* will draw BUILD goals.

This will draw the location of goals to your screen. Useful to see where bots "think" an object is located.

## **draw\_goalroutes**

syntax: draw\_goals on/off <optional goal name expression>

usage: used to draw the given goal(s) routes

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Map\\_Scripting\\_Enemy\\_Territory](http://omni-bot.com/wiki/index.php?title=Map_Scripting_Enemy_Territory)"

This page has been accessed 1,428 times. This page was last modified 21:31, 26 June 2008.

# User:DrEvil

From Omni-bot Wiki

## About Me & Bots

I am the primary developer of Omni-bot. I started Omni-bot in October or so of 2004 as both a project in which to gain valuable AI programming experience, but also as a project to feed my need to do FPS bot programming.

After a couple years of clanning in Team Fortress Classic, I went through a phase of being tired of playing against humans. From cheating, to immature trash talking, to bunny hopping, and other such experiences that are all too common when playing online, I went searching for a bot for TFC, as it was around the time where bots were booming for Half-life 1.

In Half-life 1, a clever developer who goes by botman was the first to discover that bots could be made for any Half-life mod by creating a library that sat between the engine and the mod dll. This was a first for bot developers, as traditionally in order to have bots in a 3rd party mod, one had to develop them with the mod source code.

As a result of this discovery by botman, the community of bot developers became very large for Half-life, spawning Counter-Strike bots in the double digits, bots for Natural Selection, Day of Defeat, and some other mods.

At the time I came looking for a TFC bot, I had a good deal of free time and was hoping to help one of the bot developers so as to speed up development. At this time I wasn't much of a coder myself, serving a 4 year Air Force term in Alaska. After contacting the TFC bot developers I could find, I only really got a good response from Redfox, of Foxbot. At the time, Foxbot was the more advanced of the available bots, and as a bonus, the developer seemed cool and willing to have me help out.

Throughout Foxbot development, I was basically responsible for the waypoints and scripts for all the stock TFC maps, as well as many popular custom maps, and of course huge amounts of testing and feedback to Redfox to ensure steady progress and an overall great quality bot. The community grew to be very large, we had an official Foxbot TFC server where we rolled out new features early to surprise regular players, and a very active website and forums.

After a couple years of helping with Foxbot, and after I left the military and went to school for game programming, I began to take on a more active development role with Foxbot. Free time for Redfox was very low, and it was perfect timing for me to be able to contribute



more significantly to Foxbot. I essentially took on lead development role for the last 4-5 releases of Foxbot, adding such features as Rocket Jumping, Concussion Jumping, Defend points, Pipe traps, etc.

Long story short, as is what happens with every game, eventually TFC popularity started declining, most likely as everyone migrated to Counter-Strike. Throughout Foxbot development, Redfox and I always talked about a more general bot framework, much like discussions that were happening in the BotsUnited forum about a proposed United Bot.

Unfortunately, discussion was about the only thing that came out of either attempt, and in 2004 as I finished my Bachelors in Game Programming I decided that I would start such a project myself.

I chose the Omni-bot name because the Omni prefix is fitting for a bot that is planned to support numerous games. Initially during very early Omni-bot development, I joined the ETF development team to develop bots for their Team Fortress mod for Enemy Territory. Pretty much all of the early Omni-bot development was for ETF, as I fleshed out the low levels of the common systems for the bot.

ETF released with competent bot support, though I would have liked to do more with them. During ETF development, the next logical step for games to support was Enemy Territory itself. Enemy Territory was and still is a popular multi-player only game released for free by Splash Damage from a canceled commercial game. As luck would have it, the author of an existing bot for Enemy Territory was interested in joining Omni-bot for Enemy Territory support. Magik joined the Omni-bot team of just me, to develop support for Enemy Territory in parallel to my own developments of ETF. After ETF released, I moved to help out with Enemy Territory as my focus as well, primarily because I was unable to release my own updates to ETF, because most would mean distributing an updated server dll as well as bot files.

Enemy Territory support is still alive and strong today, and the Omni-bot framework itself has evolved leaps and bounds over what it started as. Omni-bot has had its interface integrated with several popular Enemy Territory mods, such as etpub, jaymod, and noquarter. Omni-bot has also expanded past Enemy Territory into Doom 3, Quake 4, and Fortress Forever.

I've also picked up some new help over time, who have benefitted Omni-bot immensely.

- Magik - Started Enemy Territory support, hosts the website and code repository.
- Geekfeststarter - Tons of testing, waypointing, scripting, for Enemy Territory, Fortress Forever, Doom 3, and Quake 4.
- Crapshoot - Tons of testing, waypointing, scripting, for Enemy Territory, Doom 3, and Quake 4. Lots of documentation help on the Wiki.

Thanks to all these guys, and anyone else in the community that has contributed their time to waypointing, scripting, or helping other users. It is very much appreciated.

## Development Logs

Visit [Development Logs](#) for news on what the team is up to.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=User:DrEvil>"

This page has been accessed 315 times. This page was last modified 14:47, 28 July 2007.

# User:Magik

From Omni-bot Wiki

Magik's personal web site

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=User:Magik>"

This page has been accessed 186 times. This page was last modified 22:40, 27 July 2007.

# User:Geekfeststarter

From Omni-bot Wiki

## About Me

For a long time, computers have been a part of my life. I first started programming computers back in sixth grade in (GASP!) 1970. In high school I was ahead of most of the computer courses being taught and I was allowed to do my own private projects for my grades. In college I obtained a degree in Electrical Engineering, minoring in Computer Science. When I graduated, my choice was, design hardware or design software. I chose to design software and I have never regretted it nor have I ever been pressed to find work. I work for the 2nd largest CAD/CAM company, Siemens PLM Software Solutions. As a senior engineer I research new tools, operating systems, compilers, etc. I also get to play with the latest hardware from Intel, AMD, Nvidia, etc.

In the late 1980's, early 1990's when computer video games hit, I was instantly hooked and wasted more hours than I can remember playing games like King's Quest, Space Quest, Leisure Suit Larry, Police Quest etc. Then in 1992, Wolfenstein 3D was released and the FPS became my one true obsession. Doom and Doom2 followed in 1993 and 1994. After buying a 10 Mb coax network card and convincing friends and family to do the same, the first LAN parties were born for us, where each month we would hook up 4 PC's and spend hours late into the night, blowing each others brains out in DM and TDM or playing 4 vs the AI in coop mode. I remember in 1994, buying 16 Mb of DRAM for \$500 to keep Doom and Doom2 running! At the time there was no such name as LAN party, so my wife, surveying her house full of computer geeks, called it a Geek Fest, and even today we still call it such. As the years went by we played our share of Descent, Duke Nukem 3D, Wing Commander Armada, Xwing vs Tie Fighter, Jedi Knight, Quake, Unreal Tournament, Quake 2, Quake 3, Return to Castle Wolfenstein, Enemy Territory, Counter Strike, Counter Strike Source and Enemy Territory Quake Wars.

With the release of Duke Nukem 3D, I first became interested in working on modifications. At the time I remember making the shotgun more powerful to the point of a 1 shot kill, to making pipe bombs so powerful you could take out a whole street. Once I was flying in a Duke city level with the jet pack, I dropped an "improved" pipe bomb on my friend and set it off before it got to the ground. I took out myself as well, and my friend and I decided to use the mod at our next fest and it was a hit with the rest of our crew. Duke was the first time we hooked up 8 players and although it crashed a lot, it was fun as hell.

When Quake was released, all my spare time went out the window. I immediately started porting all of our favorite Doom and Doom2 levels. I then made a super QuakeC mod of every mod I liked (Plasma gun, BFG, grappling hook, Airplane, Mech Warrior, demolition charges, Reaper bots). I also ported a few Duke 3D and Dark Forces levels to Quake, using

the monthly fests to try out the maps and code. It was at this time that Quake was starting to bring people together in the Internet for FPS matches and I remember having a ton of fun playing online in those early days. At that time, few people had real connections, so more fun was to be had playing LAN then online. Between fests in order to play, Reaper bots would have to do. While porting maps to Quake, I used Reaper bots to help me balance older maps designed for 4 players, to Quake maps that could have 16. The guy who wrote the Reaper bots went on to do the bots for Unreal Tournament. The cool thing about the Reaper bots, was they were learning bots, that would write out paths and data for future runs once run on a map, and would use this data in subsequent runs.

As the years and games went by, I ported and coded to Half Life, Quake 2, Quake 3. Then Return to Castle Wolfenstein and Counter Strike came along, and these became the focus for me. For RTCW, I converted 3 SP maps to MP CP maps, mp\_forest, mp\_norway and mp\_dark. These were great maps, made by the RTCW developers, that were too good not to be MP. The fueldump map in ET, started as a SP map called forest, which the SD guys took, redesigned and winterized for ET. For a year I rented a server from Artofwar and ran a ton of custom RTCW maps. I wasted tons of time, playing every night with a crew of really good people and players.

Enemy Territory was then released, and for me, it was RTCW with some missing pieces. Adding prone, mortar, grenade rifle, covert, swapping out the chaingun for a mobile mg42, as well as adding buildable objectives and a vehicle to escort, ET had almost everything I wanted in a game. Once FIOS was setup in my area, I ran an ET server. It was at that time that I needed some bots, to run on my server and help draw people to it. At the time I had d/l'ed Fritz Bot ET, but I was unimpressed with it. I then took a look at Omnibot, and I immediately liked it. After lurking in the forums for several months, I talked with DrEvil and inquired how I could help. For the last 2 years or so, I have been helping DrEvil get this puppy out!

Today, I help DrEvil with Omnibot development, occasionally dipping into the code, but mainly with waypointing and scripting. I created a testing system that automatically runs bots through maps, to help us debug code, scripts and waypoints. I ported Doom 3 with help from DrEvil to Omnibot, and sooner or later we will release it when we are happy with it. I have also started a port to Enemy Territory Quake Wars, and who knows maybe someday we can see if we can out do the bots that came with the game. I have also been working on the RTCW and Quake4 ports. Finally, I have been helping DrEvil with Fortress Forever, doing training scripts, waypointing and map scripts and anything else that I can do.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=User:Geekfeststarter>"

This page has been accessed 55 times. This page was last modified 13:35, 16 February 2008.

# User:Crapshoot

From Omni-bot Wiki

## About Me

I am currently a beta tester, waypointer, and scripter for Omni-Bot. My first interest in bots came with the introduction of the first RTCW bot called wolfbot in 2003. After playing for a while, waypointing interested me to the point that it became a bit of a hobby.

After the second release of wolfbot, the project suddenly died as the developer, Kaji disappeared. Waypointing continued for some time after the project died, but missing features resulted in the community of waypointers dieing as well.

In late 2004, a buddy of mine turned me on to a small project headed up by Maleficus. It was a limited CTF mode bot for RTCW with support for 2 maps when first released. Immediately I recognized the potential for the bot and asked to join the team as waypointer and beta tester for Fritzbot.

A nice team of people consisting of Denny, the bindlestiff, 420Blunt, Maleficus, and myself worked for a year on developing full bot support for RTCW. Maleficus was the sole coder while the rest of us waypointed and beta tested as well as made feature requests.

In late 2005, after numerous user requests, Maleficus submitted and began a port to support Enemy Territory. Fritzbot ET was released in early 2006 and quickly became extremely popular. Ultimately the work on this project resulted in Maleficus joining ID Software to help develop a bot for ET QuakeWars. Sadly, it meant work on Fritzbot ET became very limited.

At this time I decided to research Omni-Bot as the idea of a bot framework interested me. The release of 0.60 answered some important questions I had about the bot in general and soon afterwards I began looking at it closer. After familiarizing myself with gm scripting through hands on testing and usage of some nice examples provided by Bladen and Jaskot, I asked DrEvil if there was a way I could contribute to the project.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=User:Crapshoot>"

This page has been accessed 272 times. This page was last modified 04:18, 21 May 2007.

# Omni-bot 0.66

From Omni-bot Wiki

## Contents

- 1 Omni-bot 0.66 Changelog
- 2 Updates
  - 2.1 Bot
  - 2.2 Scripts
  - 2.3 Waypoints

## Omni-bot 0.66 Changelog

### Updates

#### Bot

- Fixed Medics healing enemy teammates & fieldops giving ammo to enemy.
- Fixed Voice macro events going to everyone, instead of just team.
- Fixed bots not tapping out in some rare-ish circumstances.
- Fixed a crash when running very large scripts(warbell, goldrush\_ga, etc).
- Fixed SetBiasGoals so passing null doesn't set bias on every goal.
- Fixed source parameter of voice macro event.
- Fixed a redundant weapon script caching that would overwrite weapon properties previously set.
- Fixed empty directory being created when saving waypoints.
- Removed SetFinalFunction script function.

#### Scripts

- Extended useswitch\_goal to include support for LimitDistance, ExitConditions, PressOnce, and Debug
- Added Weapon aim error and aim offsets to difficulty.gm
- Fixed escorting of invulnerable vehicles

### Waypoints

- Adlernest
  - Improved pathing

- Braundorf\_b4
  - Fixed Logic bug that affected bots switching to covertops to destroy side gate
- Caen
  - Fixed availability of tank barrier construct after tank passed
  - Improved pathing
- Fueldump
  - Fixed availability of bridge construct after tank passed
- Goldrush
  - Fixed availability of truck barrier constructs after truck passed
- Oasis
  - Tweaked defense at guns
  - Fixed availability of goals after completion (wall and pumps)
  - Added map variable for dynamiting water pump 2 after wall is destroyed (default is false)
  - Increased priority of water tunnel routes
  - Improved response to dynamite planted at guns
- Radar
  - Removed blockable flags from windows in Allied spawn
  - Re-enabled side door mg and added routing to support allied cp build
- Railgun
  - Improved bots ability to construct controls
  - Improved pathing
  - Improved goal management
- Venice
  - Fixed boat barrier 2 construction
  - Improved pathing

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.66](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.66)"

This page has been accessed 5,254 times. This page was last modified 22:44, 23 June 2007.



# Omni-bot 0.65

From Omni-bot Wiki

## Contents

- 1 Omni-bot 0.65 Changelog
- 2 Framework Improvements
- 3 Features
- 4 Fixes
- 5 Misc

## Omni-bot 0.65 Changelog

### Framework Improvements

- Routing support. After putting it off for too long I've finally added routing support to Omni-bot. This means that alternate routes can be set up to get bots to take paths to goals other than the shortest distance. Setting up routes requires a bit of waypointing and scripting, but the capability opens up alot of opportunity in getting bots to take multiple approach points to a goal.
  - See [Routing Tutorial Here](#) Omni-bot Routing
- Added RequiresTargetOutside, RequiresShooterOutside, UseMortarTrajectory, PitchOffset, FuseTime weapon properties.
- Added SetBurstRange function to weapon for burst fire support.
- Added support for Use Points for Map Goals.
- Added GetLocalBounds, GetUsePoint, GetNumUsePoint script functions for Map Goals.
- Significantly improved bot loading process.
- Made bot log file use relevant game paths and not always root game folder.

### Features

- Quake 4 deadzone support in latest 1.41 patch.
- Tons of Quake 4 waypoints & scripts from crapshoot.
- Waypoint updater from internet database.
- Merged changes from NQ into ET interface.
- Fixed ammo cabinet goal being broken.
- Made defending bots in ET return dropped flags.
- Changed dropped flags in ET to use same name as normal flag, with \_dropped appended.
- Added a TAKE\_CHECKPOINT goal in ET to auto detect checkpoint goals(no need to set

them up as attack goals now).

- Added some more useful triggers in ET.
- Mounted weapons now have their own weapon scripts, so they are treated as a separate weapon than normal carried weapons.
- Added FireMode parameter to weapon script callbacks, changed 'this' to reference the bot using it.
- Made bot.Name property settable, to change the name of the bot.
- Added ServerCommand script function. Used for executing server console commands. This function is cheat protected.
- Added EntityKill script function. This function is cheat protected.
- Added Log script function.
- Added CheatsEnabled script function. Checks if cheats are enabled.
- Added BUILDABLE category for TF.
- Added VEHICLE\_NODAMAGE class for ET entities.
- Moved vehicles in ET to CAT.VEHICLE.
- Changed how vehicles are detected in ET. Should result in more detections of indestructible vehicles.
- Changed MAP\_VEHICLE to MAP\_MOVER in ET.
- Added omnibot\_playing cvar, which keeps count of number of bots in the game. Should display in server browsers.
- Added g\_omnibotflags cvar to ET for a bit of control over behavior. Add up the desired numbers from the following options.
  - 1 - Don't XP Save for bots.
  - 2 - Don't mount tanks. Bots shouldn't mount tanks.
  - 4 - Don't mount guns. Bots shouldn't mount mg42s.
  - 8 - Don't track bot count in omnibot\_playing cvar.
- Added ENTFLAG.POISONED for use in scripting bots behavior while poisoned.

## Fixes

- Fixed GetGameTimeLeft script command always returning 0.
- Fixed bots trying to mount broken mg42 on route to it.
- Improved ET sniper sniping behavior.
- Fixed bot not turning while reloading.
- Added a delay to considering cabinet goals if they are empty.
- Numerous fixes to debug window.
- Several stability fixes to scripting system.
- Vector3 scripting type now a value type, which completely eliminates dynamic memory allocation and thus garbage collection due to many temporary Vector3's being created.
- Fixed omnibot\_path usage.
- Merged EVENT.WEAPON\_FIRE and EVENT.WEAPON\_FIRE\_PROJECTILE.
- Fixed crash when trying to run 2 instances of ET due to debugger socket usage.
- Fixed passing 0 for any class to certain functions. now uses CLASS.ANYPLAYER

- Fixed corpse entity recognition including injured players / added CLASS. INJUREDPLAYER
- Fixed bot dontshoot command.
- Fixed chat events being sent to bots multiple times.
- Fixed ET client to disable cg\_omnibotdrawing if interprocess throws an exception.
- Fixed init time to print to console not just log.
- Fixed bots to not attempt to move or aim in dead or frozen state in ET(Should fix aiming after revive).
- Fixed Map Goal bounds to work for moving objects.
- Removed DynamicBounds script property. Bounds automatically follow position.
- Changed cl\_guid to use client index, rather than name in ET.
- Changed mover\_goto triggers to goalname\_goto, and the velocity as the action(Can use script function ToVector to convert to a vector).
- Fixed bot.IgnoreTarget if called on entities the bot doesn't yet know about.
- Fixed entity flags not being valid until 1 frame after spawn script callback.
- Fixed bots usage of walk button in ET.

## Misc

- Renamed PickNewPrimaryWeapon to ChangePrimaryWeapon.
- Renamed PickNewSecondaryWeapon to ChangeSecondaryWeapon.
- Added EnableRemoteDebugger option to config.gm.
- Added EnableInterProcess option to config.gm.
- Disable .vis file generation. Legacy feature, not used anymore. You can delete all .vis files from your nav directory.
- A bit more logging of useful info.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.65](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.65)"

This page has been accessed 1,983 times. This page was last modified 01:29, 8 October 2007.

# Omni-bot 0.61

From Omni-bot Wiki

## Omni-bot 0.61 Changelog

### Fixes

- Fixed bot using omnibot\_path for setting up the file system.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.61](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.61)"

This page has been accessed 248 times. This page was last modified 15:16, 14 April 2007.

# Omni-bot 0.6

From Omni-bot Wiki

## Contents

- 1 Omni-Bot 0.6 STABLE
- 2 Framework Improvements
- 3 Features
- 4 Fixes
- 5 Misc

## Omni-Bot 0.6 STABLE

### Framework Improvements

- New script defined weapon properties.
- New file system. Faster, more reliable, portable, and more secure. (<http://icculus.org/physfs/> )
- Set delay execute flag on most script callbacks, to prevent crashes if trying to kick a bot from an event for example.
- Converted entity flags to use BitField class.
- Removed attached console.
- Improved reliability of team/class events from interface. 7 ) Reduced bot view jerking.
- Removed all mod weapon classes in native code. All script driven now.
- New assert thats supports breaking, ignore once, ignore all.
- Updated navigation functions with better usage output.
- All waypoints now grab the players facing by default when added.
- Removed old broken msvc2003 project files from SDK.

### Features

- Added global callback function "SelectWeapons", to allow script to control bot weapon selection.
- Made script execution check full file path, then scripts directory, then global\_scripts directory.
- Added some simple avoidance of other bots.
- Added ET\_CLASSEX\_VEHICLE\_HVY script class type. Represents vehicles damagable by heavy weapons/explosives only. (ET)
- Added bot script\_run command, to run a script snippet from the games console. Use ` instead of `.

- Added echoTable script function.
- New global\_scripts folder can contain utility/library/global scripts that can be used or added to.
- Added Jaymod weapon scripts for its custom weapons.
- Replaced GetEntityKills and GetEntityScore with GetEntityStat, GetTeamStat.
- Added fully configurable waypoint colors with waypoint\_color command.
- Added plain jump flag which forces a jump when bot gets in radius, no checks like jumpgap, jumpoffset.
- Added global config table object for scripts to set global options like dump file/logging options.
- Added gmAutoHealthArmorInfo class to automatically update health/armor info for script use.
- Fixed movable waypoint initialization after mirroring.
- Added waypoint UID to error output for movables and bad mapgoal inits.
- Added revision command.
- Added waypoint\_viewfacing 1 command.
- Added support for mirror command to do multiple mirrors in 1 call.
- Added extra debug info on trigger output.
- Changed AddSignalThread script function to optionally take a true/false for auto delete thread or not.
- Added OnBotAutoJoin script callback for when minbots adds a bot. Allows script control of class/name/team of bot being added.
- Added waypoint\_translate function to move all waypoints.
- Changed autoradius to take an additional parameter to process current or all waypoints, also echoed settings used to console.
- Added PERCEPT\_FEEL\_PLAYER\_USE event.
- Added reasons to all goal success/failures if goal debugging on.
- Updated waypoint version file. Saves movable entity Id to file(experimental).
- Added script waypoint flag to run scripts for a waypoint when map loads.
- Added callbacks for bots "SelectTeam" and "SelectClass" so bot scripts can choose their start team/class.
- Improved visuals from draw\_goals command.
- Added auto facing set to all new waypoints set with bot waypoint\_add. Uses your current facing.
- Added thread info to bot script\_stats.
- Added an external GUI with a script console, profiler output, and message log. Enable with bot dwn
- Added IsStuck, and ResetStuckTime script functions.
- Added TransformVector, InverseTransformVector, Inverse functions to Matrix3 script type.(OMG MATH!)
- Added optional 2nd param to MoveTowards, which is a distance tolerance to 1st param. If within, function returns true.

## Fixes

- Fixed bot parsing all pk3/pk4 files in game folder. This accounted for long load times.
- Fixed interface forgetting weapon selections after warmup (ET).
- Fixed team setting from event at bot spawn.
- Fixed availability check in plant explosive goal.
- Fixed minor logic error in command argument concatenation.
- Fixed bug in vision system that caused target classes to be checked wrong.
- Added bias check on call arty goals (ET).
- Reduced .vis file size by 8x.
- Fixed TargetBreakableDistance property to be settable at any time.
- Rewrote debug line drawing in ET. Draws much faster without using entities.
- Fixed bots not releasing mount mg42 goal when made unavailable.
- Fixed waypoint visibility being rebuilt after a waypoint delete.
- Clear waypoint selection on radius change so it should immediately draw the updated info.
- Removed 'blocked' flags from goals in favor of a more reliable method of handling failed goals.
- Fixed check to abandon ctf goals if their availability changes.
- Fixed version command to give more meaningful information about bot version. Remove version string from interface.
- Removed hard coded 'fire in the hole' voice chats from grenade throws (ET).
- Fixed bots leaving game free'ing their name for re-use.
- Significant optimizations to entity lookups.
- Fixed some potential crashes with bad entities for event death, event feelpain.
- Reduced memory usage, both app and script.
- Removed 2nd messagebox from MiniDumper.
- Fixed a bug that could cause a corrupt .way when saving after using delete\_axis.
- Fixed touch sources giving visibility through walls.
- Fixed feel pain script event on null entities.
- Added optional bool to bot.GoTo Function, to append goal rather than replace goal.
- Removed disconnect waypoint commands, made connect versions toggle, to reduce commands and supporting code.
- Fixed ReleaseButton script function.
- fixed bug with choosing 0 desirability goals.
- Fixed bug where minbot added bot would have null name.
- Fixed potential memory leak with script threads sticking around when bot is kicked.
- Fixed bots aiming at targets when their weapon shooting is disabled.
- Fixed weapon\_fire script event to pass weapon.
- Fixed potential crashes in triggerinfo accessors.
- Fixed saving of waypoint properties.
- Fixed a problem where some mg42s might not get registered as goals(ET radar).
- Removed Panzerfaust and Aistrike waypoint commands( they weren't implemented at all )
- Significantly improved goal detection by caching position, facing, bounds. Some goals would get screwed bounds. (dual team objectives).

- Updated position of health entity in the GetHealth goal.
- Fixed goal flipping for CTF and Snipe goal.
- Raised height for blockable wall check.
- Fixed PostRecord parameter ordering.
- Merged a critical bug fix from Game Monkey Script.
- Fixed script command SelectBestWeapon.
- Improved debug line drawing significantly.
- Artificially bloated up map extents in ET interface by 2x(game has it wrong for some reason, wtf?)
- Heavily reduced script memory usage of GameEntity types. Now can also use them as table keys in script.
- Fixed a crash with auto health and armor script type.
- Improved Stuck detection.
- Added capability to compare gameentities to game id's for equality and non equality.

## Misc

- Game Monkey Script Remote Debugger!

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.6](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.6)"

This page has been accessed 189 times. This page was last modified 15:20, 14 April 2007.



# Omni-bot 0.532

From Omni-bot Wiki

## Omni-Bot 0.532 STABLE

- Fixed dump file generation on every shutdown of bot library(effected windows only, no change to linux, so no 0.532 linux.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.532](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.532)"

This page has been accessed 151 times. This page was last modified 17:55, 15 April 2007.

# Omni-bot 0.531

From Omni-bot Wiki

## Omni-Bot 0.531 STABLE

- Fixed a bug that caused subtle navigation problems, including frequently failed paths

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.531](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.531)"

This page has been accessed 110 times. This page was last modified 17:55, 15 April 2007.

# Omni-bot 0.53

From Omni-bot Wiki

## Omni-Bot 0.53 STABLE

- Fixed a crash if no goals were usable.
- Fixed a crash if the bot didn't have a valid current weapon.
- Fixed a crash if script accessed null values in trigger infos.
- Added ClearWatchEntity script function.
- Added OnBotAutoJoin script callback for controlling bot setup for auto added bots (minbots/maxbots)
- Removed messagebox on crashes, dmp file saved automatically now.
- Added bot script callback SelectTeam and SelectClass that is called when a bot is added with a non specified team and/or class.
- Removed etpub from installer.
- Removed some stray debug line drawing that could cause clients to error with Unknown Event: 131

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.53](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.53)"

This page has been accessed 129 times. This page was last modified 17:56, 15 April 2007.

# Omni-bot 0.52

From Omni-bot Wiki

## Omni-Bot 0.52 STABLE

- Fixed crash with certain maps(baserace, et\_ice)
- Added small offset to health/ammo/armor pickup offset

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52)"

This page has been accessed 160 times. This page was last modified 17:57, 15 April 2007.

# Omni-bot 0.52 beta 6

From Omni-bot Wiki

## Omni-Bot 0.52 Beta 6

- Fixed MESSAGE\_DEATH and MESSAGE\_KILLEDSOMEONE source parameter.
- Fixed waypoint\_addflag functions to optionally take multiple flags
- Fixed GetCursorHint return value
- Fixed waypoint\_autobuild parameters(disconnect param)
- Fixed unable to default construct a script type
- Added bot function ResetSubGoals to script
- Added command waypoint\_mirror, which copies all waypoints and rotates around an axis. Useful for symmetric maps.
- Added GetBotVersion which returns the numeric version of the bot, for optional versioning scripts.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52\\_beta\\_6](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52_beta_6)"

This page has been accessed 121 times. This page was last modified 17:57, 15 April 2007.

# Omni-bot 0.52 beta 5

From Omni-bot Wiki

## Omni-Bot 0.52 Beta 5

- Fixed about 4 crash bugs
- Fixed vehicles not being registered if health == 0
- Added bot.HoldButton() script command, allowing an easy way to make a bot hold a button down for a time period.
- Added a difficulty script, which registers a command /bot difficulty ?, where ? is the difficulty number or name. Additional difficulties can be added, changed in the script.
- Added script library giving basic File IO.
- Added goal registration for dropped items, so dropped flags will be registered as goals under the name allies\_flag, or axis\_flag, and giving them the ability to go for dropped flags.
- Added mini-dump support to windows version. Should create a .dmp that can be sent to me for debugging crashes.
- Added optional filename parameter to /bot show\_goals command, which will dump all the names to a given file, eg /bot show\_goals goals.txt
- Vehicle entities are registered as goals, allowing scripts to implement escort goals and such.
- Changed PressButton function to take multiple buttons as seperate parameters, instead of the user having to | them together.
- Changed parameters of /bot commands so they are passed to script as appropriate types. (previously were all strings)

This release focused on stability for the most part. It should be much more stable than previous versions, including the last stable version. There were several bugs in the scripting system which the author helped to track down and fix.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52\\_beta\\_5](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52_beta_5)"

This page has been accessed 123 times. This page was last modified 17:58, 15 April 2007.

# Omni-bot 0.52 beta 4

From Omni-bot Wiki

## Omni-Bot 0.52 Beta 4

- Optimized some stuff
- Major refactoring of internal code to simplify further development.
- Added /bot dontshoot 1/0
- Fixed setting goal properties.
- Fixed naming of axis goals in some maps. This may break existing scripts.
- Changed first person spectator debug output to be disabled by default. Can re-enable with debugbot command
- Changed syntax for debugbot command
- changed waypoint\_clearallflags to take flag names to clear specific flags from all waypoints.

Not a whole lot of things that are new with this beta, since most of it was internal optimizations and refactoring, and a few fixes.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52\\_beta\\_4](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52_beta_4)"

This page has been accessed 115 times. This page was last modified 17:58, 15 April 2007.

# Omni-bot 0.52 beta 3

From Omni-bot Wiki

## Omni-Bot 0.52 beta 3

Saturday Jan 14, 2006

- Added omnibot\_enable cvar. Defaults to 1
- Added radius check for panzerfaust and flamethrower weapon for friendlies
- Added bot script functions HasLineOfSightTo & InFieldOfView
- Added drawthreats command to highlight detected threats in red
- Added line of sight test to artillery target positions
- Added script command GetCursorHint
- Added script command ChangeSpawnPoint
- Added PERCEPT\_HEAR\_CHATMSG event for chat messages
- Added script function GetEntVelocity
- Added example script that allows bots to get in mountable vehicles
- Added a number of Math functions for scripting: RandFloat, RandInt, ASin, ACos, ATan, ToInt, Abs, Sqrt, Min, Max, Floor, Ceil, Round
- Fixed bridge blockable detection, was swapped
- Fixed removing single connections
- Fixed bots defusing
- Fixed an aiming bug that caused bots forward to get set to ZERO
- Fixed output of script\_stats

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52\\_beta\\_3](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52_beta_3)"

This page has been accessed 108 times. This page was last modified 17:59, 15 April 2007.



# Omni-bot 0.52 beta 2

From Omni-bot Wiki

## Omni-Bot 0.52 beta 2

Thursday Jan 5, 2006

- Added a goal highlighting feature to help scripters. /bot draw\_goals [1/0]
- Added some intelligence that limits the # of bots that can take on a specific goal. Scriptable too
- Added randomization of goals that have the same bias, eliminating order dependency.
- Added cvar g\_omnibotpath that allows users to specify the folder to look for the bot dll.
- Added entity flag LIMBO and MOUNTABLE. Exposed to script.
- Added covert ops satchel usage against appropriate targets(auto detected).
- Added exposed many goal properties to script. See the sample.gm for examples.
- Added version checking in the bot interface to give error messages when using wrong bot dll with game.
- Added all goals now are sorted and distributed by their goal bias. More bias means bots will go after them first.
- Added WEAPON\_FIRE event. Sent any time bots weapon fires. \_param.projectile is a projectile for appropriate weapons.
- Added camp timer in attack/defend goals.
- Added CHAT\_MSG event. Whenever the bot receives a text chat message. \_params.msg is the string that was said.
- Added waypoint command - waypoint\_biconnect and waypoint\_bidisconnect. Same as normal connect/disconnect except it does it both ways in 1 step.
- Added extra parameters to waypoint\_autoradius command for added flexibility.
- Fixed attack goal name.
- Fixed bug where reload checks would change weapons, potentially messing with goals or scripts.
- Fixed blockable collision test to be more reliable(was missing the tank barriers in goldrush).
- Fixed bots always running to a goal that has been completed en-route. Auto success.
- Fixed bug that prevented multiple goal flags to be usable on 1 waypoint.
- \*Fixed bot aiming at close range targets.
- \*\*Fixed bug in Aim calculations - IMPORTANT SEE BELOW
- Changed bot scripts. Eliminated auto creation of weapon tables. Saves a bit of memory.
- Change the bot chat function Say and SayTeam to take any number of parameters and types.

- Previously the bots aim tolerance represented an angular value the bots aim vector would have to be within in order for the bot to be allowed to start shooting. This causes problems in close combat, where it is harder for the bot to maintain an aim vector within that tolerance with horizontally moving targets. For this reason, the AimTolerance property of the bot has been revamped. Now, instead of an angular value of a few degrees(default 3 previously), the value now represents a radius of a sphere that the bot has to be aiming within in order to be within his tolerance. This improves close quarter combat by giving a wider range that more accurately represents the closer targets screen presence. This should improve the bots close quarter target tracking/firing. Due to this change, all previous AimPersistant values within scripts need to be updated or the bots will likely behave oddly and have trouble firing. See the main bot scripts sample.gm and def\_bot.gm for those values.

- \*\*There was a bug in the previous aim calculations that made the bots aiming non time-based. This means anyone running sv\_fps greater than 20(the default) would have had bots that turn much faster than intended. This has been fixed. Unfortunately a side effect of this fix is that it invalidates ALL previous aim values, so scripts need to be updated to the correct values. See the main bot scripts sample.gm and def\_bot.gm for those values.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.52\\_beta\\_2](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.52_beta_2)"

This page has been accessed 120 times. This page was last modified 18:01, 15 April 2007.

# Omni-bot 0.51

From Omni-bot Wiki

## Omni-Bot 0.51 beta

Thursday December 1, 2005

- Updated for etpub 0.6.3
- Updated for Jaymod. Next beta should support this bot version
- Added AimTolerance option for scripting.
- Added EVENT.DEATH and EVENT.KILLEDSOMEONE to script.
- Added EVENT.DEATH example callback to def\_bot.gm.
- Added health\_goal.gm and disguise\_goal.gm example scripts.
- Added satchel\_goal.gm example script. Example usage in oasis.gm
- Added TargetBreakableDist bot property for the distance a bot can target a breakable.
- Added OnBotJoin and OnBotLeave global script callbacks. Example usage in Radar.gm.
- Added bounding box highlighting of game entities when waypoint mode is on.
- Added Utilities.gm, with useful script snippets.
- Added example command to et\_autoexec.gm that implements a 20 second auto save when waypointing.
- Added artillery goal. Implemented with 3 waypoints. See waypoint article
- Added ET specific script function "GetGameState"
- Added ET specific bot script functions, PickNewPrimaryWeapon, GetReinforceTime, IsWeaponCharged
- Added defend goal. Simple implementation currently. Basically a camp location.
- Added bots call artillery on vehicles.
- Added bot property AimAdjustDelayMin and AimAdjustDelayMax. \*
- Added corpse recognition. See disguise\_goal.gm for an example.
- Added bot GetAllType function that gets all targets matching a category and/or class.
- Added numerous additional script functions.
- Added ET specific entity flags DISGUISED, CARRYINGGOAL, MOUNTED
- Added ExecCommand to allow execution of bot commands from script.
- Fixed EVENT.SPAWNED sometimes not being sent on the first spawn after warmup.
- Fixed camp timer for mg42 goals.
- Fixed on-screen goal text when spectating a bot.
- Fixed MaxViewDistance bot property.
- Fixed reduced alot of waypoint mode stutter.
- Fixed removed necessity for wall flag on both ends of a blockable. Should work with 1 now too.
- Fixed goals should get fail signals if the bot is killed.
- Fixed memory leak during waypoint mode.
- Changed bot aim behavior\*

- Changed default memory span to 2 seconds.
- Changed default distance for breakable objects to 0 so bots ignore breakables by default.
- Changed script entity functions to global functions that can take an entity or gamelid.
- Changed door goals so the bot only 'uses' if a raycast fails. Improves hatch usage.
- Changed /bot come functions to take an optional # for the gamelid of the bot. Without specifying, defaults to all bots
- Changed /bot roam the same way
- Changed goto goal to go to the nearest waypoint, not the actual position. Improves several other issues.
- Changed script AimError from Vector3 to Vector2 for clarity. x = horizontal, y = vertical.\*
- Changed removed hard coded voice chat messages(hi, bye, class announce) Script em if you want em.

- \*The bot aiming behavior has been changed for this release, specifically by implementing a clearer influence of aimerror on the bots aiming. Among these changes are the following.

NOTE: I didn't remove the movable waypoint from this release, despite it being not finished. I recommend NOT using it though unless you just want to toy with it.

## Aim Tweaks

- AimError is now a 2d vector, representing x(horizontal), and y(vertical) aim error.
- The AimError represents a random offset from the targets position to aim for.
- This offset is a random value between -AimError.x to +AimError.x horizontally, and -AimError.y to +AimError.y vertically.
- The random offset is re-calculated at a random time between AimAdjustDelayMin and AimAdjustDelayMax, which default to 0.0 and 2.0 seconds.
- AimAdjustDelayMin and AimAdjustDelayMax can be tweaked by the user from script.

This will allow users to more easily tweak the bots aim to a more acceptable level.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.51](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.51)"

This page has been accessed 132 times. This page was last modified 18:03, 15 April 2007.

# Omni-bot 0.4

From Omni-bot Wiki

## Omni-Bot 0.4 beta

Tuesday November 1, 2005

- Added a ton more script functions.
- Added map extents to waypoint header.
- Added version 5 waypoint file format.
- Added waypoint\_addflagx command.
- Added ability to add custom functions through scripts.
- Added auto detected 'flag' goal type for carryable goals such as the radar parts in radar, or gold in goldrush.
- Added cappoint waypoint flag, used as destination point for flag goals.
- Added sample.gm script, heavily documented as a demonstration script.
- Added movable waypoint flag(not implemented yet).
- Fixed bug where bots picked wrong class after warmup expired.
- Fixed long load times due to vis table calculation. Vis tables now written to file to prevent re-calculation.
- Fixed numerous crashes.
- Fixed bug where large maps could still run out of entities.
- Fixed crash when deleting a waypoint if it was marked to be connected.
- Fixed crash bug where a bot could still have evaluators for class.
- Fixed default profile usage.
- Fixed MG42 goal so bots dont fire at unoccupied mg42s.
- Fixed waypoint archive loading.
- Fixed bot sprinting for sprint waypoints.
- Fixed ET class choosing.
- Fixed revive goals if target is in water.
- Fixed goal scripting.
- Changed goal names to use a unique waypoint Id, rather than an ever increasing number. This should ensure constant goal names for scripts.
- Changed bot scripting. Huge improvements.
- Changed names that most goal types use. Should now be more unique(but not always).
- Changed(got rid of) the goal thread related functions in script, in favor of using gm threads.
- Changed internal function \_GetClosestWaypoint to no longer use line of sight tests. Fixes many nav issues.
- Changed breakable targets to be limited to very close(300 units) distance.
- Changed weapon desirabilities versus some target types such as breakables. Should

no longer use panzers/flamers/etc... against breakables.

- Changed bot script callbacks to take table, which has event dependant data.
- Changed A\* path planner. Increased performance by 20%
- Updated waypoint files.
- Updated script files.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.4](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.4)"

This page has been accessed 124 times. This page was last modified 18:04, 15 April 2007.

# Omni-bot 0.31

From Omni-bot Wiki

## Omni-Bot 0.31 beta

Tuesday August 23, 2005

- Updated nav files.
- Improved how bots handle "badly" waypointed maps.
- Reduced voice chat usage for grenades.
- Fixed duplicate bot names.
- Fixed minbots/maxbots.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.31](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.31)"

This page has been accessed 113 times. This page was last modified 18:04, 15 April 2007.

# Omni-bot 0.3

From Omni-bot Wiki

## Omni-Bot 0.3 beta

Wednesday August 17, 2005

- Added mobile MG42 goal for soldiers.
- Added bots can be fooled by disguised CovertOps now.
- Added map scripting(triggers).
- Fixed AI related causes of the "bots do nothing" error.
- Changed parameter order of addbot command.
- Improved bot grenade handling.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.3](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.3)"

This page has been accessed 139 times. This page was last modified 18:05, 15 April 2007.



# Omni-bot 0.231

From Omni-bot Wiki

## Omni-Bot 0.231 beta

Sunday July 31, 2005

- Fixed another invisible waypoint bug.
- Fixed bots shooting at unbuilt MG42s.
- Fixed maps crashing with "G\_Spawn: no free entities" error.
- Tweaked aiming a bit.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.231](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.231)"

This page has been accessed 102 times. This page was last modified 18:05, 15 April 2007.

# Omni-bot 0.23

From Omni-bot Wiki

## Omni-Bot 0.23 beta

Saturday July 23, 2005

- Added ShowFunctions() script command
- Added inwater, underwater waypoint flags.
- Added buttons drop, leanleft, leanright, aim
- Added inwater, underwater entity flag
- Added contents enumeration
- Added visual feedback to status of blockable paths in waypoint mode
- Added waypoint\_move command to move existing waypoints, preserving connections
- Added waterblockable waypoint flag
- Added waypoint\_clearflags to clear all flags from current Waypoint
- Added blockable list rebuild on waypoint\_save
- Added waypoint\_add automatically detect if its in water.
- Added bots respond to medic and ammo requests
- Added bots throw smoke markers(air strikes) at vehicles
- Added mount mg42 map goal, auto detected
- Added Prone waypoint tag and bot ability
- Fixed ET sniper goal choose random instead of closest Goal
- Fixed bots allowed to shoot while waiting at cabinets
- Fixed bots looking at ground while at health/ammo cabinets
- Changed ET sniper goal obey crouch/prone flags on target Waypoint

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.23](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.23)"

This page has been accessed 132 times. This page was last modified 18:06, 15 April 2007.

# Omni-bot 0.22

From Omni-bot Wiki

## Omni-Bot 0.22 beta

Monday June 27, 2005

- Fixed engineer saying "I'm a medic"
- Linux version now runs on glibc-2.2

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.22](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.22)"

This page has been accessed 123 times. This page was last modified 18:06, 15 April 2007.

# Omni-bot 0.21

From Omni-bot Wiki

## Omni-Bot 0.21 beta

Friday June 24, 2005

- Greatly reduced voice macro spam of "I'm taking fire".
- Fixed jumplow and jumpgap flag in ET due to improper implementation of getting player bounding box.
- Improved jumplow handling.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.21](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.21)"

This page has been accessed 124 times. This page was last modified 18:07, 15 April 2007.

# Omni-bot 0.2

From Omni-bot Wiki

## Omni-Bot 0.2

First release, more or less representing the ETF implementation of Omni-bot.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.2](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.2)"

This page has been accessed 154 times. This page was last modified 18:41, 12 October 2007.

# Category:ET

From Omni-bot Wiki

## Subcategories

There are 4 subcategories to this category.

### C

- Community Scripts
- Constants

### E

- ET Constants
- ET ScriptGoals

## Pages in category "ET"

There are 36 pages in this category.

### C

- Omni-bot Command Reference
- Omni-bot Wiki: Community Portal
- Configuring Omni-bot

### E

- ET:goal airstrike
- ET:goal askforammo
- ET:goal askforhealth
- ET:goal combatmovement
- ET:goal deliversupplies
- ET:goal dispenseammo
- ET:goal dispensehealth
- ET:goal escortvehicle
- ET:goal grenadetarget

### E cont.

- ET:goal mountvehicle
- ET:goal ridevehicle
- ET:goal supplyself
- ET:goal useswitch

### F

- Omni-bot F.A.Q.

### I

- Installing Omni-bot
- Intro to Omni-bot

### M

- Omni-bot Map

### S cont.

- ScriptGoal
- Enemy Territory Script Goals
- Omni-bot Script Goals
- Omni-bot Scripting
- Sniper Spots

### V

- Vehicle Pathing

### W

- Omni-bot Waypointing
- ET Waypoints

- Scripting
- Map Scripting Enemy Territory
- Map template

- ET Waypointing Tutorial
- Waypointing Trick
- Omni-bot Weapon Scripting

## **R**

- Omni-bot Releases
- Omni-bot Routing

## **S**

- Omni-bot Script Reference

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Category:ET>"

This page has been accessed 733 times. This page was last modified 19:12, 27 May 2007.

# Category:Quake 4

From Omni-bot Wiki

UNDER CONSTRUCTION

## Subcategories

There is one subcategory to this category.

### C

- Constants

## Pages in category "Quake 4"

There are 15 pages in this category.

### C

- Omni-bot Command Reference
- Omni-bot Wiki: Community Portal
- Configuring Omni-bot

### F

- Omni-bot F.A.Q.

### I

- Installing Omni-bot

### I cont.

- Intro to Omni-bot

### M

- Omni-bot Map Scripting

### R

- Omni-bot Releases
- Omni-bot Routing

### S

- Omni-bot Script Reference
- ScriptGoal

### S cont.

- Omni-bot Script Goals
- Omni-bot Scripting

### W

- Omni-bot Waypointing
- Omni-bot Weapon Scripting



# Category:Doom 3

From Omni-bot Wiki

UNDER CONSTRUCTION

## Subcategories

There is one subcategory to this category.

### C

- Constants

## Pages in category "Doom 3"

There are 15 pages in this category.

### C

- Omni-bot Command Reference
- Omni-bot Wiki: Community Portal
- Configuring Omni-bot

### F

- Omni-bot F.A.Q.

### I

- Installing Omni-bot

### I cont.

- Intro to Omni-bot

### M

- Omni-bot Map Scripting

### R

- Omni-bot Releases
- Omni-bot Routing

### S

- Omni-bot Script Reference
- ScriptGoal

### S cont.

- Omni-bot Script Goals
- Omni-bot Scripting

### W

- Omni-bot Waypointing
- Omni-bot Weapon Scripting

# Permission error

From Omni-bot Wiki

You do not have permission to do that, for the following reasons:

- The action you have requested is limited to users in the group user.
- Omni-bot Wiki has restricted the ability to create new pages. You can go back and edit an existing page, or log in or create an account.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Omni-bot\\_0.66\\_Current\\_Known\\_Bugs](http://omni-bot.com/wiki/index.php?title=Omni-bot_0.66_Current_Known_Bugs)"

# ET Waypointing Tutorial

From Omni-bot Wiki

<a href="#">ET Main Page</a>	<b>Waypoint Tutorial</b>	<a href="#">Archives</a>
------------------------------	--------------------------	--------------------------

This is an introduction to making waypoints for Omni-bot. Most examples used here come from Detoeni's Alpine Assault map for the game RTCW: Enemy Territory. However, it should be relatively easy to transfer the information given here to other maps and games.

Before starting to make your own waypoints, you should familiarize yourself with the topic by studying some of the waypoints for the 'official' maps that come bundled with Omni-bot. In order to do this, load the map in question, bring down the in-game console and type `/bot waypoint_view on`. You will see waypoints and connections between them, much like the ones you are about to create.

Also, an important prerequisite for making good waypoints is that you have played with bots before for some time, because you should have a general idea of what bots can and can't do before starting to waypoint.

<b>Contents</b>
<ul style="list-style-type: none"><li>• 1 Understanding the "story" of the Map</li><li>• 2 Waypointing: The First Steps</li><li>• 3 Streamlining Bot Navigation</li><li>• 4 Using Waypoint Flags<ul style="list-style-type: none"><li>◦ 4.1 Navigational Flags</li><li>◦ 4.2 Tactical Flags</li></ul></li><li>• 5 Map Scripts</li></ul>

## Understanding the "story" of the Map

Before you start waypointing a map, it's important that you have a good understanding of the map's game flow and know your way around the map. You may still discover some things when in waypoint mode, but generally, you should know what's going on in the map before you start. In our Alpine Assault example, a rough outline of the map's "storyline" seen from Allied perspective would be:

Starting from your first spawn point, capture and defend the second spawn in the garage:



Gain access to the mine complex, where the Axis is defending the safe key in the Armory:



Grab the key from the desk in the Armory, and try to reach the safe with the documents ...



... which will of course be guarded by the enemy. Steal the documents from the safe ...



... and escape to the safety of your first spawn point.



*All stills were taken on location with real bots. No bots were harmed in the making of these screenshots.*

With such a basic outline in mind of what a typical game flow will be like, you'll be able to build efficient paths for the bots, and above all, make a good map script.

## Waypointing: The First Steps

**1. Start a listen server.** Start a server with a client on the same machine (a.k.a. "listen server", as opposed to a dedicated server). Normally, double-clicking the shortcut that Omni-bot generates on your desktop during the installation process will do just that.

If you don't know how to start, visit the Omni-bot F.A.Q.

**2. Load the map you want to waypoint**, either by selecting "Host game" from the game menu and selecting the map from the list, or by typing

`/map <mapname>`

into the game console. Instead of `/map`, you could consider using `/devmap`, which enables some "cheats" (mainly intended for use by map developers), such as walking through walls, warping to a given position, etc.

**3. Disable the time limit of the map** or set it to something very high if possible. In ET, use the command `/ref timelimit 0` to have unlimited time on a map. If a map restarts while you are editing waypoints, your unsaved changes will be lost.

**4. Join a team.** On most maps, joining the attacking team as an engineer is the best thing to do, because then you can dynamite obstacles out of your own way, build bridges etc. If you used `/devmap` to start the map, try issuing the `/noclip` command in the game console and experiment a little with what you can do. Use `/noclip off` to turn it off. It's better not to add waypoints in noclip mode, but it might help to reach some spot very quickly that would be otherwise very difficult and cumbersome to reach.

**5. Enable waypoint mode and start adding waypoints.** Waypoint mode is turned on by using the `/bot waypoint_view` on command. Add waypoints by issuing `/bot waypoint_add` in the console, while running around the map as a normal player would. A single waypoint will look like this:



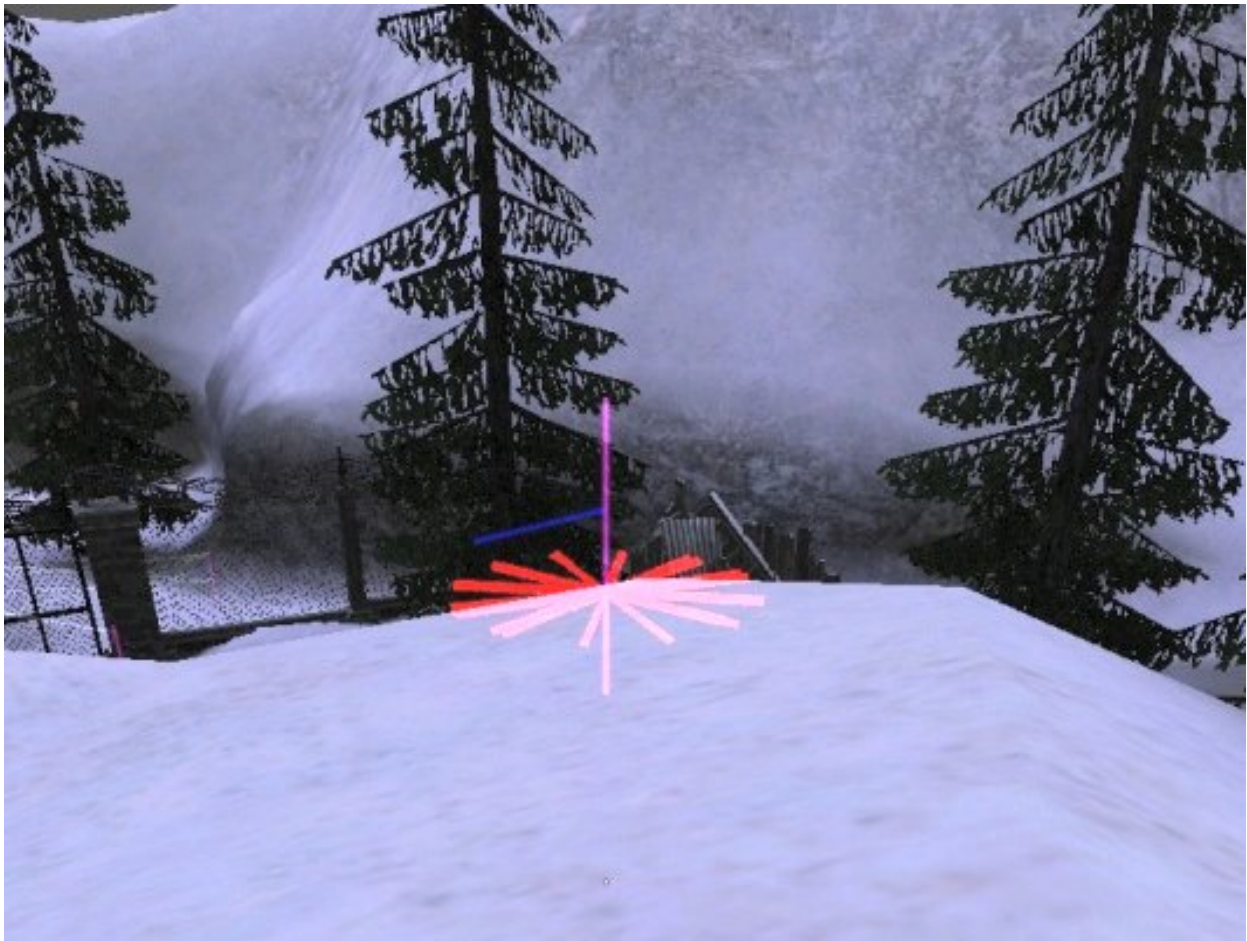


**6. Basic waypoint properties.** When you hover your crosshairs over the waypoint, some information about it will be echoed to the screen:

- the waypoint's number,
- the waypoint's radius,
- the waypoint's unique ID (this will always stay the same, even if you delete some waypoints later on, while the number may change).

While the number and UID are relatively unimportant at the moment, the radius is worth mentioning. The radius is effectively the tolerance for moving toward a waypoint. If you need the bot to get closer, use a smaller radius, if the waypoint represents a large room, use a large radius. You can change a waypoint's radius with the command `/bot waypoint_setradius <value>`. However, leaving it at the default radius of 35 game units will be okay in most cases. We will get back to radius later on.

**7. Advanced waypoint properties.** Additionally to a radius, UID and number, all waypoints added with Omni-bot 0.6 and higher will have a *facing*, which is basically the direction in which you looked when you added the waypoint. Use the command `/bot waypoint_viewfacing on` to see the facing, indicated by a blue line:



A waypoint's facing is a vector associated with the waypoint that can be used to tell the bot in which direction to aim in some situations, e.g. when sniping. For merely navigational purposes, however, the facing is irrelevant in most cases. We will cover facing later on.

Finally, a waypoint can have one or more *flags* that can serve different purposes, for example tell a bot to jump, prone, crouch or walk. We will go into the details of flags later in chapter 5, just keep in mind that there are such things for now.

**8. Connect the waypoints.** By themselves, waypoints are pretty useless. They have to be *connected* to each other in order to enable the bots to move. The waypoints together with their connections can be considered a (directed) graph that gives a rough representation of the map's geometry. It's up to you if you first add a large number of waypoints and connect them later on, optionally using the `/bot waypoint_autobuild` command, or if you want to connect every newly added waypoint to one or more other waypoints immediately after adding it.

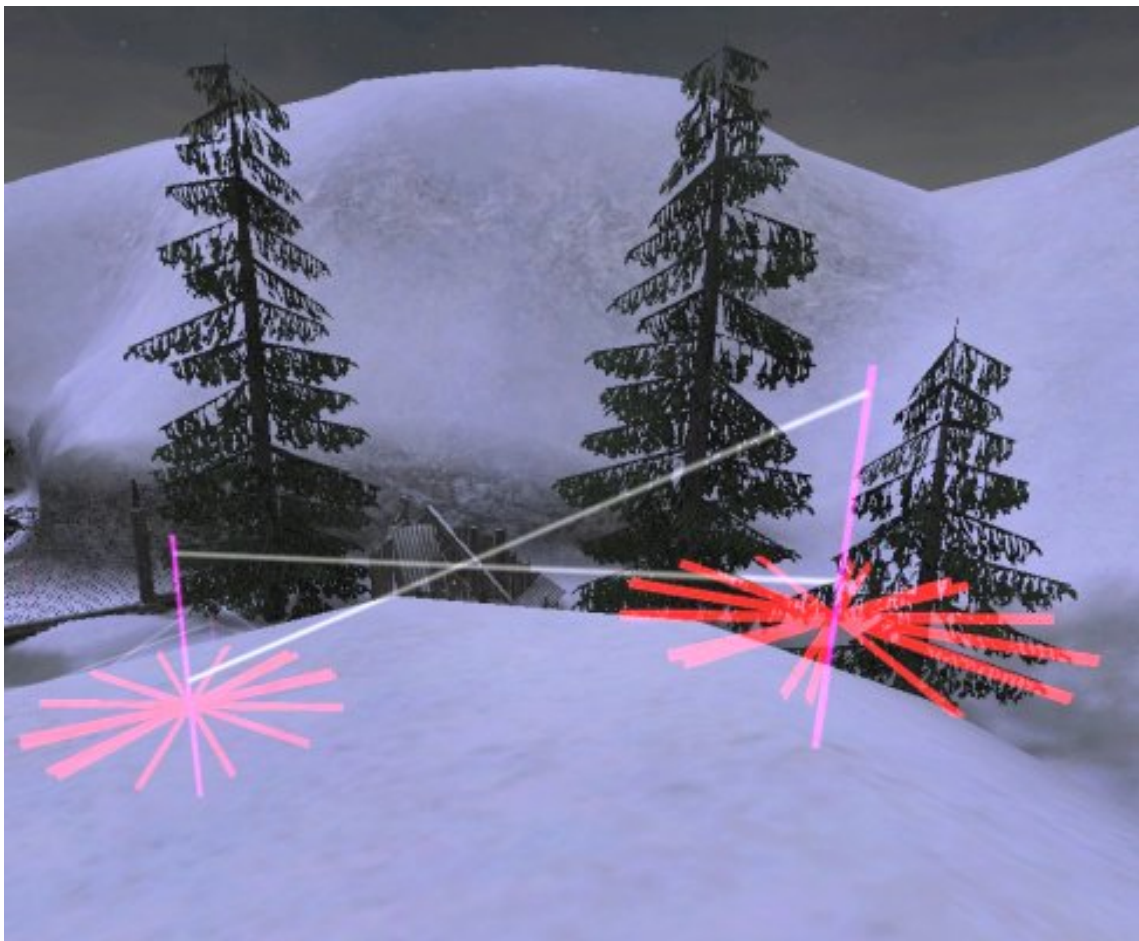
A one-way connection between two waypoints is added by issuing the `/bot waypoint_connect` command in the console twice, while standing over both waypoints. The connection will look like this:





This screenshot shows two waypoints connected by a one-way path. We know the connection here is from the right waypoint to the left because of the angle of the link. Link direction is shown by a downward sloping link.

In the following screenshot we see what a two-way connection looks like, which is added by using the `/bot waypoint_biconnect` command (or by using `/bot waypoint_connect` twice on each waypoint). Since there is a connection going both ways, the resulting link appears as an 'X' between the waypoints:



It is strongly recommended to use bidirectional connections in most cases; one-way connections should be constrained to jumping down and other special cases. The distance between the two waypoints shown above is a good one to use in many cases. As a general rule, distances much greater than, say, 200 game units should be avoided in most cases, because it allows for much smoother bot navigation if the paths between waypoints are relatively short.

**9. Save your waypoints** using `/bot waypoint_save`. This saves the waypoints for the currently loaded map to a file called `<mapname>.way` in the `nav` directory under the game currently running.

*Warning:* This will overwrite an existing waypoint file of the same name without asking. Rename or backup your original waypoint files when you want to experiment.

Generally, it is a good habit to save often during the process of waypointing a map. But since there is no undo command at the moment, do not save changes that are extremely experimental without taking a moment to think before. The only way to undo mistakes is to load the previously saved version.

Now you should be able to add one or more bots. If you have added waypoints close to the spawn areas, you will see them move around and do more or less useful things, and most likely get stuck occasionally. This is why we have to fine-tune our rough-and-ready waypoints a bit.

---

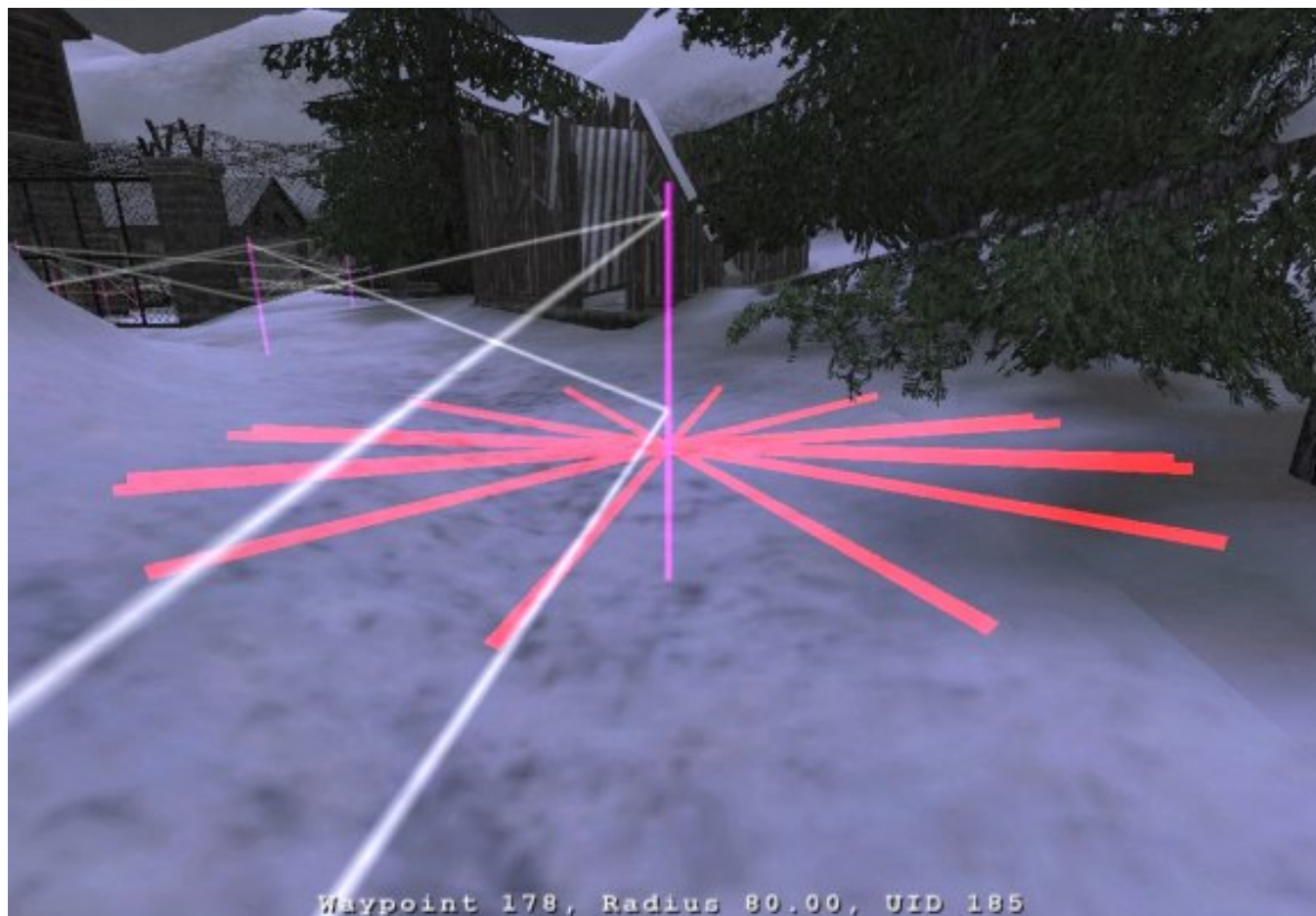
**TIP:** Bind your most frequently used commands to keys. In ET for example, a bot command is bound to a key using the following command in the game console: `/bind <key> "bot <command>"`. For example: `/bind f5 "bot waypoint_setradius 25"`, `/bind f6 "bot waypoint_addflag blockwall;bot waypoint_setradius 30"`  
You can easily save lots of these bind commands to a plain text file, say `waypointing.cfg`, and later execute this file by issuing the following in the console: `/exec waypointing.cfg` (Save the file in your `etmain` folder.)

---

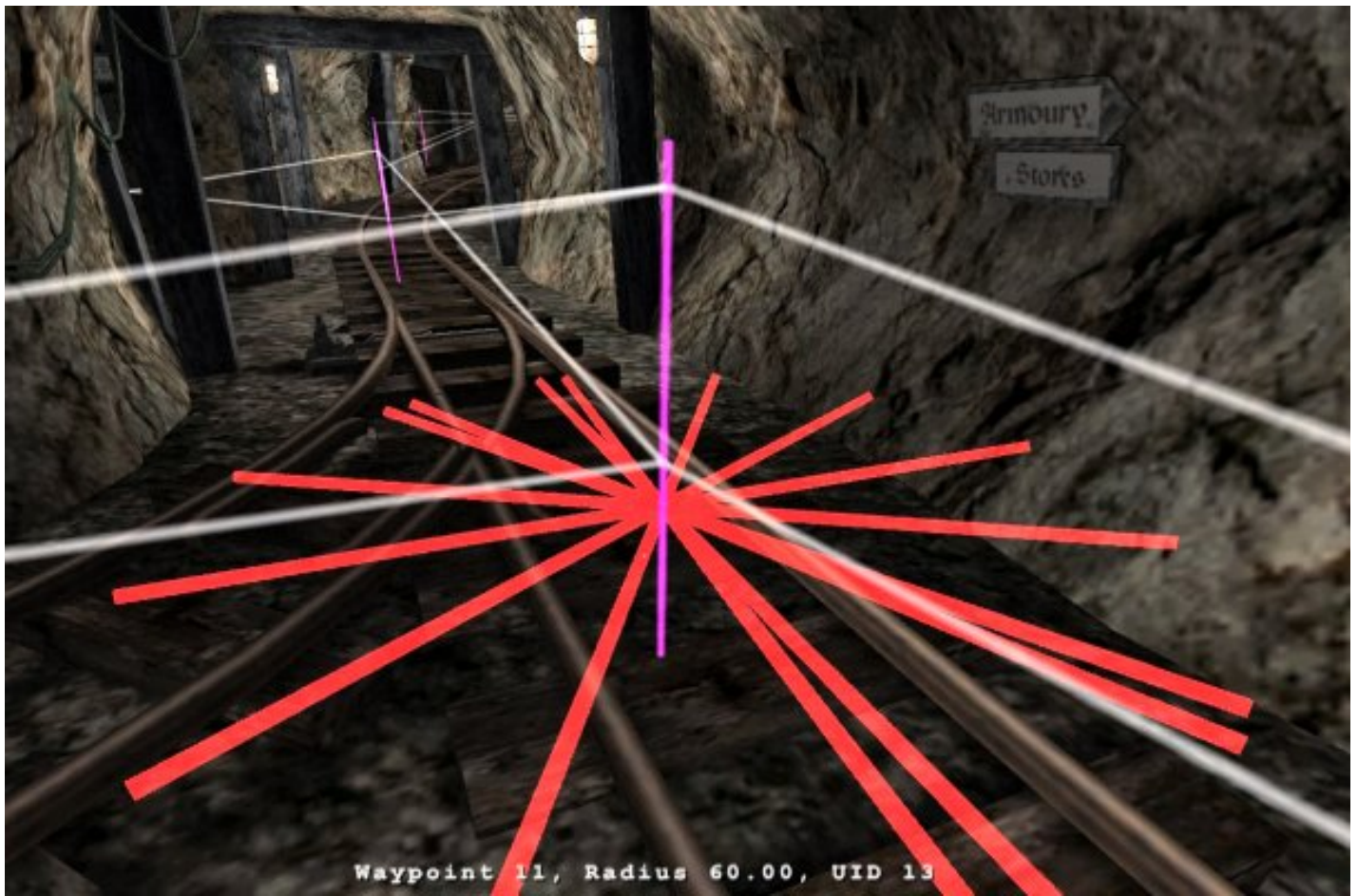
## Streamlining Bot Navigation

**1. Radius in more detail.** As mentioned before, a waypoint's radius is the tolerance for moving towards a waypoint. If you want the bots to get very close to a certain spot, use a smaller radius, if it doesn't matter much if a bot reaches this specific waypoint exactly, use a large radius. You can change a waypoint's radius with the command `/bot waypoint_setradius <value>`. Don't use radius values of less than 10 or so.

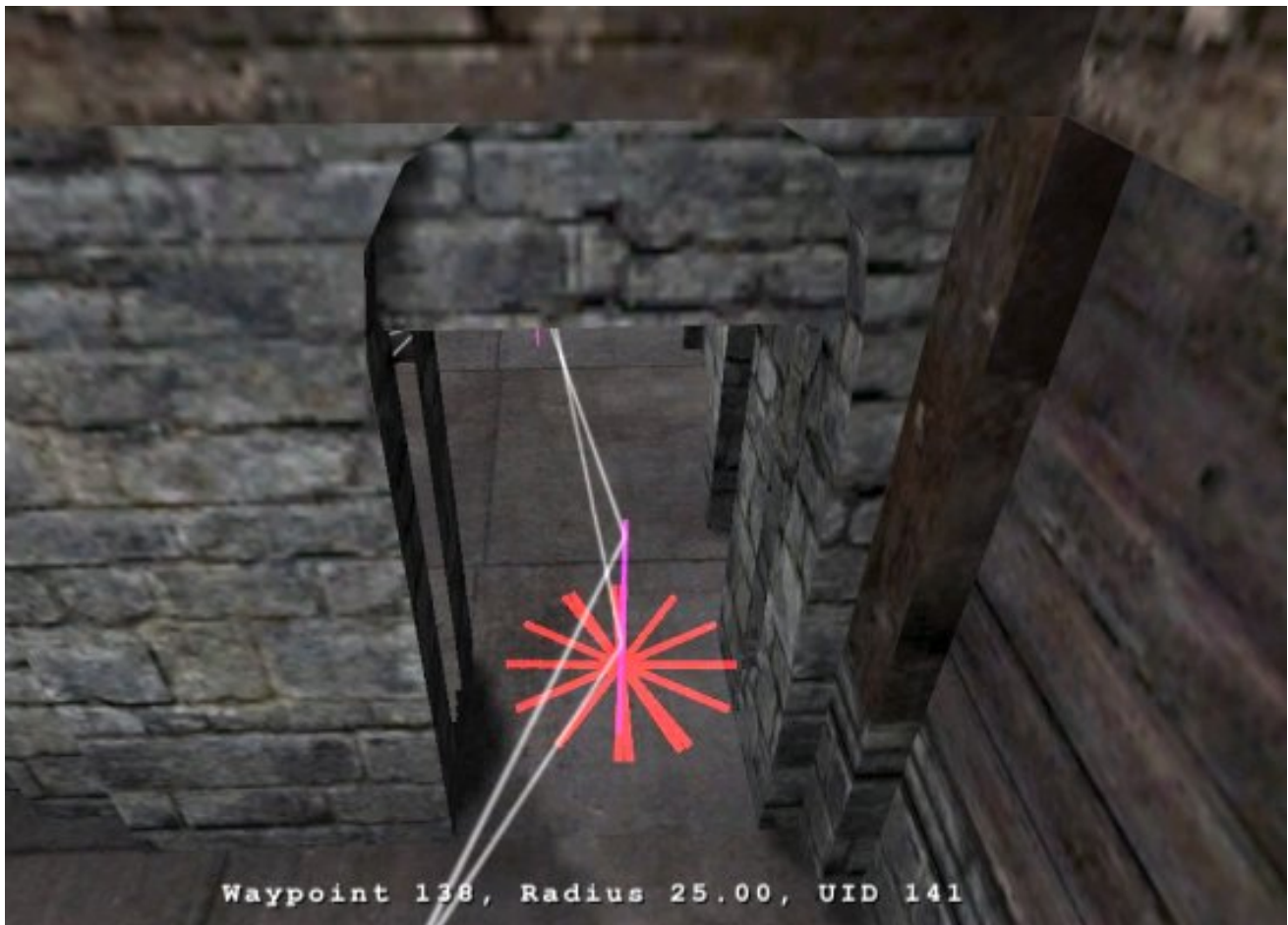
While the default radius of 35 game units will do in most cases, some situations require some radius tweaking. In areas where there is plenty of space, a larger radius will make it easier for the bots to avoid running into each other when they meet on a waypoint, so a radius of 60, 80, or even 100 will be a better choice:



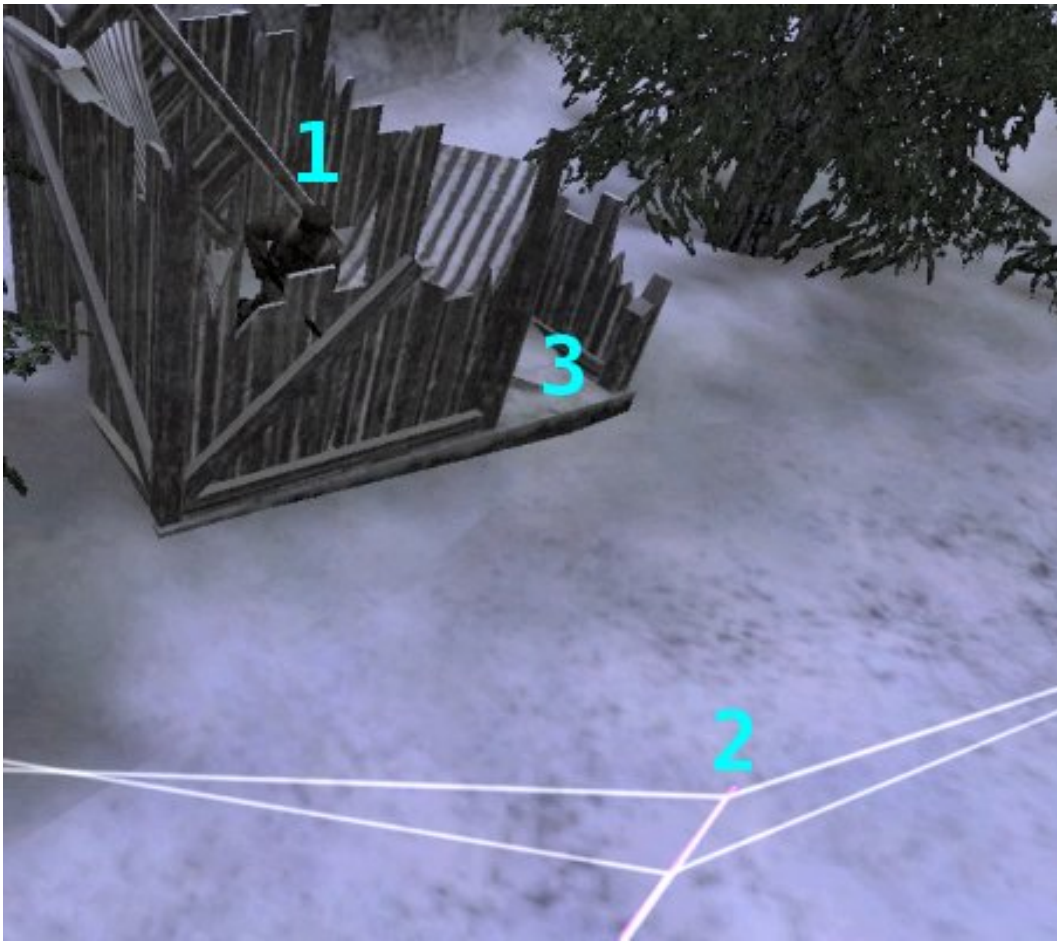




In tight areas, in particular at doors and ladders, use a smaller radius, but normally not much smaller than 20. Make sure that the red "beams" that visualize the radius do not hit walls or other objects:



**2. Avoiding stuckage.** When you try to add bots to a map you are waypointing, they will almost inevitably get stuck at first. A typical example of what has come to be called 'stuckage' can be seen in the following screenshot:



The bot (1) is trying to move directly towards the closest waypoint (2), unaware of the path 1 - 3 - 2 a human player would naturally choose. Even though there is no path the bot could possibly have followed to reach its present location (1), this type of problem is very common, because bots often get pushed off their intended paths by other players or due to the impact of weapons, especially grenades and such.

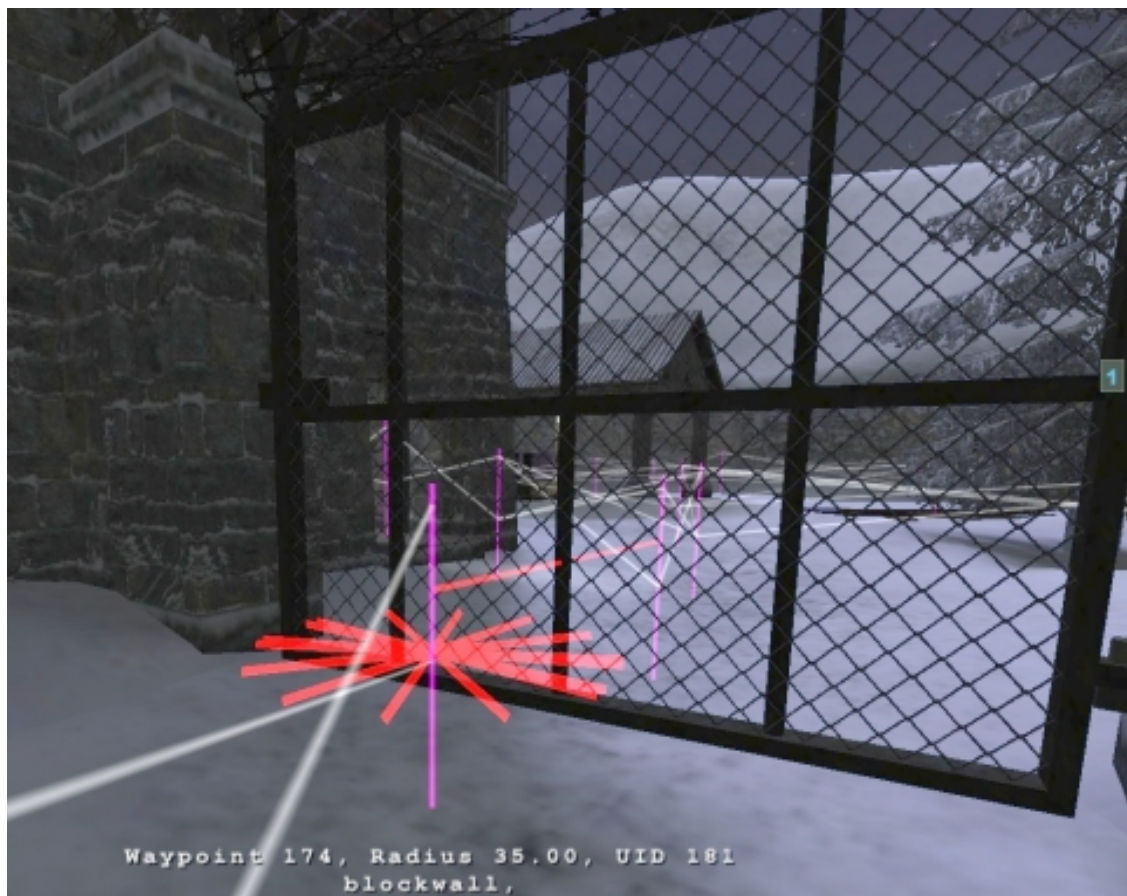
The next image shows a solution:





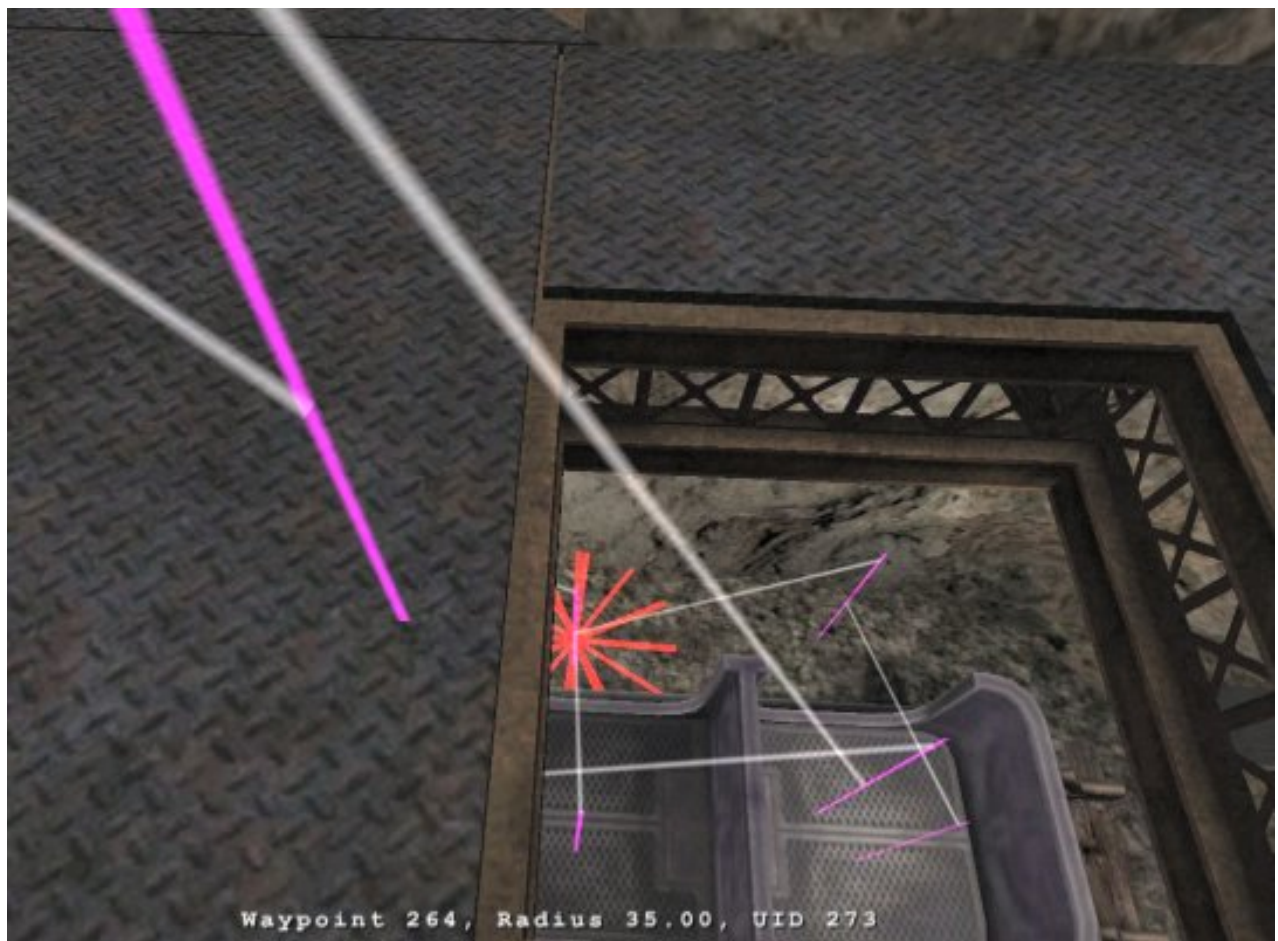
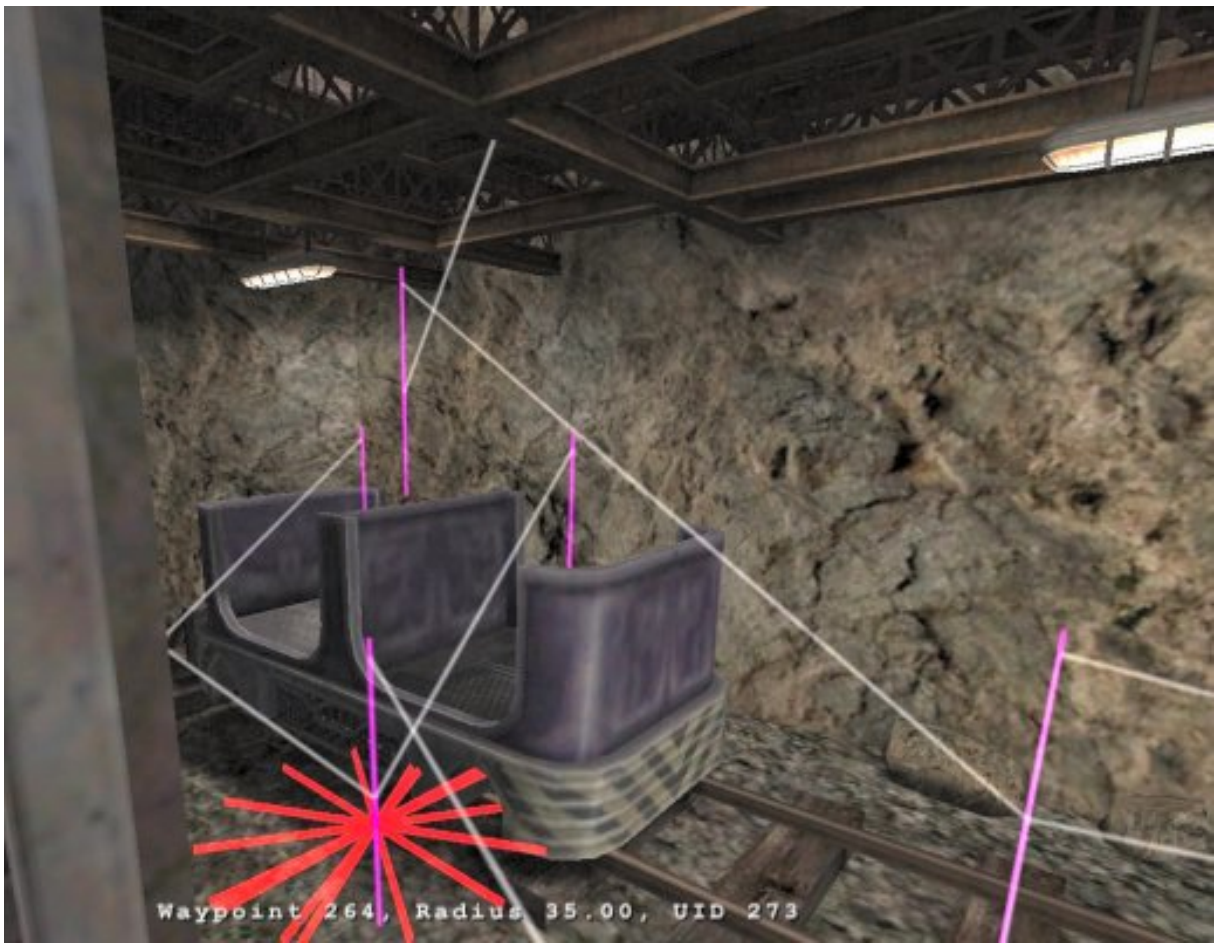
In the above example, adding a waypoint at position 3 is sufficient to "free" the bot that got stuck at 1, because 3 rather than 2 will be the closest waypoint once it is added.

**3. Make sure the bots can reach the objectives.** A rough sketch of the bot logic is as follows: First the bot picks a goal, which can be just going somewhere, or something to achieve, like constructing or destroying some object. Then the bot computes a path to this goal along the waypoints given by the waypointer. Therefore, it is important that there are waypoints close to the objectives you want the bots to accomplish. The following screenshot shows a dynamite/satchel target with waypoints nearby:



**4. Make moderate use of one-way paths.** Most of the time, bidirectional connections between waypoints should be preferred. Just think of medics to see why: dead or injured players can end up virtually anywhere on a map, thus the more spots on the map bots can reach, the better. In some situations, however, one-way connections are in order, in particular for jump-down paths. The following screenshots show an example of legitimate use of one-way connections from two different perspectives:





By the way, when you want to connect waypoints in a situation similar to the one shown above, the `/bot waypoint_connectx` command might be helpful. It is almost the same as `waypoint_connect`, but applies to the waypoint you are aiming at. Thus you don't have to actually jump down in order to connect the upper waypoint to the lower one.

## Using Waypoint Flags

Waypoint flags can serve a large number of purposes. A flag is added to a waypoint by standing over the waypoint and writing `/bot waypoint_addflag <flagname>` into the console. In order to remove a flag, use the same command. In order to remove all flags of a given kind, use `/waypoint_clearallflags <flagname>`. `<flagname>` can be a list of flags, both when adding and deleting flags. When you hover your crosshairs over a waypoint, the flags it carries will be echoed to the screen along with other information about it.

The following is not intended as a complete guide to all the waypoint flags, we will cover only the most basic flags and their usage here. See the Wiki flag list.

### Navigational Flags

**Movement flags.** The names of the crouch, prone, sprint, climb and sneak flags speak pretty much for themselves: They affect the way a bot will move towards a waypoint with these flags. Used in conjunction with a snipe flag on the same waypoint, however, crouch and prone will be interpreted as a stance to be in, i.e. the bot that snipes there will crouch or prone all the time.

The difference between the jump, jumpgap and jumplow flags is as follows:

- jumpgap will make the bot check the ground ahead for gaps to jump over,
- jumplow will make the bot check for obstacles such as small walls, crates, boxes, and jump upon or over them,
- jump will make the bots jump without any of those checks.

jump is useful if the detection methods mentioned above fail. The other jump flags should normally be preferred.

**The door flag.** Doors are typically waypointed depending on how they operate. In many games, doors are automatic and simply open when the player approaches them. It isn't necessary to do anything special in the case of those doors. In some games, doors have to be "used", such as in Enemy Territory. In these cases, the door waypoint flag is useful. The door flag is used by adding the flag to the two waypoints on either side of a door. This tells the bot to hit the USE key while moving across the link between the two waypoints. In the following picture we see both waypoints flagged with door, and in this case they are team specific:



**Team flags.** The purpose of team flags is to make a waypoint off limits for any team whose team flag it hasn't. The teamonly flag is added automatically once you add any of the team1-team4 flags to a waypoint. For example, a waypoint can have a team1 and a team3 flag, which will make it unavailable for Team 2 and Team 4. In Enemy Territory, it obviously doesn't make sense to add both team flags (there are only two), because having both is as good as having none at all, it will just confuse any waypointer who might try to update your waypoints in the future. Also in Enemy Territory it's more convenient to use the alias axis for team1, allies for team2. The teamonly flag will be added automatically and should not be used directly.

Keep in mind that team flags will mark the waypoint as a no-go area for the other teams, so use with care. Team flags are most useful in conjunction with door flags ("team doors"), and perhaps the snipe, defend, cappoint and attack flag. They can also be used at spawn point entries (thinking of huts and the like), in order to keep bots out of the other team's spawn areas. (This is considered good style by many players.)

**The cappoint flag** is used only on the "capture and deliver" style of maps. On those maps, however, it is crucial. The bot will auto-detect the items to be stolen (captured), but not the point on the map where these items should be delivered. Use /bot waypoint\_addflag cappoint to mark a place where the bots should deliver the objectives. It will create a goal for bots carrying objectives that should by default override most other goals.

**Blockable paths.** Blockable paths deserve special mentioning. They are a feature of Omni-bot that allow paths to update their own blocked status, saving some scripting effort. Similar to door, the block flags



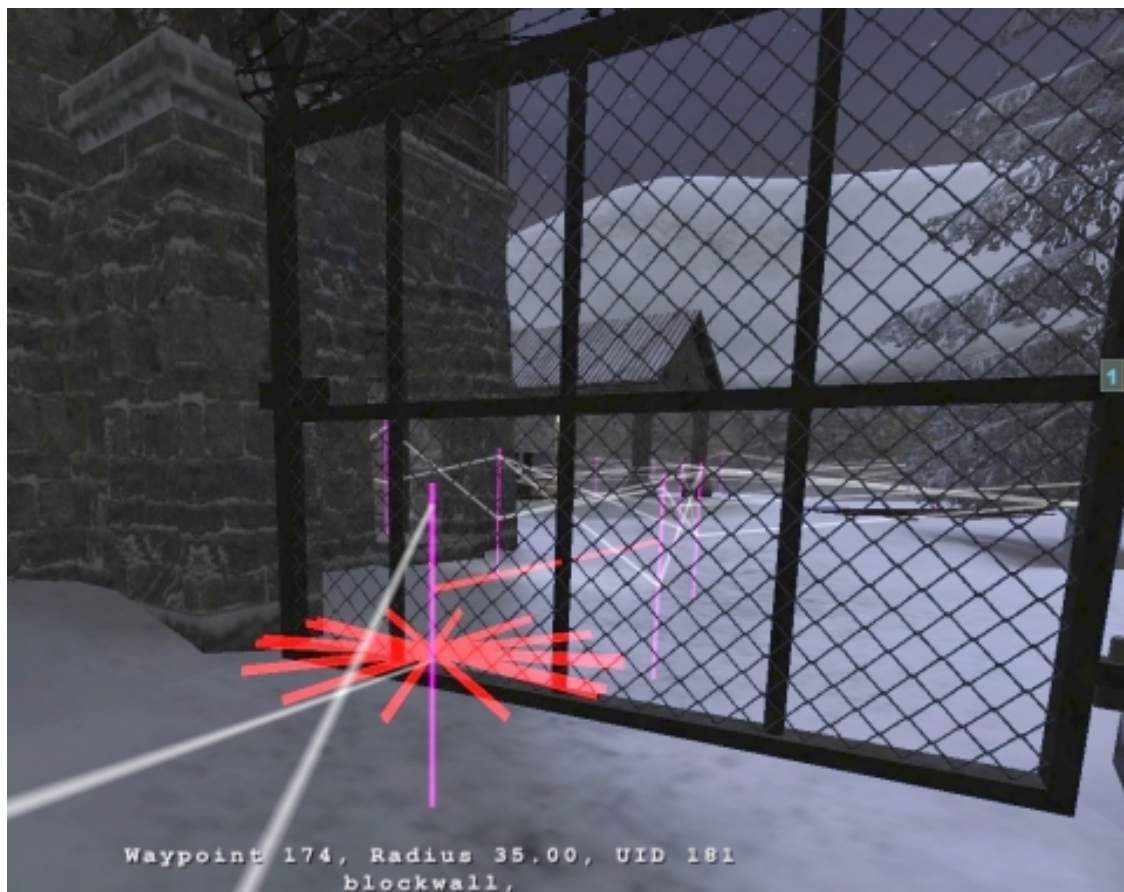
need to be placed on both sides of the object between them (a wall or a bridge, typically).

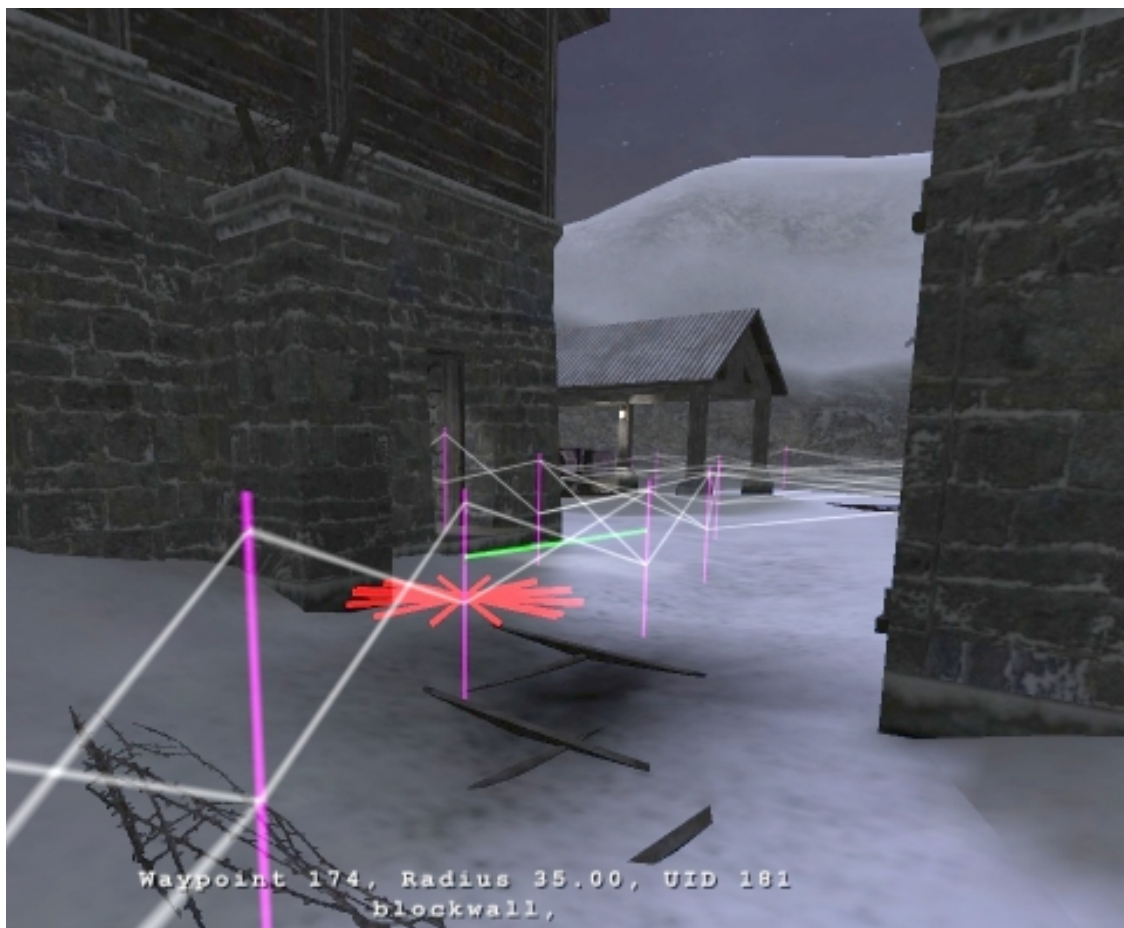
There are three types of blockable paths:

- blockwall - If a line of sight check passes between the 2 waypoints, the path is opened, otherwise it is closed.
- blockbridge - If an object isn't detected half-way between the 2 waypoints (such as a bridge, a barrier or the like), the path is blocked, otherwise it is open.
- blockwater - If the path between the waypoints is under water, the path is blocked, otherwise it is open (examples: the tunnels in Oasis).

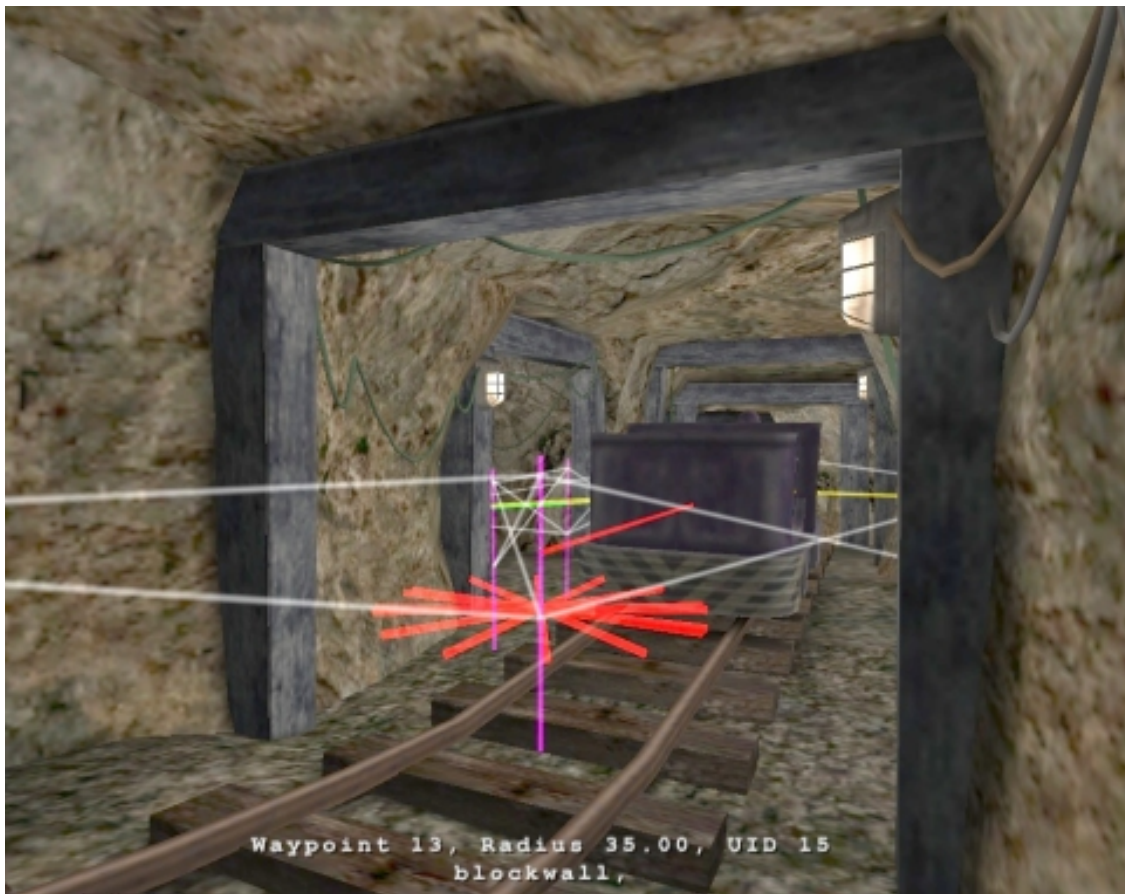
When in waypointing mode, a red line marks a path that is detected as blocked, while a green line indicates a free path.

An example of blockwall:





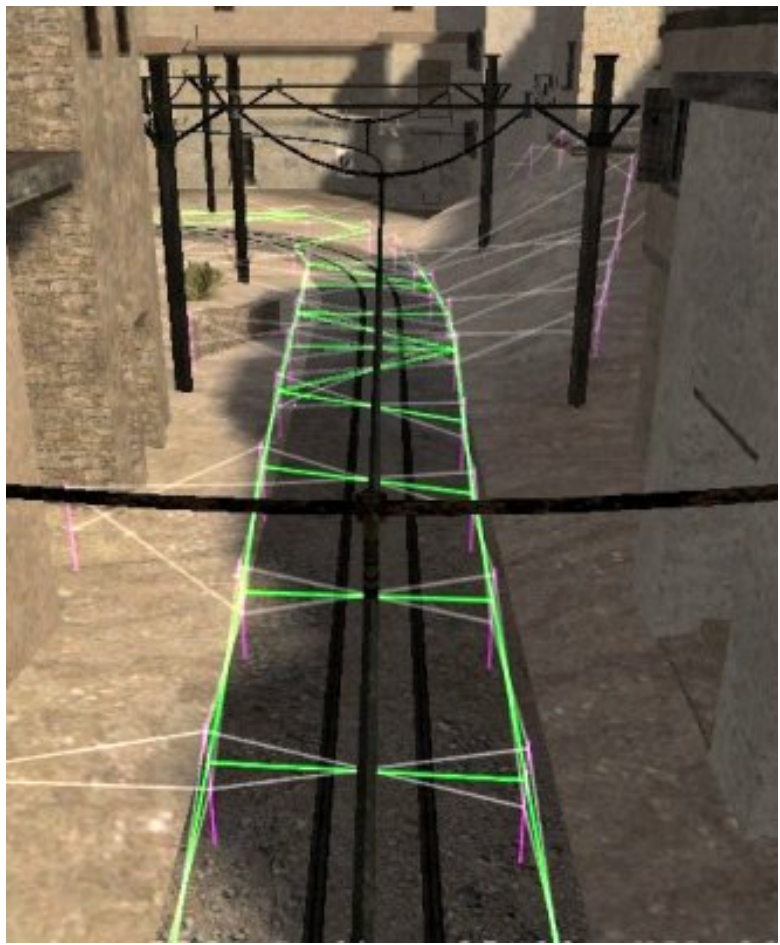
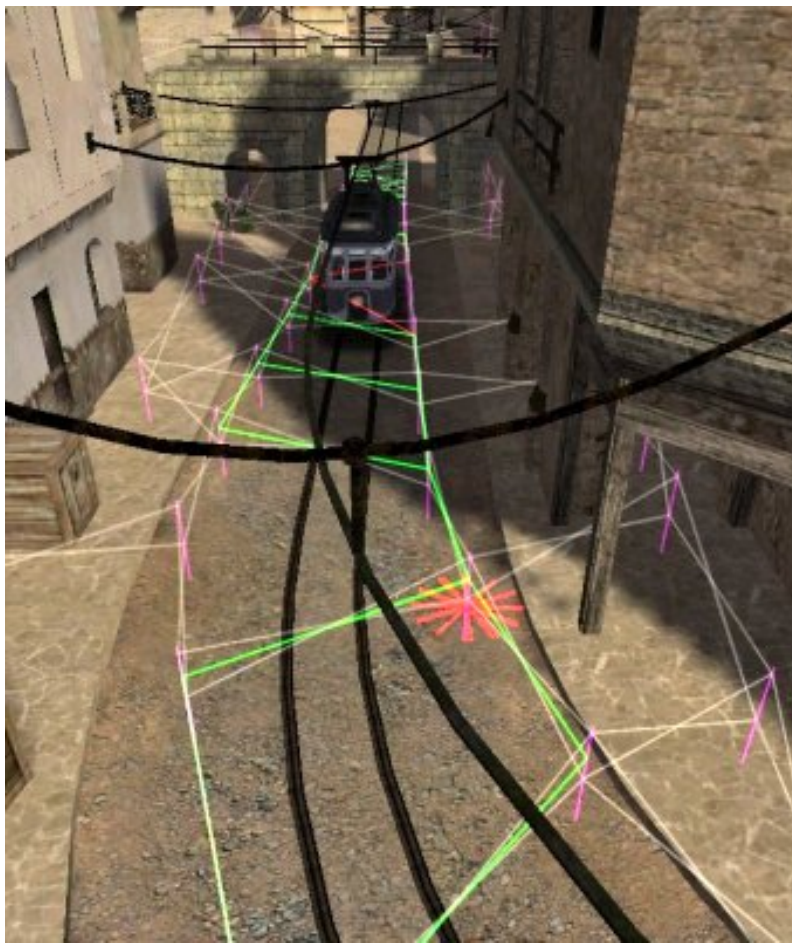
Keep in mind that vehicles can block paths, too. As far as Omni-bot navigation is concerned, vehicles are "wall-like" objects:



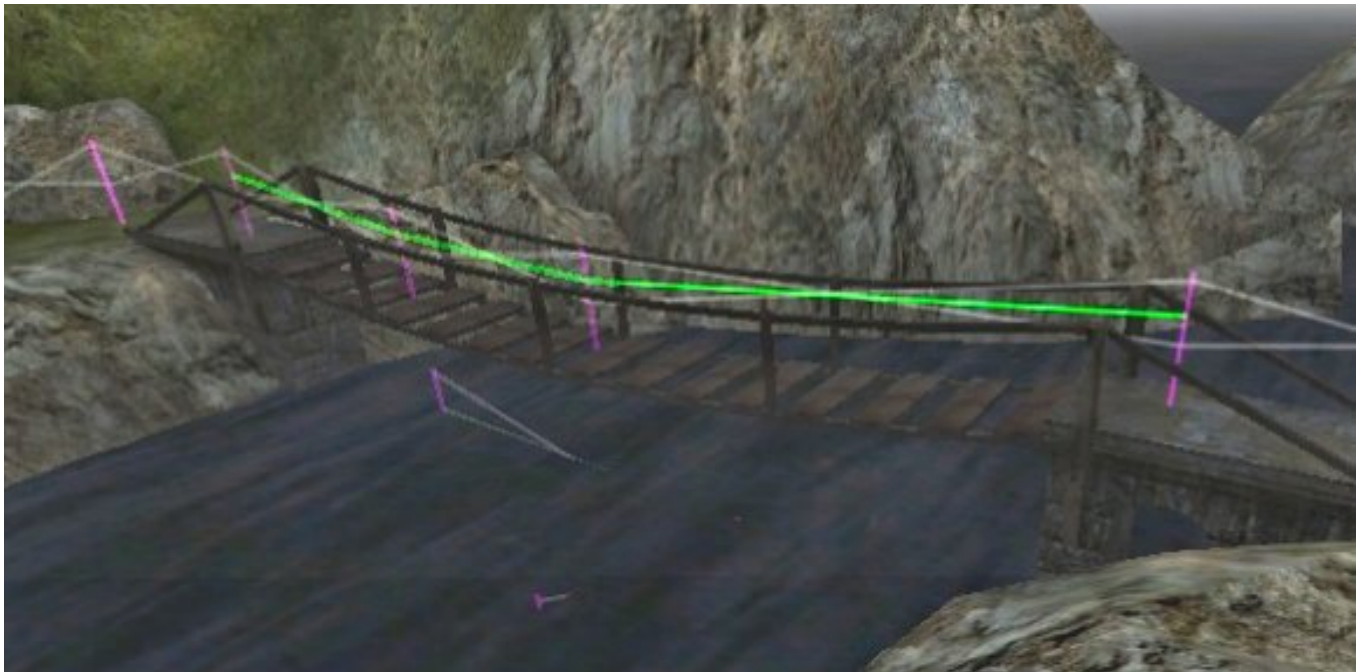
Note how the waypoints in the image above indicate an alternative path around the vehicle.

Normally, all the waypoints along the path of a vehicle should have the blockwall flag set, because otherwise the bots will run into the vehicle when it is destroyed or standing still for some other reason. The tug in Alpine Assault is a special case, because it can only stop at predefined places and is indestructible. There is a page on the Omni-bot Wiki dedicated to vehicle pathing. Here's how a vehicle route will typically look:





An example of blockbridge (taken from the Bergen map):



The check for blockbridge is basically a probe to see if there is a surface detected under the link between the waypoints. When there is no bridge built, the probe doesn't detect the surface, and results in a

blocked path. When the bridge is built, the surface is detected, and the link is opened.

Due to this method of detecting the blocked status of a path across a bridge, rope bridges and similar constructions with a slightly curved shape may require three or more waypoints, all of which should have blockbridge flags. Otherwise, the detection of a surface beneath the path might fail, as can be seen in this screenshot:



See the Waypointing page on the Omni-bot Wiki to learn more about blockable paths.

## Tactical Flags

The three waypoint flags that influence the bots' tactical behavior are attack, defend, and snipe. They create goals for the bots, much like the constructible and destructible objects that are auto-detected when a map is loaded.

**Sniper spots.** Probably one of the most popular and most widely used waypoint flags is the snipe flag. As with all flags, a snipe flag is added to a waypoint by standing over the waypoint in question and issuing the command `/bot waypoint_addflag snipe` in the game console. This flag makes bots that have the appropriate weapons camp on this waypoint for some time with their sniper rifle in scoped mode. Do not use it to make bots switch to scoped mode somewhere in an open area, they will camp there and might make fools of themselves, standing still in the middle of a market place with tons of enemies around them. See the Wiki page about sniper spots to learn more.

It is important to understand that sniper waypoints must have a valid facing. Without the facing, you could as well not add the sniper flag at all. A waypoint's facing is basically a vector associated with the waypoint that tells the bot in which direction to aim when sniping. Since version 0.6, Omni-bot adds the waypointer's direction of view automatically to a waypoint when it is placed. However, it's a good habit to adjust the facing the very moment you are adding the snipe flag, using the `waypoint_setfacing` command. You can easily bind this functionality to a key like this:



```
/bind f5 "bot waypoint_addflag snipe;bot waypoint_setfacing"
```

There are several reasons for adjusting the facing at the time the snipe flag is added: The waypoint may have no facing at all, because (some of) the waypoints were made with an older version of Omni-bot, or it may have some random facing that is totally inappropriate.

It is a good idea to actually join as a sniper when you plan to add sniper spots to a map. Then before issuing the setfacing command, try to take the position the sniping bot will have (e.g. go prone etc.), scope your rifle and aim where you want the bots to aim. When you use the `/bot waypoint_setfacing` command, your current aim direction will be saved as the waypoint's facing.

Also, sniper spots should in many cases have a smaller radius than other waypoints (say 20-30), because the exact position matters a great deal: A few inches to the left, and the sniper will expose himself to the enemy's sight, a few inches to the right, and the sniper might aim in a direction where he will hardly ever get a target, or aim at a rock that is two yards in front of him, blocking his sight.

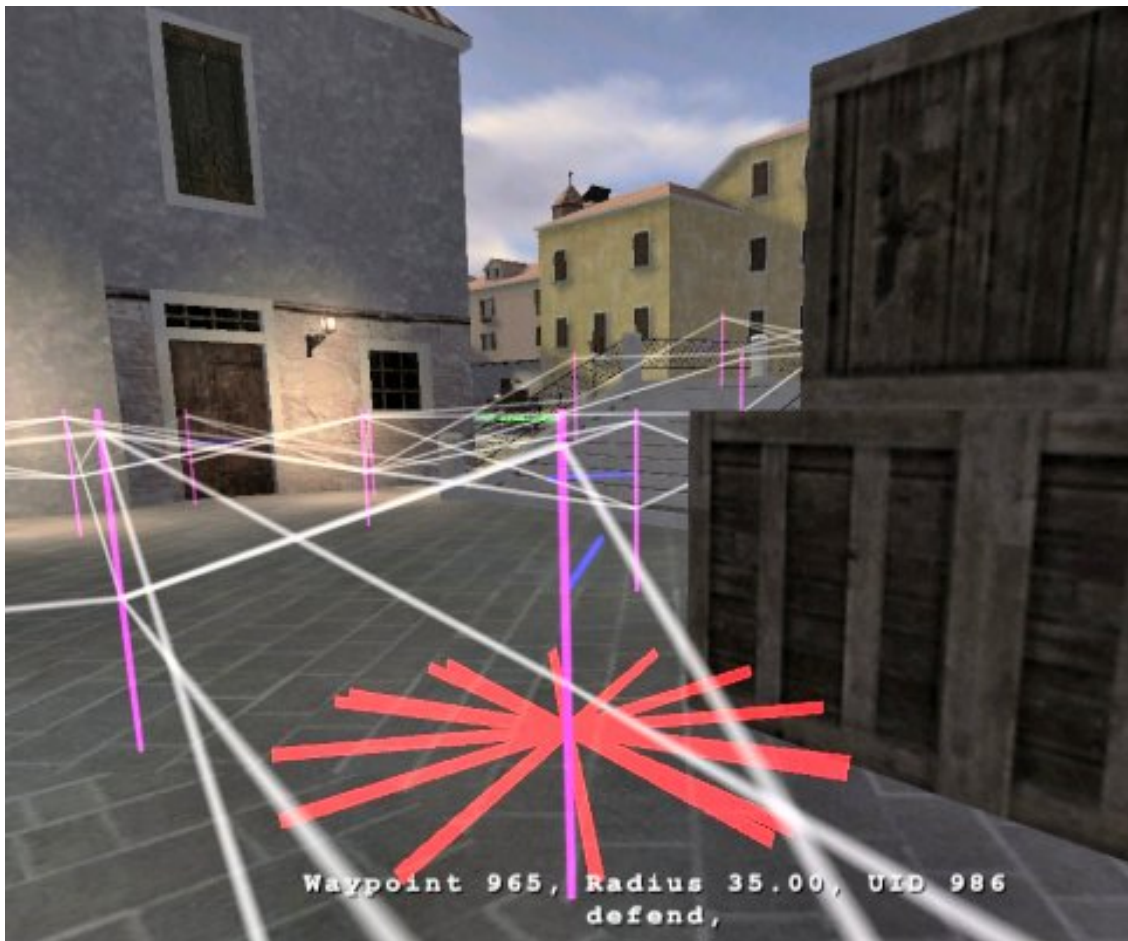
Two typical sniper spots on the Alpine Assault map:





**Attack and defend flags** are basically only goto goals, though by default, the bots will camp on defend spots for some time, which is not the case for attack spots. (This can be overridden via script.) Note that the attack goal will take precedence over the defend goal, so attack and defend goals shouldn't be active for the same team at the same time. attack and defend flags should be placed on waypoints near important objects, ideally in places where the environment provides some cover for the bots.

The following screenshot (taken from the Venice map) shows a well chosen defend spot on the square in front of the library. The facing is set in such a way as to make the bot primarily aim towards the bridge, from where an attack is likely to be expected; the crates on the right provide a good cover:



**Make your attack, defend, and snipe spots easily accessible from scripts.** Attack, defend, and sniper spots will frequently be obsoleted by the game flow. Using the well-known Venice map (ET) as an example again: It makes obviously no sense for bots to camp on the market place (where the game starts) once the Allies have taken over their second spawn. This is where the map script comes in. The script can serve other purposes as well, but disabling and enabling certain goals for the bots based on what has happened on a map is one of a script's primary tasks.

In order to make the goals easier to access from a script, name them using the `/bot waypoint_setname` command. Ideally, make up some naming scheme before-hand. If you put the team name in the waypoint name, this makes things even easier: For example, waypoint names like "phase 1 axis 1", "phase 1 axis 2" and so on will create goals for the bots that will look like "SNIPE\_phase\_1\_axis\_1" or "DEFEND\_phase\_2\_axis\_3". The next chapter will provide a (very) brief introduction to map scripting, and make it clear why this is helpful.

## Map Scripts

**Introduction.** This isn't the place to provide a full-fledged map scripting tutorial; see the appropriate Wiki page for all you need to know about that. All we're up to here is an introduction to the most basic concepts from a waypointer's point of view, mostly based on our Alpine Assault example.

A map script is a plain text file (written in a scripting language called GameMonkey, if you're interested), which should have the same name as the waypoint file with a `.gm` extension, and should be in the same folder (your nav folder). You can use the bot command `makemapgm` to generate a skeleton map script for you in the folder `/omni-bot/et/user/`. For our present purposes however, it's easier to start from scratch.



As we said in the last chapter, the map script's chief task from our current point of view is disabling and enabling certain goals for the bots, based on events that happen on a map. Let's see what that means.

**The OnMapLoad function.** The most important place to begin with is the OnMapLoad function. It is called automatically after the waypoint file is loaded and the bot goals are initialized. This function is the place where we set up the initial state of the goals in the map, as well as trigger callback functions that should be called when events happen in the game.

Pretending that we have set all our waypoint names according to a common scheme ("phase *n* axis *i*"/"phase *n* allies *i*"), our OnMapLoad function could look like this:

```
global OnMapLoad = function()
{
    //Disable all snipe, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "SNIPE_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "SNIPE_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "SNIPE_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "SNIPE_phase_1_allies.*");

    //Set up triggers for two important events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
};
```

The triggers used at the end of the OnMapLoad function can be found by watching the messages you see on your screen when playing (read more about triggers on the Wiki). Most maps have far more triggers than two, of course, but let's focus on these two for the moment.

**The Map table.** The Map.FunctionName syntax in the last two lines of the OnMapLoad function tells us that the callback functions set up there can be found in the Map table. Thus, in addition to our OnMapLoad function, we need at least a Map table in our script. (For those who have a basic knowledge of programming, a GameMonkey table is pretty much the same thing as an array in most other languages. For those who haven't, just accept the syntax for the moment, you'll gain a better understanding of what it's all about along the way.) So, assuming we have two triggers, we should add this to our GM script:

```
global Map =
{
    KeyStolen = function()
    {
```

```

},
KeyReturned = function()
{
},
};

```

Let's assume we have only two major events in our map. When the attacking team (ALLIES) has stolen the safe key, the defending team (AXIS) should focus on defending the safe that the attacking team will try to open with the key. If the key is returned, they should focus on the initial defend spots again:

```

global Map =
{
  KeyStolen = function()
  {
    //Disable phase 1 goals when the key is stolen:
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, false, "SNIPE_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, false, "SNIPE_phase_1_allies.*");

    //Enable phase 2 goals:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_2_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_2_allies.*");
  },

  KeyReturned = function()
  {
    //Disable phase 2 goals when the key is returned:
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_2_axis.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_2_allies.*");

    // ... and reenables phase 1 goals:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "SNIPE_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "SNIPE_phase_1_allies.*");
  },
};

```

**Disabling and enabling cappoint goals.** If these two events were all that happened on the map, we'd be pretty much done now. But there is one more thing we are going to cover here: the use of cappoints (capture points). One interesting feature of the Alpine Assault map is the existence of two items that must be captured, and two different capture points, one for each item. The second item (a briefcase with documents) is stored in a safe, and becomes available only after the safe has been opened with a key (the first item). Thus the safe is the capture point for the first item, while the second item must be brought to the halftrack (a vehicle at the Allies' first spawn point).

If you type `/bot show_goals "CAPPOINT_.*"` into the game console, you'll see some output similar to the following:

```
— Goal List —
1: CAPPOINT_halftrack -> 1100 serial 2 priority 1.00
2: CAPPOINT_safe -> 1100 serial 1 priority 1.00
— End Goal List —
```

The ones and zeros after the `"->"` are bits that indicate the availability for each team, in numerical order. That means for example that the goal `CAPPOINT_halftrack` is available for team 1 and 2 (in Enemy Territory, we can ignore whatever is said about the other teams). So we see that both cappoint goals are available at the moment, which is not what we want, because the bot that captures the key from the desk has no clue where to run with it.

That means we should disable the halftrack cappoint at the beginning; and we will need one more event that enables it and disables the safe as a cappoint. So let's update our `OnMapLoad` function:

```
global OnMapLoad = function()
{
    //Disable all sniper, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "SNIPE_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "SNIPE_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "SNIPE_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "SNIPE_phase_1_allies.*");

    //Disable unusable capture objectives and capture spots:
    SetAvailableMapGoals(Team.ALLIES, false, "CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, false, "FLAG_case");

    //Set up triggers for three events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
    OnTrigger("Allied team has opened the Safe!", Map.SafeOpened);
};
```

We add one more function to the Map table, so the second cappoint becomes available when the safe is open:

```
SafeOpened = function()
```

```

{
    //Enable new CAPPOINT and FLAG goals ...
    SetAvailableMapGoals(Team.ALLIES, true, "CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, true, "FLAG_case");

    // ... and disable the obsolete ones:
    SetAvailableMapGoals(Team.ALLIES, false, "CAPPOINT_safe");
    SetAvailableMapGoals(Team.ALLIES, false, "FLAG_key");
},

```

Putting all this together, we finally end up with the following (simplified) map script:

```

//=====
// Map script for Alpine Assault by d00d
// Works with Omni-bot 0.7
// Last update: Thu, 16 May 2008 15:07:38 GMT
//=====

global Map =
{
    KeyStolen = function()
    {
        //Disable phase 1 goals:
        SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_1_axis.*");
        SetAvailableMapGoals(Team.AXIS, false, "SNIPE_phase_1_axis.*");
        SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_1_allies.*");
        SetAvailableMapGoals(Team.ALLIES, false, "SNIPE_phase_1_allies.*");

        //Enable phase 2 goals:
        SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_2_axis.*");
        SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_2_allies.*");
    },

    KeyReturned = function()
    {
        //Disable phase 2 goals when the key is returned:
        SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_2_axis.*");
        SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_2_allies.*");

        // ... and reenables phase 1 goals:
        SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
        SetAvailableMapGoals(Team.AXIS, true, "SNIPE_phase_1_axis.*");
        SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
        SetAvailableMapGoals(Team.ALLIES, true, "SNIPE_phase_1_allies.*");
    },

    SafeOpened = function()
    {
        //Enable new CAPPOINT and FLAG goals ...
        SetAvailableMapGoals(Team.ALLIES, true, "CAPPOINT_halftrack");
    }
}

```

```

    SetAvailableMapGoals(Team.ALLIES, true, "FLAG_case");

    // ... and disable the obsolete ones:
    SetAvailableMapGoals(Team.ALLIES, false, "CAPPOINT_safe");
    SetAvailableMapGoals(Team.ALLIES, false, "FLAG_key");
},
};

global OnMapLoad = function()
{
    //Disable all sniper, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "SNIPE_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "SNIPE_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "SNIPE_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "SNIPE_phase_1_allies.*");

    //Disable unusable capture objectives and capture spots:
    SetAvailableMapGoals(Team.ALLIES, false, "CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, false, "FLAG_case");

    //Set up triggers for three events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
    OnTrigger("Allied team has opened the Safe!", Map.SafeOpened);
};

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET\\_Waypointing\\_Tutorial](http://omni-bot.com/wiki/index.php?title=ET_Waypointing_Tutorial)"

This page has been accessed 751 times. This page was last modified 23:03, 16 May 2008.



# Vehicle Pathing

From Omni-bot Wiki

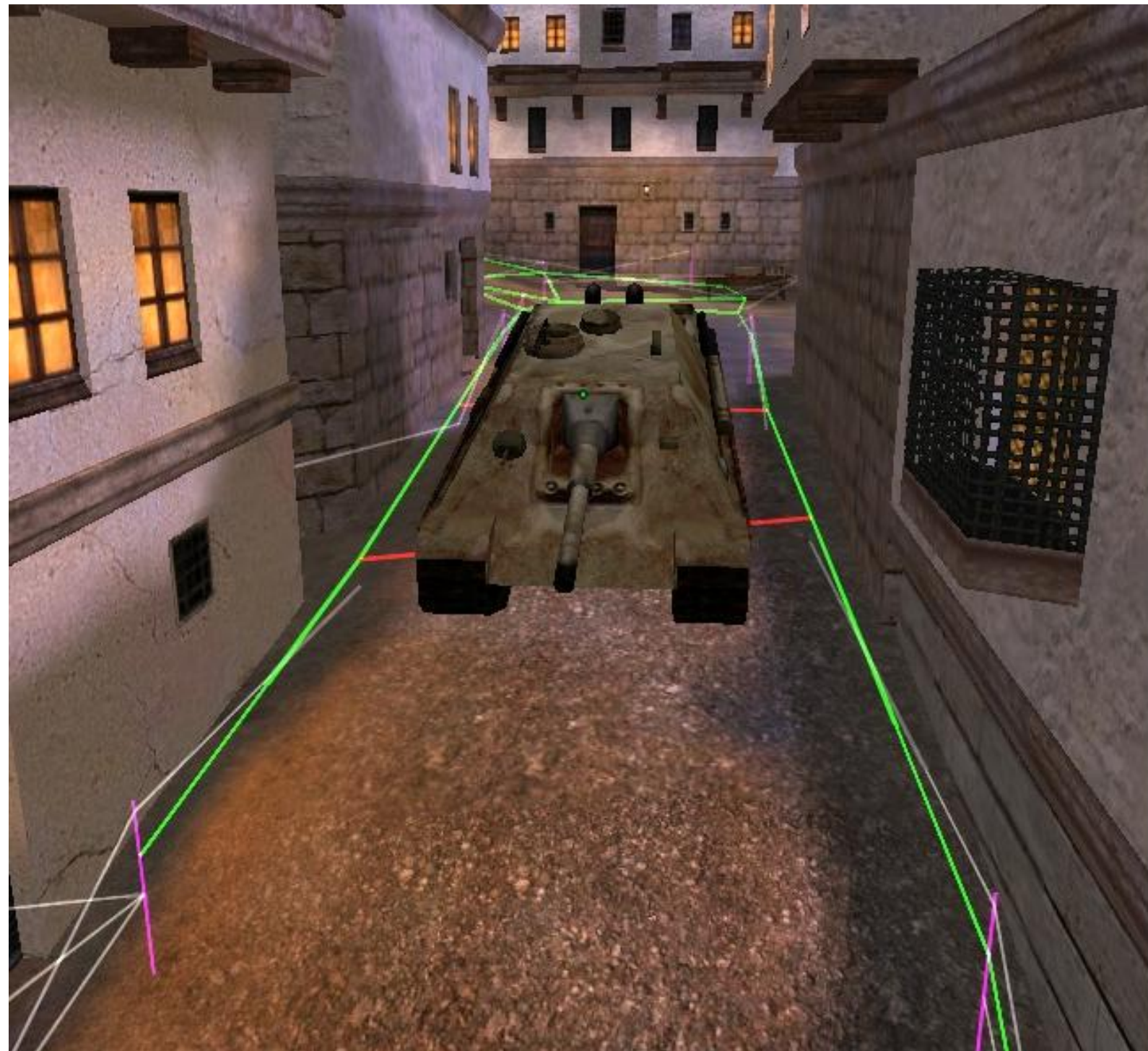
[ET Main Page](#)

**Vehicle Pathing**

[Archives](#)

## Pathing For Vehicles

When pathing for vehicles in Enemy Territory, waypointers must create paths on at least one side of the vehicle for the entire route. It is recommended to create paths on each side of the vehicle as shown here:





Usage of the blockable flag is necessary wherever the tank may cross over any connections between the paths. In the picture, the red lines mark the connections currently blocked by the tank. To add the blockable flags to the waypoints you may consider binding a key as typical vehicle paths require several blockable connections:

```
bind x bot waypoint_addflag blockable wall
```

Placement of the waypoints is critical as bots that try to repair the tank will travel to the closest node to the constructable goal first. No waypoints should be placed in a location where they will be potentially covered by the vehicle. In Omni-bot version 0.65 and above, there are two tools available that will help ensure that the vehicles can be repaired along the entire route; BuildVehiclePath and CheckVehiclePath. Essentially, these two commands will damage the vehicle at a given interval which makes it easy to see where waypoints should be placed.

Cheats Enabled is required for both commands. To enable cheats while waypointing, you will need to start the map with the devmap command:

```
/devmap mapname
```

It is also recommended that the g\_engineerChargeTime cvar is set to 0 to make rebuilding the vehicle go quicker.

## Build Vehicle Path

```
syntax: /bot bvp goalname1 goalname2 <optional seconds>
example: /bot bvp MAP_CONSTRUCTION_tank_construct MAP_MOVER_tank 2
```

goalname1 should be set the the construct goal and goalname2 should be set to the vehicles goal name. The last parameter is the number of seconds to wait before the vehicle will be damaged.

This command will automatically damage the tank after the waypointer repairs it. The default interval is 2 seconds and can be set with the optional parameter.

To turn the command off, type:

```
/bot bvpo
```

## Check Vehicle Path

```
syntax: /bot cvp goalname1 goalname2 <optional seconds>
example: /bot cvp MAP_CONSTRUCTION_tank_construct MAP_MOVER_tank 2
```

goalname1 should be set the the construct goal and goalname2 should be set to the vehicles goal name. The last parameter is the number of seconds to wait before the vehicle will be damaged.

This command is used to check if bots can repair the vehicle at each point along the vehicle path. Once issued, the command will damage the vehicle and add an engineer bot to the correct team. The waypointer can then watch and update any pathing that requires adjusting.

To turn the command off, type:

```
/bot cvpo
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Vehicle\\_Pathing](http://omni-bot.com/wiki/index.php?title=Vehicle_Pathing)"

This page has been accessed 368 times. This page was last modified 01:58, 2 May 2008.

# Sniper Spots

From Omni-bot Wiki

ET : RTCW

Sniper Spots

Archives

Probably one of the most popular and most widely used waypoint flags is the snipe flag. A snipe flag is added to a waypoint by issuing the command `"/bot waypoint_addflag snipe"` in the game console. This flag makes bots that have the appropriate weapons camp on this waypoint for some time with their sniper rifle in scoped mode. Do not use it to make bots switch to scoped mode somewhere in an open area, they will camp there and might make fools of themselves, standing still in the middle of a market place with tons of enemies around them. When used appropriately, sniping can be a very powerful (and perhaps slightly amusing) part of the bots' behavior.

## Contents

- 1 Setting up the waypoint
- 2 Test
- 3 Combining flags
- 4 Scripting

## Setting up the waypoint

**First things first.** In order to add a sniper flag to an existing waypoint, bring down the in-game console and type `"/bot waypoint_addflag snipe"`. If you want to make sure the sniper bot will aim in the correct direction, move your own crosshairs where you want the bot to aim and type `"/bot waypoint_setfacing"` in the console. (Depending on the game, the slash at the beginning may be unnecessary.) Make sure you pick a waypoint where it's relatively safe for the bots to camp for at least 10 seconds or so. Choose waypoints where you would like to snipe for some time yourself: behind bushes, windows, etc.

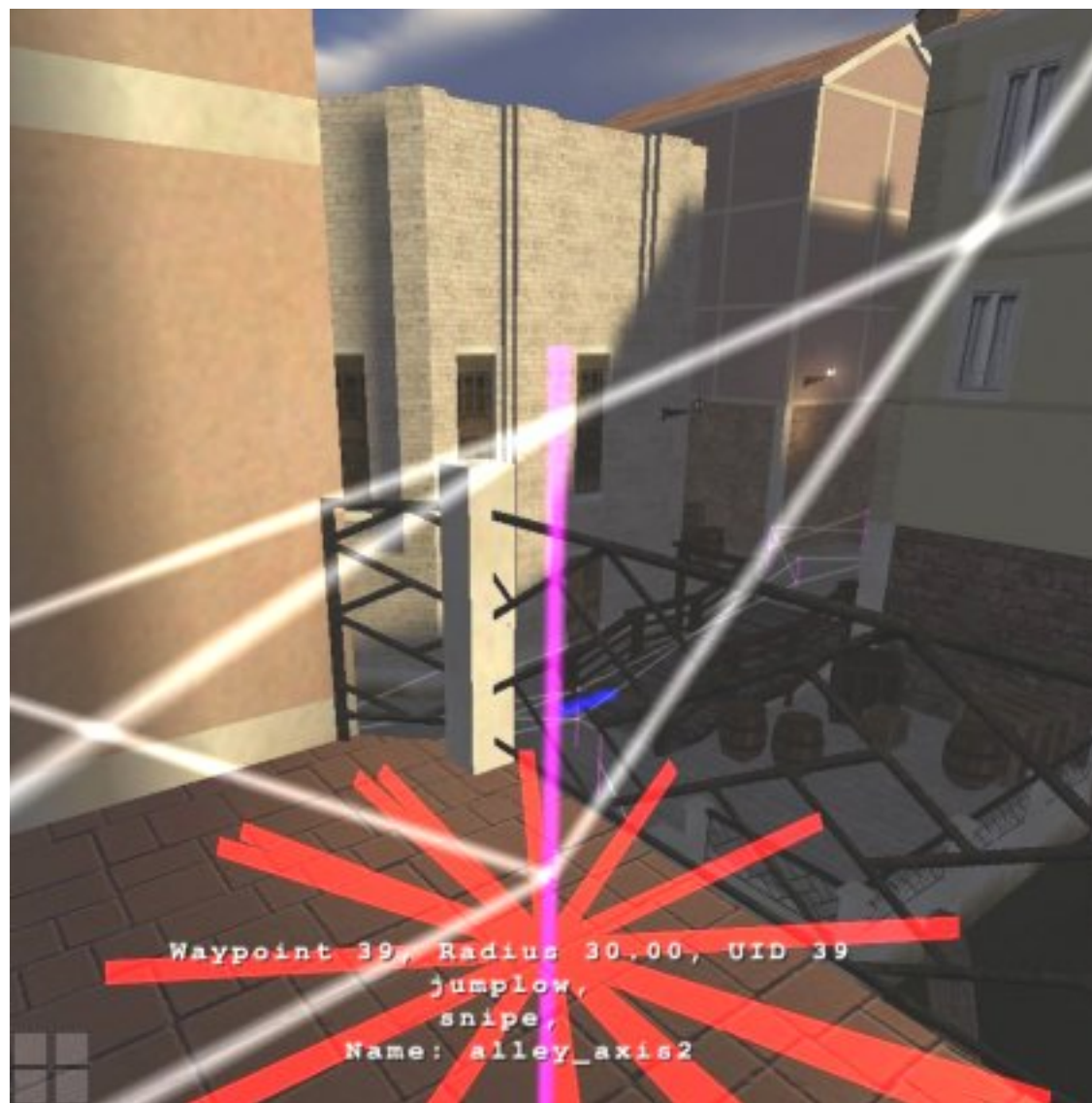
**Facing: the facts.** It is important to understand that sniper waypoints must have a valid facing set. Without the facing, you could as well not add the sniper flag at all. A waypoint's facing is basically a vector associated with the waypoint that tells the bot in which direction to aim. Since version 0.6, Omni-bot adds the waypointer's facing automatically to a waypoint when it is placed. However, it's a good habit to adjust the facing the very moment you are adding the snipe flag, using the waypoint setfacing command. You can easily bind this functionality to a key like this: `/bind f5 "bot waypoint_addflag snipe;bot`



waypoint\_setfacing". There are several reasons for adjusting the facing at the time the snipe flag is added: The waypoint may have no facing at all, because (some of) the waypoints were made with an older version of Omni-bot, or it may have some random facing that is totally inappropriate.

The following images might help to get the general idea. They show the same waypoint, viewed from two different perspectives (screenshots from Venice). The right image shows more or less the point of view the sniper bot will have, the blue line indicates the facing of the waypoint (bot waypoint viewfacing on):





**Join as sniper when waypointing.** It is a good idea to actually join as a sniper when you plan to add sniper spots to a map. Then before issuing the setfacing command, try to take the position the sniping bot will have (e.g. go prone etc.), scope your rifle and aim where you want the bots to aim.

**Radius.** Also, sniper spots should in many cases have a smaller radius than other waypoints (say 20-30), because the exact position matters a great deal: A few inches to the left, and the sniper will expose himself to the enemy's sight, a few inches to the right, and the sniper might aim in a direction where he will hardly ever get a target, or aim at a rock that is two yards in front of him, blocking his sight.

## Test

After adding the sniper flags, save the waypoints, restart the map (/map\_restart) and add a few covert ops bots to the appropriate team, like this: /bot addbot 1 5. Spectate them in first-person view in order to make sure they really aim where you thought they would. Due

to the nature of the scoped view, a slight variation in the position and/or aim direction can have a huge effect on what the bot actually "sees", so be prepared for some disappointments here. Here's a screenshot taken from spectating a bot sniping at our first example waypoint, that shows us that we have finally accomplished what we were up to:



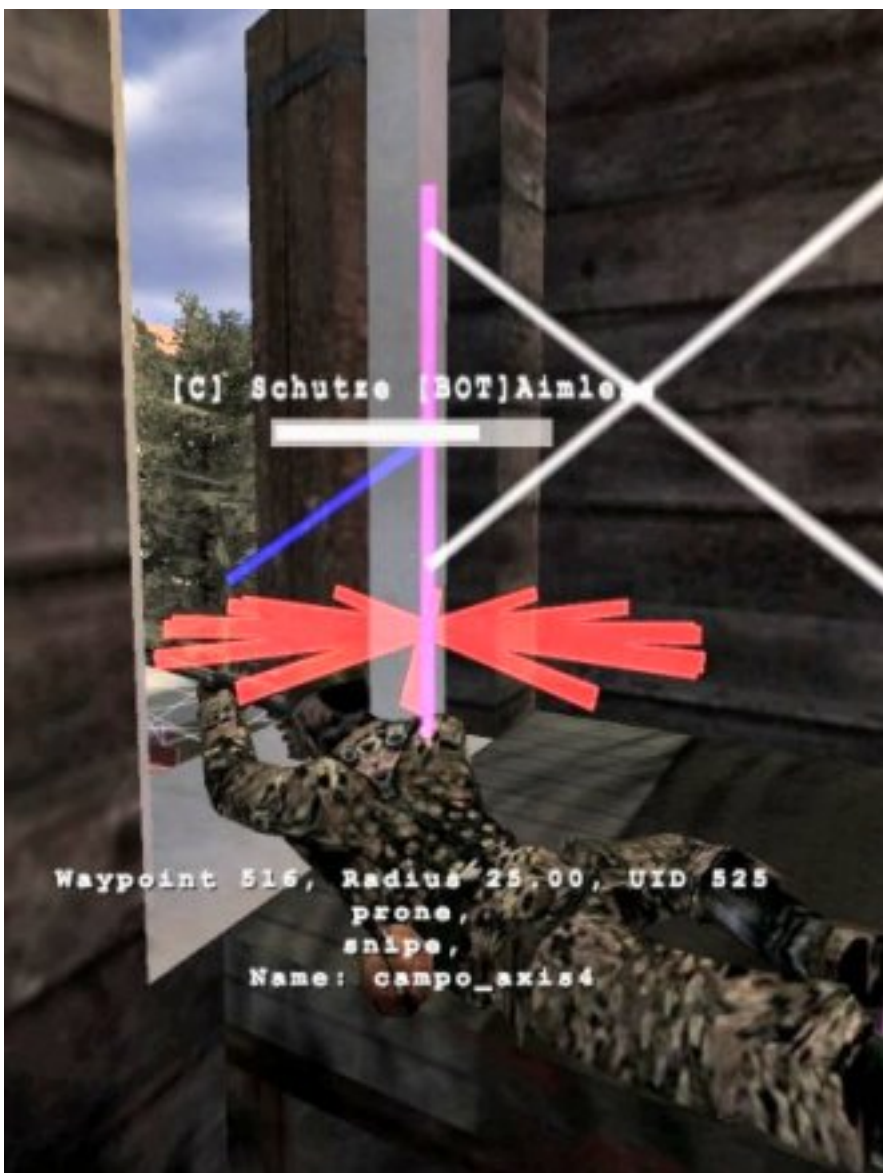
An additional (admittedly somewhat troublesome) test is to join as member of the opposite team, walk make-believe naively along the path the snipers are supposed to cover, and wait for the "bang!" If there is no "bang!", you'll probably have to tweak the facing of the sniper spot. Adding bots as sniper targets will serve the same purpose in most cases.

## Combining flags

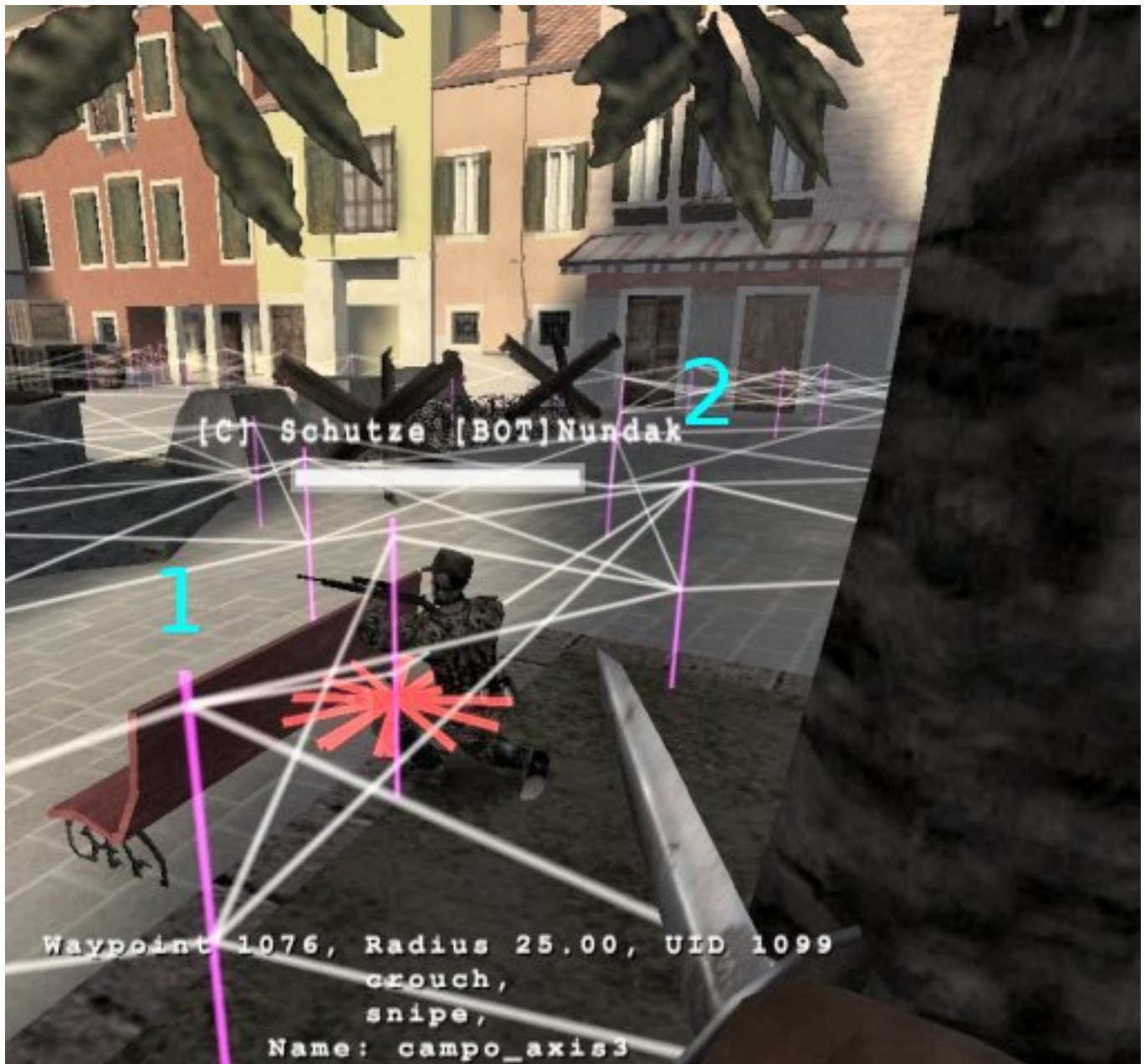
Other waypoint flags that make good company for the snipe flags are crouch and prone, and perhaps one of the team flags if the sniper spot is placed in an area where the other team has no business to do. (See the Scripting section below for a method to make a sniper spot team-specific without using flags.) It is *not* such a good idea to combine sniper flags with attack or defend flags (let alone mine or others), because this will make other bots go there and shove the sniper away from his cunningly chosen position, hurting his aim ability. If you want a defending soldier and a sniper on the same spot, add two waypoints close to each other and flag one as sniper spot, the other as defend spot. As a general rule, try to keep sniper spots out of paths that are frequented by other classes a lot. (Often this does not require any special effort, because most waypointers will naturally choose sniper spots where no other bots are likely to have any business, like the one in the example above.)

The following images show a prone and a crouching sniper bot, respectively. The direct bidirectional connection between waypoint 1 and 2 in the right image will make other bots

that need to move from 1 to 2 or vice versa walk behind the sniper, thus not disturbing him too much. Waypoints 1 and 2 (could) have defend or attack flags.







## Scripting

Much like attack and defend goals, sniper spots will frequently be obsoleted by the game flow. Using the Venice map (ET) as an example: It makes obviously no sense for snipers to camp on the market place (where the game starts) once the Allies have taken over their second spawn. This is where the map script comes in. The script can serve other purposes as well, but disabling and enabling certain goals for the bots based on what has happened on a map is one of a script's primary tasks.

In order to make the sniper goals easier to access from a script, name them using the bot waypoint\_setname command. If you put the team name in the waypoint name, this makes things even easier: For example, waypoint names like "phase 1 axis 1", "phase 1 axis 2" and so on will create snipe goals for the bot that will look like "SNIPE\_phase\_1\_axis\_1".

In the OnMapLoad function of the map script, you can disable the team-specific sniper spots for the other team in one go, like this:

```
SetAvailableMapGoals( TEAM.AXIS, false, "SNIPE_phase_1_allies.*" );  
SetAvailableMapGoals( TEAM.ALLIES, false, "SNIPE_phase_1_axis.*" );
```

The third argument of this function is a regular expression. Using this script based technique for making sniper spots team-specific has a great advantage over using team flags: It doesn't put the waypoint completely off limits for one team, so medics etc. can still go there.

Another helpful thing for the OnMapLoad function:

```
Util.SetMaxUsersInProgress( 1, "SNIPE_.*" );
```

This should make sure the sniper spots won't be too crowded if there is a large number of sniper bots on a map.

In the OnBotJoin function of the script, you can control how long the bots will camp on a sniper spot. The time they spend on sniper spots will vary randomly between MinCampTime and MaxCampTime (the values are in seconds):

```
global OnBotJoin = function( bot )  
{  
    bot.SetGoalProperty( "SNIPE", "MinCampTime", 15 );  
    bot.SetGoalProperty( "SNIPE", "MaxCampTime", 45 );  
};
```

Based on the events that are part of the game flow, you can additionally disable sniper spots that are obsoleted by some event:

```
OnTankPastBarrier = function(trigger)  
{  
    SetAvailableMapGoals( TEAM.ALLIES, false, "SNIPE_phase_1.*" );  
    SetAvailableMapGoals( TEAM.AXIS, false, "SNIPE_phase_1.*" );  
};
```

This will disable all the phase 1 sniper spots for both teams.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Sniper\\_Spots](http://omni-bot.com/wiki/index.php?title=Sniper_Spots)"

This page has been accessed 376 times. This page was last modified 03:04, 2 May 2008.

# Waypointing Trick

From Omni-bot Wiki

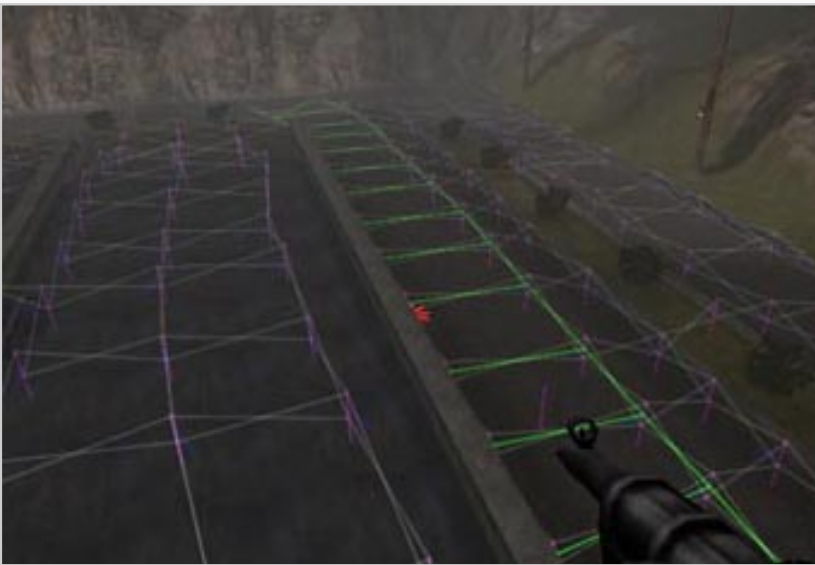
ET Main Page	Waypointing Trick	Archives
--------------	-------------------	----------

## Waypointing Trick : Save your time!

You wanna have a good waypoint?

You have no time and wanna have a quick way to edit waypoint?

You can use radius!



I wanna have a good waypoint, but I must draw like this?

No! You can do like these page.



Do it like this !

With **/bot waypoint setdefaultradius** and **/bot waypoint setradius**

to anti-bot stuck, you can draw some small radius of waypoint near it. Enjoy!

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Waypointing\\_Trick](http://omni-bot.com/wiki/index.php?title=Waypointing_Trick)"

This page has been accessed 127 times. This page was last modified 02:02, 2 May 2008.

# ET Waypoints

From Omni-bot Wiki

## 0.65 / 0.66 Released Waypoints

Map	Map URL	WP URL	Author	Gm Script	Current Owner	Notes
Adlernest	[1]	INC	crapshoot / blindman	yes	crapshoot	included in 0.66 release
Battery	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release
Braundorf_b4	[2]	INC	crapshoot	yes	crapshoot	included in 0.66 release
Caen	[3]	INC	crapshoot	yes	crapshoot	included in 0.66 release
ET Ice	[4]	INC	crapshoot	yes	crapshoot	included in 0.66 release
Frostbite	[5]	INC	crapshoot	yes	crapshoot	included in 0.66 release
Fueldump	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release
Goldrush	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release
Oasis	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release



Radar	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release
Railgun	ET	INC	crapshoot / Jaskot	yes	crapshoot	included in 0.66 release
Venice	[6]	INC	crapshoot	yes	d00d	included in 0.66 release

## 0.60/61 Released Waypoints

Map	Map URL	WP URL	Author	Gm Script	Current Owner	Notes
Alpine Assault (final)	[7]	[8]	d00d	yes	d00d	included in 0.66 release
Bergen	[9]	[10]	d00d	yes	d00d	included in 0.66 release
Bremen b2	[11]	[12]	Blindman / IRATA	yes	IRATA	0.61 version
Dover Bridge	[13]	[14]	d00d	Yes Trivial	d00d	included in 0.66 release
JFF Playground b1	[15]	[16]	Brockkiller	no	Brockkiller / IRATA	included in 0.66 release  waiting for user feedback
SupplyDepot2	[17]	[18]	Red Dwarf	yes	uploaded	included in 0.66 release

Transmitter	[19]	[20]	d00d	yes	d00d	included in 0.66 release
Wolken 1 Final	[21]	[22]	MetalHead	Yes Trivial	MetalHead	Uploaded
Wolken 2 Final	[23]	[24]	MetalHead	Yes Trivial	MetalHead	Uploaded

## Works In Progress

Map	Map Url	Compatible	Author	Gm Script	Notes
110 factory 2.0.0	[25]	0.61	IRATA [*]	yes	testing mapscript phase2, wp-version 0.7 uploaded
AA-Guns	[26]	0.66	Advieszeur	yes	Uploaded: [27]
Axis Lab	[28]	0.66	d00d	yes	Test version uploaded.
Canyon Lake	[29]	0.66	MetalHead	yes	Still testing.
Cluedo 1.1.0	[30]	0.61	IRATA	yes	Preparing next release, more : Cluedo Info
DeadCold	[31]	0.66	Advieszeur	yes	Testing/ Problems with defend flag



et_ufo_b3	[32]	0.66	crapshoot	yes	complete wp / script with an Airstrike goal to be converted @ 0.7 release.  link: et_ufo_b3.zip
Fun Beach	[33]	0.66	Advieszeur	yes	Uploaded: [34]
Hotchkiss	[35]	0.61	QuickFix	yes	working on script
JFF Playground b2 V2	[36]	0.61	IRATA	yes	still testing
King of the Radio	[37]	0.61	C4r3?	yes	Testphase
Maat b2	[38]	0.66	Advieszeur	yes	Waypointing
MLB Daybreak	[39]	0.61	ailmanki	yes	attack/defend management, waiting for 0.65
MLB Egypt	[40]	0.66	Advieszeur	yes	Uploaded: [41]
mp_rocket	[42]	0.61	420Blunt	yes	just beginning
Navarone	[43]	0.61	ailmanki	yes	attack/defend management, waiting for 0.65
negoshk_b2	[44]	0.65	Cube	yes	Test phase
Over the Top	[45]	0.66	>WES< P! NNAN / Advieszeur	yes	Test phase

Pitch Black	Summer: [46]  Winter: [47]	0.66	Advieszeur	yes	uploaded: [48] and [49]
Secret Bay	[50]	0.66	>WES< P! NNAN / Advieszeur	yes	(advieszeur: Just helping). Almost Done,  testversion ready see: [51]
Secret Bay	[52]	0.66	MetalHead	yes	Almost Done, bit of tweaking required.
StargateCenter 1945 V2	[53]	0.66	>WES< P! NNAN	yes	test phase
CityContest (V1)	[54]	0.66	Hank_officer	yes	my map and my waypoint XD... still a beta on my server
snatch2	[55]	0.66	Hank_officer	yes	Download Here : Completed Waypoints
Tankbuster 2.0.0	[56]	0.66	>WES< P! NNAN	yes	Test phase
V2 Factory	[57]	0.66	d00d/IRATA [*]	yes	Test phase

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET\\_Waypoints](http://omni-bot.com/wiki/index.php?title=ET_Waypoints)"

This page has been accessed 1,635 times. This page was last modified 16:39, 5 May 2008.

# Enemy Territory Script Goals

From Omni-bot Wiki

[ET Main Page](#)

**Script Goals**

[Archives](#)

## Contents

- 1 Enemy Territory Script Goals
  - 1.1 goal\_airstrike.gm
  - 1.2 goal\_askforammo.gm
  - 1.3 goal\_askforhealth.gm
  - 1.4 goal\_combatmovement.gm
  - 1.5 goal\_deliversupplies.gm
  - 1.6 goal\_disguise.gm
  - 1.7 goal\_dispenseammo.gm
  - 1.8 goal\_dispensehealth.gm
  - 1.9 goal\_escortvehicle.gm
  - 1.10 goal\_grenadetarget.gm
  - 1.11 goal\_mountvehicle.gm
  - 1.12 goal\_ridevehicle.gm
  - 1.13 goal\_supplyself.gm
  - 1.14 goal\_useswitch.gm

## Enemy Territory Script Goals

### goal\_airstrike.gm

This script goal checks for a table of waypoints set up for the bot to go to and throw an airstrike cannister. Typical usage would be in cases where there is good chance enemies may be, but the bot may not have a line of sight to. For instance in Oasis if the Allies have the flag and the wall is not destroyed, you may want allied field ops to throw an airstrike over the wall.

### goal\_askforammo.gm

This script goal checks the bots ammo situation occasionally and looks for ammo packs on the ground before calling for ammo using the ET voice macro "Need Ammo!".

### goal\_askforhealth.gm

This script goal checks the bots health situation occasionally and looks for health packs on the ground before calling for health using the ET voice macro "Need Medic!".

### **goal\_combatmovement.gm**

This script goal runs while the bot has a target and implements some dodging and strafing movement while in combat.

### **goal\_deliversupplies.gm**

This script goal listens for teammate requests for health and ammo, and if the bot is a field ops or medic capable of answering the request, the bot will travel to the callers location and provide them with the requested health/ammo. Useful for mobile mg42 and other defensive positions to be able to replenish ammo or health in the field.

### **goal\_disguise.gm**

This script goal allows a covert ops bot to steal the uniform of an enemy. Desired behavior (not implemented yet), is to then favor objectives that then become available from the disguised state.

### **goal\_dispenseammo.gm**

This is a simple reimplementation of the old behavior of dropping ammo shortly after spawn.

### **goal\_dispensehealth.gm**

This is a simple reimplementation of the old behavior of dropping health shortly after spawn.

### **goal\_escortvehicle.gm**

This script goal gives bots the ability to escort vehicles in a map such as goldrush or fueldump. It supports multiple escort 'slots', allowing multiple bots to escort vehicles at different offsets from the vehicle.

- requires mapgoal configuration

### **goal\_grenadetarget.gm**

Gives bots the ability to grenade targets such as mg42's, barbed barriers, etc.

## **goal\_mountvehicle.gm**

Gives bots the ability to mount vehicles such as the tanks in goldrush and fueldump and use the vehicles mounted weaponry.

- requires mapgoal configuration

## **goal\_ridevehicle.gm**

Implements mover riding functionality, such as for the trams in railgun.

## **goal\_supplyself.gm**

This script goal provides medics and fieldops the ability to give themselves health or ammo when they need it.

## **goal\_useswitch.gm**

Allows bots to treat hitting switches in a map as a goal, also used heavily in railgun.

- requires mapgoal configuration

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Enemy\\_Territory\\_Script\\_Goals](http://omni-bot.com/wiki/index.php?title=Enemy_Territory_Script_Goals)"

This page has been accessed 245 times. This page was last modified 04:46, 21 June 2008.

# Doom 3

## From Omni-bot Wiki

There is currently no text in this page, you can search for this page title in other pages or edit this page.

---

Retrieved from "http://omni-bot.com/wiki/index.php?title=Doom\_3"

# Quake 4

## From Omni-bot Wiki

There is currently no text in this page, you can search for this page title in other pages or edit this page.

---

Retrieved from "http://omni-bot.com/wiki/index.php?title=Quake\_4"

# Fortress Forever

**From Omni-bot Wiki**

There is currently no text in this page, you can search for this page title in other pages or edit this page.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Fortress\\_Forever](http://omni-bot.com/wiki/index.php?title=Fortress_Forever)"



# RTCW:Waypoints

From Omni-bot Wiki

## Works In Progress

Map	Map Url	Compatible	Author	Gm Script	Notes
axis_complex	axis_complex	0.70	420Blunt / AG3NT	yes	finished / needs testing
braundorf_b3	braundorf	0.70	crapshoot / AG3NT	yes	finished / needs testing
mp_assault	INC	0.70	crapshoot / AG3NT	yes	finished / needs testing
mp_base	INC	0.70	crapshoot / AG3NT	yes	finished / needs testing
mp_beach	INC	0.70	crapshoot / AG3NT	yes	finished / needs testing
mp_castle	INC	0.70	AG3NT / crapshoot / Denny	yes	finished / needs testing
mp_chateau	GOTY	0.70	crapshoot / Denny / AG3NT	yes	finished / needs testing
mp_dam	INC	0.70	AG3NT / Denny	yes	finished / needs testing
mp_depot	INC	0.70	crapshoot / Denny	yes	finished / needs testing

mp_destruction	INC	0.70	crapshoot / Denny	yes	waypoints, needs script
mp_ice	GOTY	0.70	crapshoot / AG3NT	yes	finished / needs testing
mp_keep	GOTY	0.70	crapshoot / Denny	yes	finished / needs testing
mp_rocket	GOTY	0.70	crapshoot / 420Blunt / AG3NT	yes	finished / needs testing
mp_sub	INC	0.70	crapshoot	yes	finished / needs testing
mp_trenchtoast	GOTY	0.70	crapshoot	yes	waypoints, needs script
mp_village	INC	0.70	crapshoot / AG3NT	yes	finished / needs testing
se_bunker_v2	SE Pack v2 patch	0.70	420Blunt / crapshoot	yes	finished / needs testing
te_escape2	te_escape2	0.70	AG3NT	yes	finished / needs testing
te_frostbite	frostbite	0.70	crapshoot / AG3NT	yes	finished / needs testing
te_ufo	ufo	0.70	crapshoot / AG3NT	yes	finished / needs testing
valor	valor	0.70	AG3NT	yes	finished / needs testing

# RTCW:Map Scripting

From Omni-bot Wiki

Contents

- 1 Standard Format
  - 1.1 Map Table
  - 1.2 OnMapLoad
  - 1.3 OnBotJoin
  - 1.4 Triggers
    - 1.4.1 Supported Triggers
- 2 Native Functions
  - 2.1 ChangeSpawnPoint
  - 2.2 GetGameTimeLeft
  - 2.3 GetGameType
  - 2.4 GetReinforceTime
  - 2.5 MaxViewDistance
  - 2.6 SetAvailableMapGoals
  - 2.7 SetGoalPriority
  - 2.8 SetMapGoalProperties
  - 2.9 TargetBreakableDist
- 3 Utility Functions
  - 3.1 RTCWUtilities
    - 3.1.1 RTCWUtil.ChangeClass
    - 3.1.2 RTCWUtil.ChangeSpawn
    - 3.1.3 RTCWUtil.ClearMainGoals
    - 3.1.4 RTCWUtil.ClearSecondaryGoals
    - 3.1.5 RTCWUtil.CountClass
    - 3.1.6 RTCWUtil.CountTeam
    - 3.1.7 RTCWUtil.DisableGoal
    - 3.1.8 RTCWUtil.EnableGoal
    - 3.1.9 RTCWUtil.LimitToClass
    - 3.1.10 RTCWUtil.NoSnipe
    - 3.1.11 RTCWUtil.RandomSpawn
    - 3.1.12 RTCWUtil.SelectWeapon
    - 3.1.13 RTCWUtil.SetPrimaryGoals
    - 3.1.14 RTCWUtil.ShowActiveGoals
    - 3.1.15 RTCWUtil.StopSniping
    - 3.1.16 RTCWUtil.SwitchWeapon
  - 3.2 Utilities
    - 3.2.1 Util.AddInvVehicle
    - 3.2.2 Util.AliveCount
    - 3.2.3 Util.DisableGroup
    - 3.2.4 Util.DistanceView
    - 3.2.5 Util.EnableGroup
    - 3.2.6 Util.GetGroup
    - 3.2.7 Util.OnTriggerPosition
    - 3.2.8 Util.RemoveGoal
    - 3.2.9 Util.SetGoalOffset

- 3.2.10 Util.SetGoalPosition
- 3.2.11 Util.SetGroup
- 3.2.12 Util.ShowGroup
- 3.2.13 Util.SetMaxUsersInProgress
- 3.2.14 Util.SetPositionGoal
- 3.2.15 Util.ShowGoalInfo
- 3.2.16 Util.ShowGoalName
- 3.2.17 Util.ShowGoalOffset
- 3.2.18 Util.AddUsePoint
- 3.2.19 Util.AddUseWp
- 4 Conditionals
- 5 Scripting Tools
  - 5.1 makegm
- 6 Debugging Tools
  - 6.1 debugbot
  - 6.2 ScriptDebug
  - 6.3 GameMonkey interpreter
  - 6.4 print
  - 6.5 show\_goals
  - 6.6 show\_goalroutes
  - 6.7 draw\_goals
  - 6.8 draw\_goalroutes

## Standard Format

All official map scripts share a common layout. Having a standard for map scripts has several benefits; ease of documentation, debugging, and implementation of scripted goals are among those benefits. A typical RTCW map script will consist of one Map table and two functions that are automatically called by Omni-Bot:

```
global Map =
{
};

global OnMapLoad = function()
{
};

global OnBotJoin = function( bot )
{
};
```

Using makemapgm will ensure that a map script is in a standard format.

## Map Table

The Map Table has become an important part of an ET map script as some scripted goals rely on its existence. It also provides a 'safe' place to store map specific information. Typically it will consist of map variables, triggers / functions, and in some cases additional tables:

```
global Map =
```

```
{
    SomeGoal = "PLANT_something",
    someVar = true,
    someTable =
    {
        someVar = false,
    },

    some_trigger = function(trigger)
    {
        print("some_trigger");
    },
};
```

It is important to note that when you add to a table you use a comma and not a semi-colon. Calling functions or referencing variables within the example table is done like this:

```
Map.someVar = false; //changes the value of someVar to false
print(Map.SomeGoal); //will print BUILD_some_construct
Map.someTable.someVar = true; //changes the value of someTable.someVar to true
```

The key thing to note is that you need to put Map. in front of whatever you want to call or reference within the Map table.

## OnMapLoad

The OnMapLoad function is used to initialize settings for the map. It is automatically called by Omni-Bot whenever a map loads (i.e. /map\_restart). Setting up triggers, goal availability, and goal bias' are the most common jobs performed by this function:

```
global OnMapLoad = function()
{
    OnTrigger( "Some Team did Something!", Map.some_trigger );
    SetAvailableMapGoals( TEAM.AXIS, false, Map.SomeGoal );
    SetGoalPriority( Map.SomeGoal, 1.1 );
};
```

## OnBotJoin

The OnBotJoin function is automatically called by Omni-Bot when a bot joins. Typical usage for OnBotJoin is for setting specific properties on the bot that you want to be map specific. View distance and breakable object distance are the two most commonly used:

```
global OnBotJoin = function( bot )
{
    bot.MaxViewDistance = 2500;
    bot.TargetBreakableDist = 150.0;
};
```

## Triggers

A trigger is an event that is recognized by Omni-Bot in game. With map scripting, triggers can be set up to

perform operations when a map event occurs. There are two steps to setting up triggers; defining the trigger and setting up the trigger function.

Triggers are most commonly defined in OnMapLoad using the OnTrigger function:

```
global OnMapLoad = function()  
{  
    OnTrigger("string", function);  
};
```

The "string" parameter is usually found using the wm\_announce messages seen during the game. This must be matched exactly including capitalization, punctuation, and color codes (if any). To see the wm\_announce message requires the waypointer to either play through the map and perform the objectives or look through the map script that is included inside a map's pk3 file.

If the wm\_announce messages appear in non-standard color, the latter method is often more convenient. The map's native script (not to be confused with the Omni-bot map script) is usually the file <mapname>.script in the /maps/ folder inside the pk3 file. Open the pk3 file with some archive utility, e.g. IZArc.

Another method of finding the string parameter is using the command **/bot debugtriggers**. This command will output to console everytime a recognized event occurs. The syntax will be similar to:

```
<---> Trigger: TagName: Service Door breached! Action: announce Entity: 0x7016dd8c Activator: 0
```

The string just after TagName: is what the OnTrigger function will expect as the string parameter. For additional information about debugtriggers see the debugtriggers page. This method is required for maps that may not have wm\_announce messages that correspond to recognized map events.

Once you've collected the triggers you need, the contents of the console can be written to a plain text file by using the command /condump mytextfile. The file will be located in the omnibot folder under the game installation folder. By copying the trigger names from that file instead of typing them, the risk of typing errors is greatly reduced.

The function parameter of the OnTrigger function is the name of a function that the waypointer creates for the particular trigger. These can be named anything the waypointer wants, but should be somewhat intuitive.

Once the string parameter has been identified and a function name has been determined, the trigger definition will look something like this:

```
global OnMapLoad = function()  
{  
    OnTrigger("Service Door breached!", Map.service_breached );  
};
```

The last part of the setup is to create the trigger function that will contain code for the game to process each time the event occurs. In all official map scripts, this is done inside the Map table:

```
global Map =  
{  
    service_breached = function( trigger )  
    {  
        //things to do when this event occurs  
    },  
};
```

```
};
```

It is important to note is that the function is inside the map table and should have a comma outside the closing bracket and not a semi-colon. The print statement inside the trigger function is not necessary, but is a good way to test that the trigger is working. As an additional service to the users, you can comment these print statements out when you're done testing, so their console isn't flooded with information that doesn't mean much to them.

## Supported Triggers

Most events in RTCW will fall into the following Omni-Bot recognized categories:

- announce
- announce\_icon
- closed
- defused
- dynamited
- exploded
- mover\_goto x y z
- opened
- repair\_mg42
- returned
- round end
- stolen
- team\_announce
- thirty second
- two minute

If you are unsure if the event you want to set up a trigger for falls into one of these categories, use `/bot debugtriggers` in game.

**NOTE:** For the returned trigger (i.e. documents returned) use `/bot debugtriggers` output for when the flag is returned by expiring (no player returns it). If the regular `wm_announce` message is used, the event that occurs when the flag expires will not be used. It is recommended to manually steal the flag, `/kill` and wait for them to be automatically returned. An example can be found in `radar.gm`. Most of the time it will be something like "Flag returned flag!"

## Native Functions

This section consists of Omni-Bot functions specifically designed for use in map scripts.

### ChangeSpawnPoint

```
syntax: bot.ChangeSpawnPoint(int spawnptid);
example: bot.ChangeSpawnPoint(1);
```

note: typical spawn points are 0 - default, 1 - axis, 2 - allies

### GetGameTimeLeft

```
syntax: GetGameTimeLeft();
returns: Time left in the game in seconds
```

## GetGameType

syntax: `GetGameType()`;  
returns: The current game type

## GetReinforceTime

syntax: `bot.GetReinforceTime()`;  
returns: Time left before the bots next spawn

## MaxViewDistance

syntax: `bot.MaxViewDistance = <distance>;`  
example: `bot.MaxViewDistance = 2500;`

note: typically set in OnBotJoin as this sets the property individually

## SetAvailableMapGoals

syntax: `SetAvailableMapGoals( Team, true / false, goalname );`  
example: `SetAvailableMapGoals( TEAM.AXIS, true, "MAP_FLAG_someflag" );`  
example: `SetAvailableMapGoals( TEAM.ALLIES, false, Map.Flag );`  
example: `SetAvailableMapGoals( TEAM.AXIS, false, "DEFEND.*" );`

note: the goalname parameter supports expressions

## SetGoalPriority

syntax: `SetGoalPriority( goalname, priority, team, class, <optional persistent> );`  
example: `SetGoalPriority( Map.Flag, 1.0, 0, 0 ); //all teams and all classes`  
example: `SetGoalPriority( "DEFUSE_somedyno.*", 0.0, TEAM.AXIS, CLASS.ENGINEER, true ); //the persistent parameter is used for dynamic goals that may not have been created yet`

## SetMapGoalProperties

syntax: `SetMapGoalProperties( Goal, table )`  
example: `SetMapGoalProperties( "ATTACK_.*", {mincamptime=15, maxcamptime=30} );`

note: this function is typically called from OnMapLoad

## TargetBreakableDist

syntax: `bot.TargetBreakableDist = <distance>;`  
example: `bot.TargetBreakableDist = 100;`

note: typically set in OnBotJoin as this sets the property individually

note: used to allow bots to target breakables like windows. should be set to a relatively low number.

## Utility Functions



The functions listed in this section are custom functions created to support specific scenarios in game. They are located in ~/omni-bot/rtcw/scripts and ~/omni-bot/global\_scripts respectively.

## RTCWUtilities

RTCW utility functions are located in the ~/omni-bot/rtcw/scripts/rtcw\_utilities.gm file.

### RTCWUtil.ChangeClass

```
syntax: RTCWUtil.ChangeClass( team, originalclass, newclass, revert, maxbots )  
example: RTCWUtil.ChangeClass( TEAM.ALLIES, CLASS.SOLDIER, CLASS.MEDIC, false, 1 );  
example: RTCWUtil.ChangeClass( TEAM.ALLIES, CLASS.SOLDIER, CLASS.MEDIC, true, 1 );
```

### RTCWUtil.ChangeSpawn

```
syntax: RTCWUtil.ChangeSpawn(team, spawnpoint, numbots)  
usage: used to force bots to use a given spawn point. optionally limiting the number of bots  
to use the spawn
```

### RTCWUtil.ClearMainGoals

```
syntax: RTCWUtil.ClearMainGoals();  
usage: This function will deactivate all main goals for both teams.
```

goals deactivated: PLANT MOUNTMG42 MOVER CHECKPOINT FLAG

### RTCWUtil.ClearSecondaryGoals

```
syntax: RTCWUtil.ClearSecondaryGoals();  
usage: This function will deactivate all secondary goals for both teams.
```

goals deactivated: ARTILLERY REPAIR

### RTCWUtil.CountClass

```
syntax: RTCWUtil.CountClass( team, class )  
example: RTCWUtil.CountClass( TEAM.ALLIES, CLASS.ENGINEER );
```

usage: can be used to determine if a team has enough of a critical class for the map

### RTCWUtil.CountTeam

```
syntax: RTCWUtil.CountTeam( team )  
example: RTCWUtil.CountClass( TEAM.ALLIES );
```

### RTCWUtil.DisableGoal

```
syntax: RTCWUtil.DisableGoal(goalname, <optional true>);  
example: RTCWUtil.DisableGoal("FLAG_someflag");
```

usage: disables the goal for both teams. the optional true parameter is used to disable all

goals except for ROUTE goals.

## RTCWUtil.EnableGoal

syntax: `RTCWUtil.EnableGoal(goalname);`

example: `RTCWUtil.EnableGoal("FLAG_someflag");`

usage: enables the goal for both teams.

## RTCWUtil.LimitToClass

syntax: `RTCWUtil.LimitToClass(goalname, team, class1, class2, class3);`

usage: used to limit specific goals to a specific class or classes

example: `RTCWUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER)` //only soldiers on the allied team

example: `RTCWUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER, CLASS.MEDIC)` //only soldiers and medics

## RTCWUtil.NoSnipe

syntax: `RTCWUtil.NoSnipe(bot);`

usage: soldier bots will not choose a mauser when selecting a weapon

note: this is typically called in OnBotJoin

## RTCWUtil.RandomSpawn

syntax: `RTCWUtil.RandomSpawn(team, spawnpoint);`

usage: used to have bots randomly choose to spawn at the given spawnpoint

## RTCWUtil.SelectWeapon

syntax: `RTCWUtil.SelectWeapon(bot, weapon);`

usage: the given bot will switch to the given weapon if it is the correct class for the weapon

note: typically used in OnBotJoin to have soldiers choose a specific weapon

example: `RTCWUtil.SelectWeapon(bot, WEAPON.MAUSER);`

## RTCWUtil.SetPrimaryGoals

syntax: `RTCWUtil.SetPrimaryGoals(priority);`

usage: used for setting priorities of common goals.

note: this is typically called in OnMapLoad and effects the following goals (in order of priority):

CAPPOINT FLAGRETURN PLANT CHECKPOINT FLAG

## RTCWUtil.ShowActiveGoals

syntax: `RTCWUtil.ShowActiveGoals();`

usage: shows active goals for both teams

note: should only be used for debugging  
the command /bot sag will call this function

## RTCWUtil.StopSniping

syntax: RTCWUtil.StopSniping();

usage: all bots currently using a mauser will switch to a different weapon

note: this is typically called inside trigger functions

## RTCWUtil.SwitchWeapon

syntax: RTCWUtil.SwitchWeapon(weapon);

usage: all qualifying bots will switch to the given weapon

example: RTCWUtil.SwitchWeapon(WEAPON.PANZERFAUST); //all soldiers will switch to panzer

note: typically used in trigger functions

## Utilities

Utility functions are located in the ~/omni-bot/global\_scripts/utilities.gm file.

### Util.AddInvVehicle

syntax: Util.AddInvVehicle( goalname )

usage: adds a **vehicle** to the Map.InvVehicle **table**

used **for script goals in** cases where the **vehicle** is invulnerable, but shows as "**dead**"

### Util.AliveCount

syntax: Util.AliveCount( **team**, **class** )

usage: returns the number of bots alive on a **team** with a given **class**

### Util.DisableGroup

syntax: Util.DisableGroup(groupname, **team**);

example: Util.DisableGroup( "**somegroupname**", **TEAM.SOMETEAM** );

usage: used to disable a set of goals in the given group for a given team

### Util.DistanceView

syntax: Util.DistanceView(<optional off|0>);

usage: /bot dist

/bot dist off

/bot dist 0

note: used for determining bot.MaxViewDistance settings. aim at a position as far as you can

see and type /bot dist

## Util.EnableGroup

syntax: `Util.EnableGroup(groupname, team);`

example: `Util.EnableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to enable a set of goals in the given group for a given team

## Util.GetGroup

syntax: `Util.GetGroup(groupname);`

example: `Util.GetGroup( "somegroupname" );`

usage: this function will return a table of goals belonging to the given group name

## Util.OnTriggerPosition

syntax: `Util.OnTriggerPosition( goalname, wpname, tolerance, wpfunction )`

example: `Util.OnTriggerPosition( Map.Mover_train1, "depotyard", 200.0, Map.tug_depotyard );`

note: used for setting up positional triggers for movers.

## Util.RemoveGoal

syntax: `Util.RemoveGoal( goalname );`

example: `Util.RemoveGoal( "MOVER_truck" );`

note: this command will remove the goal from the map goal table

## Util.SetGoalOffset

syntax: `Util.SetGoalOffset( x, y, z, GoalName );`

example: `Util.SetGoalOffset( 0, 0, 30, "BUILD_construct" )`

usage: the x y and z parameters are added to the origin of the goal to move the location of where the bots will look for the goal.

## Util.SetGoalPosition

syntax: `Util.SetGoalPosition( x, y, z, GoalName );`

example: `Util.SetGoalPosition( 4534, 2168, -199, "BUILD_construct" )`

usage: used to give the goal a new origin.

note: using /devmap to load the map and issuing the /viewpos command in console will give your origin in the map

## Util.SetGroup

syntax: `Util.SetGroup(goalname, groupname);`

example: `Util.SetGroup( "somegoalname", "somegroupname" );`

usage: this function will add a given goal to a given groupname

## Util.ShowGroup

syntax: `Util.ShowGroup(groupname);`

example: `Util.ShowGroup("somegroupname");`

usage: this function will list all goals in the given group in the console.

## Util.SetMaxUsersInProgress

syntax: `Util.SetMaxUsersInProgress( Users, GoalNames );`

example: `Util.SetMaxUsersInProgress( 15, "CHECKPOINT.*" );`

usage: used to set the maximum number of bots going for a particular goal(s).

## Util.SetPositionGoal

syntax: `Util.SetPositionGoal( goalname1, goalname2 );`

example: `Util.SetPositionGoal( Map.Build_tank_construct, Map.Mover_tank );`

usage: sets the origin of one goal to match the origin of another. useful for centering construct goals on movers

## Util.ShowGoalInfo

syntax: `Util.ShowGoalInfo(goalname);`

command: `/bot sgi 'goalname'`

usage: used to print a goals current information in console including the goals health, status of the DEAD flag,  
and the goals position

## Util.ShowGoalName

syntax: `Util.ShowGoalName(radius, showOffset);`

command: `/bot sgn <radius> <optional true>`

usage: issue the command `/bot sgn` to find the goalname of any goals within 100 units. Optionally expand the radius and  
find your current offset from the goal with `/bot sgn 500 true`

## Util.ShowGoalOffset

syntax: `Util.ShowGoalOffset(goalname);`

command: `/bot sgo 'somegoalname'`

usage: used to determine a players current offset from a goal. the values given in console can be used for goals that  
require positional offsets

## Util.AddUsePoint

syntax: `Util.AddUsePoint(goalname, position);`

example: `Util.AddUsePoint( "somegoalname", Vector3(123, 456, 789) );`

usage: used to add a Use Point for a goal at a given position.

## Util.AddUseWp

syntax: `Util.AddUseWp(goalname, waypointname);`

example: `Util.AddUseWp( "somegoalname", "somewaypointname" );`

usage: used to add a Use Point for a goal at a given waypoints position.

## Conditionals

Return to Castle Wolfenstein is a complex game type. Maps are not linear and events do not necessarily happen in a specific order. Conditional statements are necessary in some cases to ensure that availability of goals is consistent with the waypoints idea of a map flow. A conditional statement begins with the 'if' function. In script, it is used to check given parameters before executing some code:

```
if ( a == b )
{
    //do something
}
```

In this example it checks if something is equal to something else before executing any code in the following brackets. If something does not equal something else, anything inside the brackets will be ignored. This conditional can be extended to make something happen if the first conditional was not met with the 'else' command:

```
if ( a == b )
{
    //do something
}
else
{
    //do something else
}
```

And finally, this can be extended even further using the 'else if' command:

```
if ( a == b )
{
    //do something
}
else if ( a == c )
{
    //do something else
}
else
{
    //do something else
}
```

```
    //do something else
}
```

Conditionals can get more complex by adding additional checks. The following example checks if both are true before executing the code in the example:

```
if ( a == b && c == d )
{
    //do something
}
```

If a check is dependant on one of many checks being true, the 'or' operator is used:

```
if ( a == b || c == d )
{
    //do something
}
```

To test if something is true or false, the syntax used is:

```
if ( !something )
{
    //do something
}
```

The apostrophe in front of the parameter checks if something is false while a `!=` checks if something doesn't equal something else:

```
if ( a != b )
{
    a = b;
}
```

Checking if a value is greater or lesser than another value is done as follows:

```
if ( a > b || c < d )
{
    //do something
}
```

In ET maps where the map flow is not pre-determined, it is important to add conditionals to some triggers in which goals are activated. While the bots order of objective completion can be set by the waypointer, the waypointer can not predict when a human player may complete a particular objective. So if goals are activated with a specific order in mind and a human player activates a goal out of turn, you may have goals available to bots earlier or later than intended; potentially breaking the gameplay.

## Scripting Tools

### makegm

makemapgm is a command that autogenerates a skeleton map script. Usage of this command will ensure a standard format and save time. Using the command is fairly straight forward:

Step 1: Load the map you want a script for

Step 2: Type /bot makegm in console. It will display a message if executed correctly

Step 3: Copy the mapname.gm file from ~/omni-bot/et/usr to ~/omni-bot/et/nav, Be sure to have backed up any mapname.gm file you may have been working on that exists in the nav folder.

The OnTriggers in OnMapLoad will need to be modified as makemap.gm does not find the information needed. Simply edit the "MISSING STRING" parameter of the OnTrigger function and the triggers should be working correctly.

## Debugging Tools

### debugbot

debugbot is a valuable tool to use if bots aren't behaving as expected. It will give success and failure messages in console for short and long term goals.

```
syntax: /bot debugbot all goals  
or: /bot debugbot <bot-name> goals
```

It is recommended to have only one or two bots connected when issuing this command with the all parameter as it will give debug output for each bot.

### ScriptDebug

Script debugging is crucial as a high percentage of errors are syntax related. There are two ways to enable script debugging:

- /bot script\_debug 1
- [EnableScriptDebug\(true\)](#);

EnableScriptDebug(true); can be placed in et\_autoexec.gm while /bot script\_debug 1 will need to be executed in console each time the map loads. If an error in script occurs while script debugging is enabled, the error will be listed in console in red text with the line number of the error.

note: in the omnibot.log, script errors will be written whether script debugging is enabled or not.

### GameMonkey interpreter

The GameMonkey download available at [www.somedude.net/gamemonkey/](http://www.somedude.net/gamemonkey/) contains an interpreter for gm files (gme.exe) that can be used as a syntax checker. With this, you won't have to start the game only to find you forgot a closing bracket or a semicolon.

Ideally, use this with an editor that can run command line applications and capture their output, such as SciTE or PSPad.

### print

print is a low level means of debugging a script. It can be used in cases where you want to test if a trigger is working correctly or if the map script has loaded:



```
global OnMapLoad = function()  
{  
    print("OnMapLoad");  
}
```

The print statement can also be used in-game via script\_run if you want to inspect the value of a variable:  
/bot script\_run "print(variable)"

## show\_goals

syntax: /bot show\_goals <optional> <optional p>  
usage: used to list goal(s) and their properties in console

note: the first optional parameter supports expressions. /bot show\_goals DEFEND.\* will give output for all DEFEND goals.  
the second optional parameter is used to show team and class specific priorities that have been set with SetGoalPriority.

show\_goals can be useful to test whether or not goals are available when they are expected to be and that they have the expected priority setting. The output from show goals is similar to:

```
PLANT_sometarget -> 1000 serial 3 bias 1.00
```

The number just to the right of the arrow represents teams and status. The first number is for Axis and the second is for Allies. 1 means it's available for that team while 0 means it is unavailable. In this case, the dynamite goal is currently available for the Axis team.

## show\_goalroutes

syntax: /bot show\_goalroutes <optional>  
usage: used to list routes used for a give goal or goals

## draw\_goals

syntax: /bot draw\_goals [on|1|off|0] <optional>  
usage: the optional parameter can be used to display one or several goals.

Note: the optional parameter supports regular expressions. /bot draw\_goals .\*BUILD.\* will draw BUILD goals.

This will draw the location of goals to your screen. Useful to see where bots "think" an object is located.

## draw\_goalroutes

syntax: draw\_goals on/off <optional goal name expression>  
usage: used to draw the given goal(s) routes

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=RTCW:Map\\_Scripting](http://omni-bot.com/wiki/index.php?title=RTCW:Map_Scripting)"

This page has been accessed 132 times. This page was last modified 03:14, 22 August 2008.

# Team Fortress 2

**From Omni-bot Wiki**

There is currently no text in this page, you can search for this page title in other pages or edit this page.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Team\\_Fortress\\_2](http://omni-bot.com/wiki/index.php?title=Team_Fortress_2)"

# Debug Window

From Omni-bot Wiki

**Contents**

- 1 Omni-bot Debug Window
- 2 Windows
  - 2.1 Console
  - 2.2 Map
  - 2.3 Profiler
  - 2.4 State Tree

## Omni-bot Debug Window

The Debug Window is primarily a debugging and informational aid for Omni-bot. When enabled, it spawns an OpenGL powered window along side the game, and is primarily used when you are running the game in windowed mode. It provides an interface to look at the bots state tree, a script console, a map, log, profiler, and eventually it will be exposed to scripts so that scripted tools or utility information can be displayed to it as well.

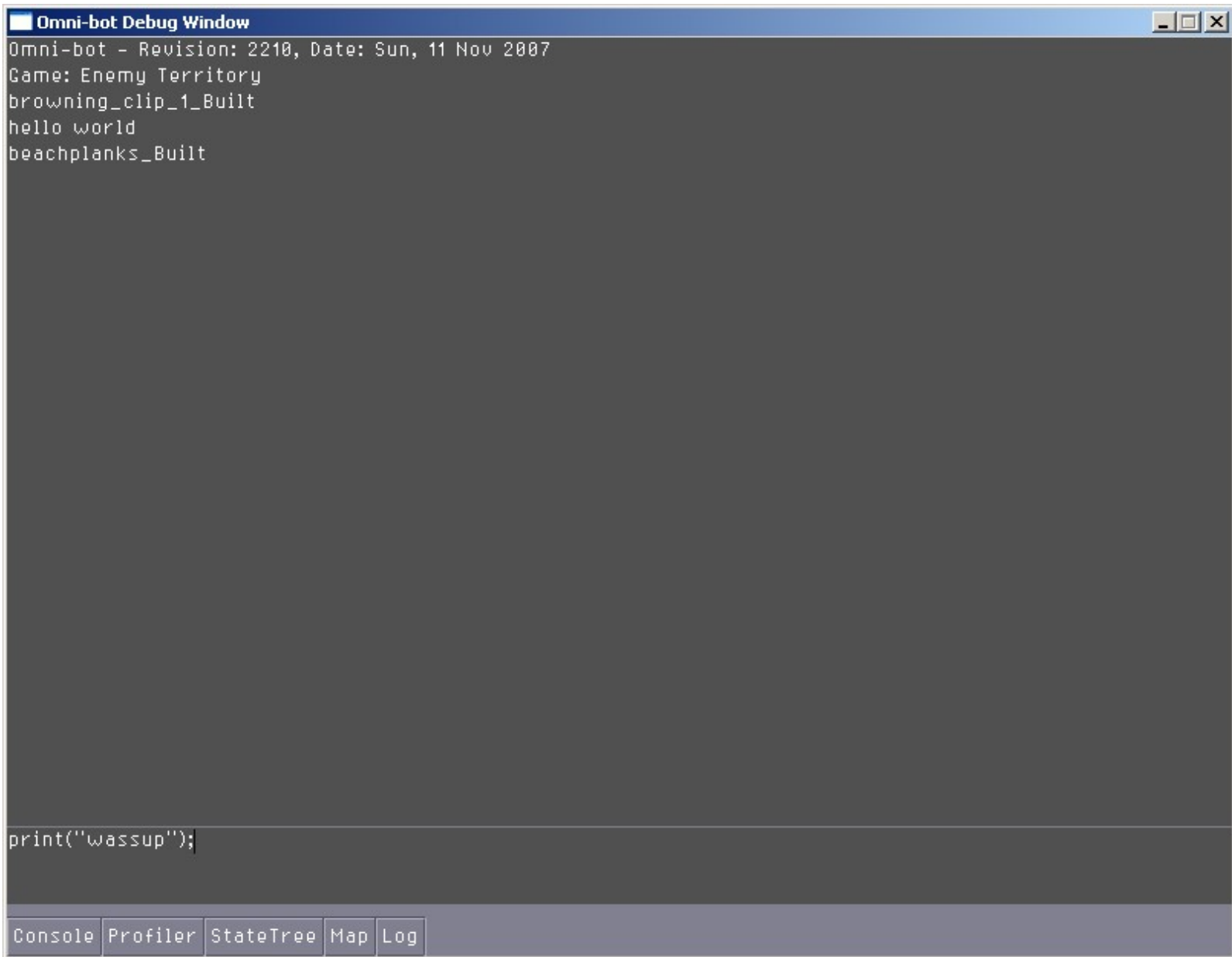
The previous debug window was powered by FLTK, and while it is a good toolkit for GUI applications, it isn't meant to be refreshed and redrawn as much as I needed it to be, and as a result incurred a high cost of updating each frame. Now, by being OpenGL powered, it can take advantage of the inherent speed of hardware acceleration and eventually some 3d elements if needed, all while staying cheap to process and render.

The debug window is now powered by OpenGL, guichan, and SFML

## Windows

### Console

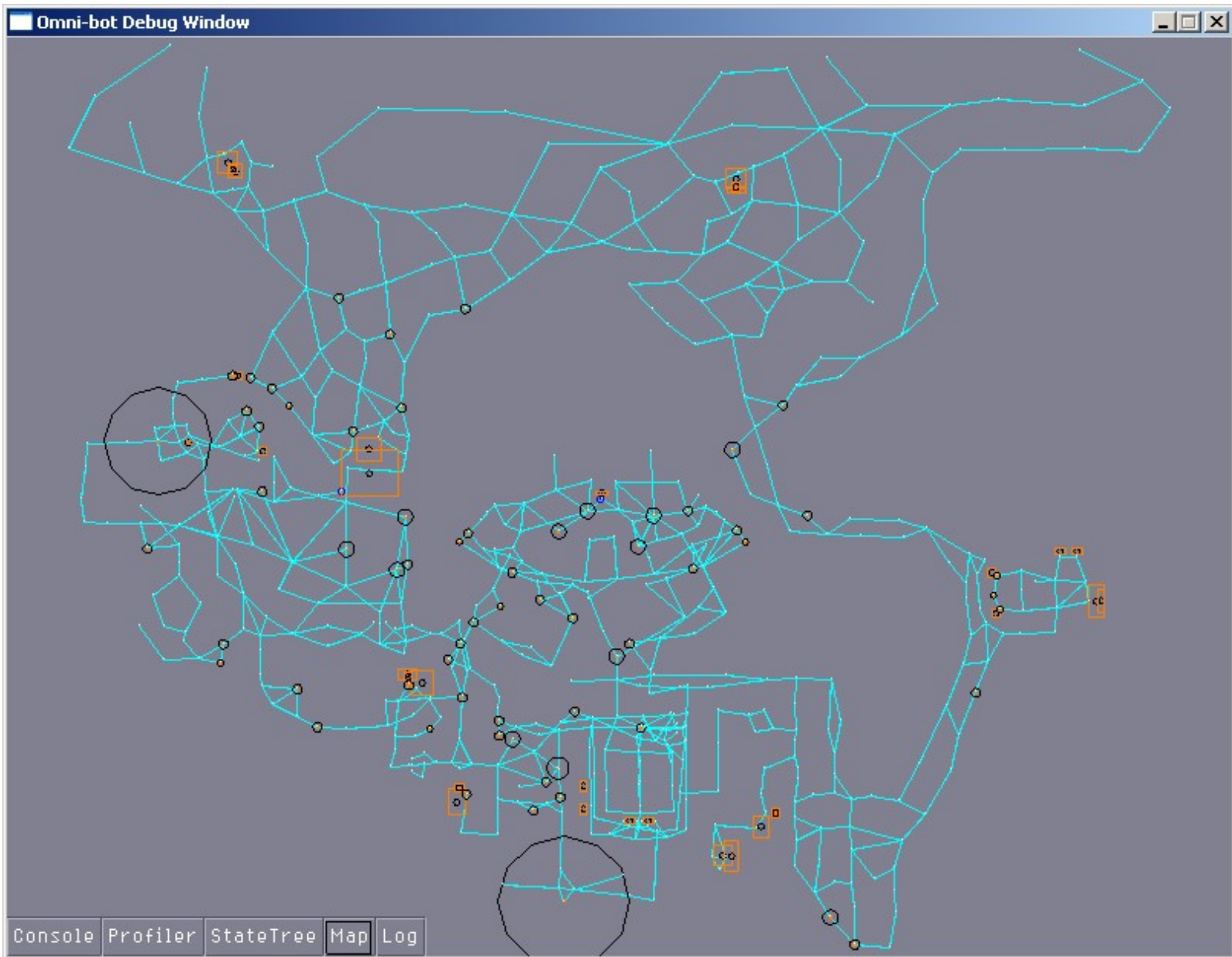
For the scripting users among us, the console window provides an interface directly into the scripting system, and allows you to execute script commands from a simple input window and see outputs such as print or error messages in the large box above.



## Map

The map shows a number of useful things about the bots navigation, goals, pathing, etc.

- Waypoints(and their radius) and links
- Goal bounding boxes
- Current bot paths
- All entities
- Mouse wheel to zoom in/out
- Click drag to scroll
- Also planning to be able to mouse over entities to get additional information



**Profiler**

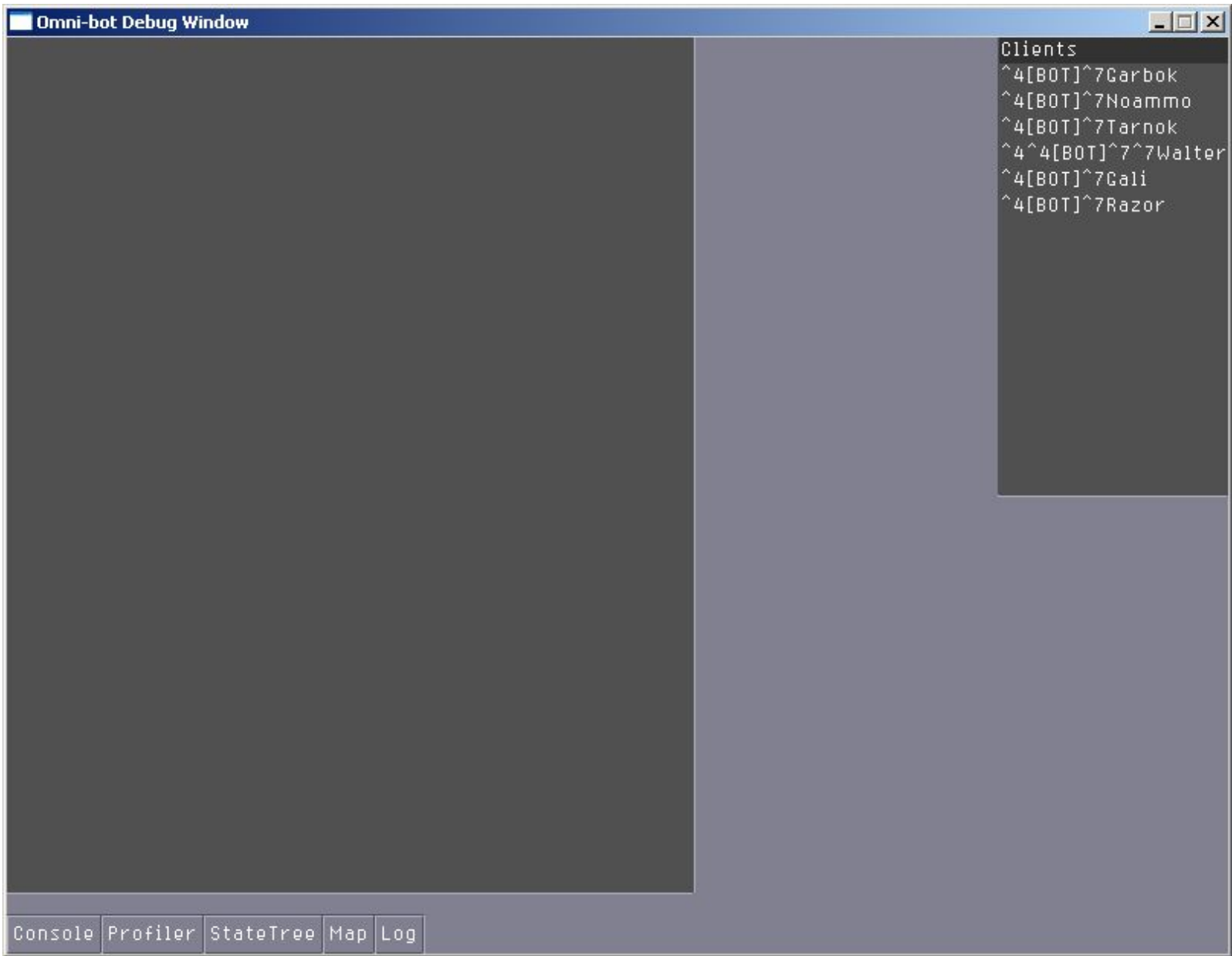
The profiler is also of course making a return. For anyone that has used it before, it allows you to peek around in the bot to examine the performance characteristics

Omni-bot Debug Window			
50.196 ms/frame (fps: 19.92) sort hier - current frame			
zone	self	hier	count
+Client_Update	2.86	3.16	2.00
-RootUpdate	2.86	3.16	2.00
+SensoryMemory	0.00	0.09	1.00
+TargetingSystem	0.00	0.06	2.00
+Aimer	0.00	0.04	2.00
+FindState	0.01	0.01	14.07
+WeaponSystem	0.00	0.01	2.00
+ScriptGoal	0.00	0.00	6.00
+SteeringSystem	0.00	0.00	2.00
SelectBestWeaponNew	0.00	0.00	0.02
FollowPath	0.00	0.00	0.00
TraceLine	0.00	0.00	0.00
RunDijkstra	0.00	0.00	0.00
GetClosestWaypoint	0.00	0.00	0.00
Client_ProcessEventImpl	0.00	0.00	0.00
AttackTarget	0.00	0.00	0.00

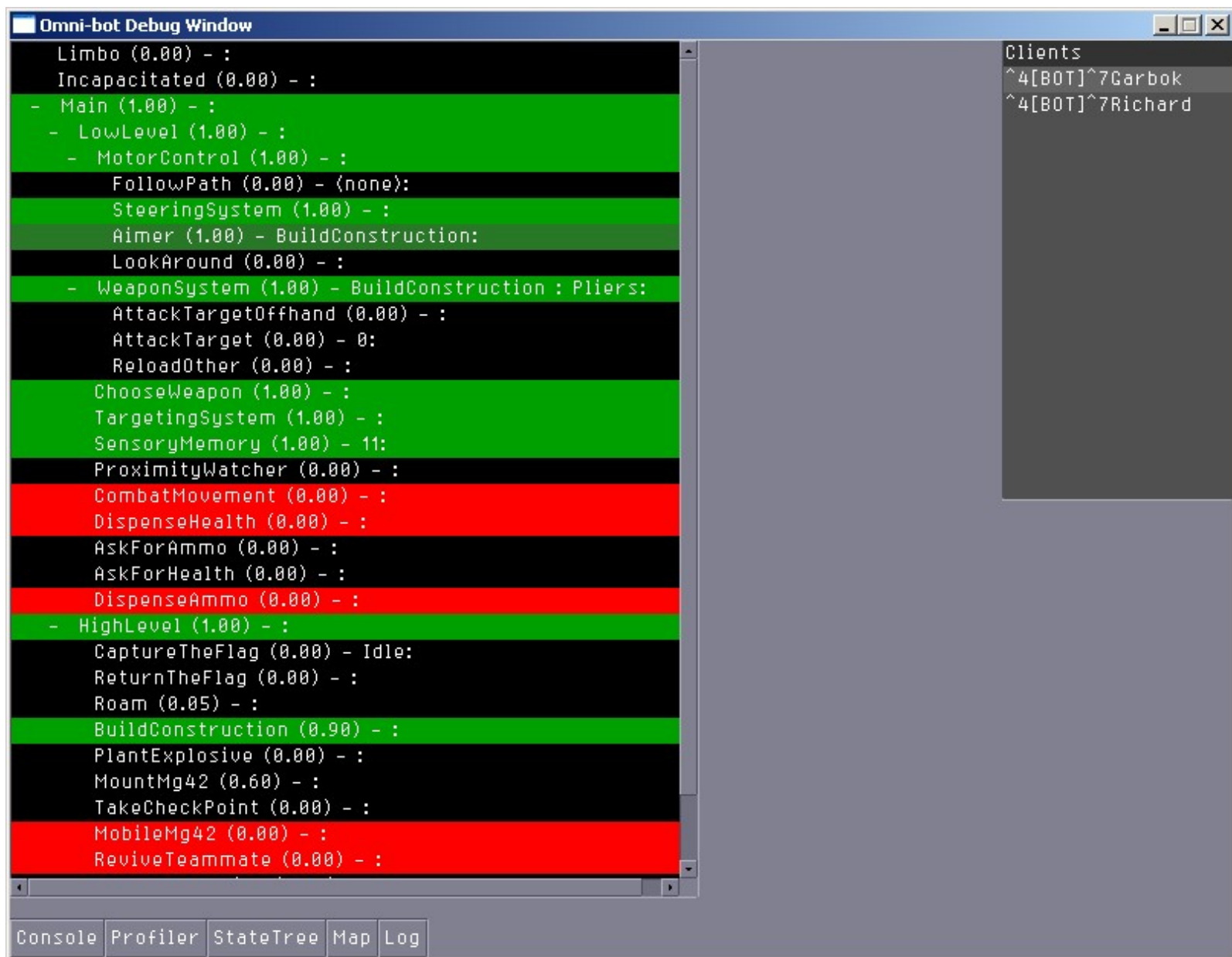
This shows the profiler information at the state machine updating level.

### State Tree

One of the most useful tools available is the state tree. The state tree gives you a complete view of what is going on in the bots 'head'. You can see from the state tree all the behaviors that are running on the bot at any given time, and also can access additional information for specific states here as well. Scripted goals will also show up in this state tree as well.



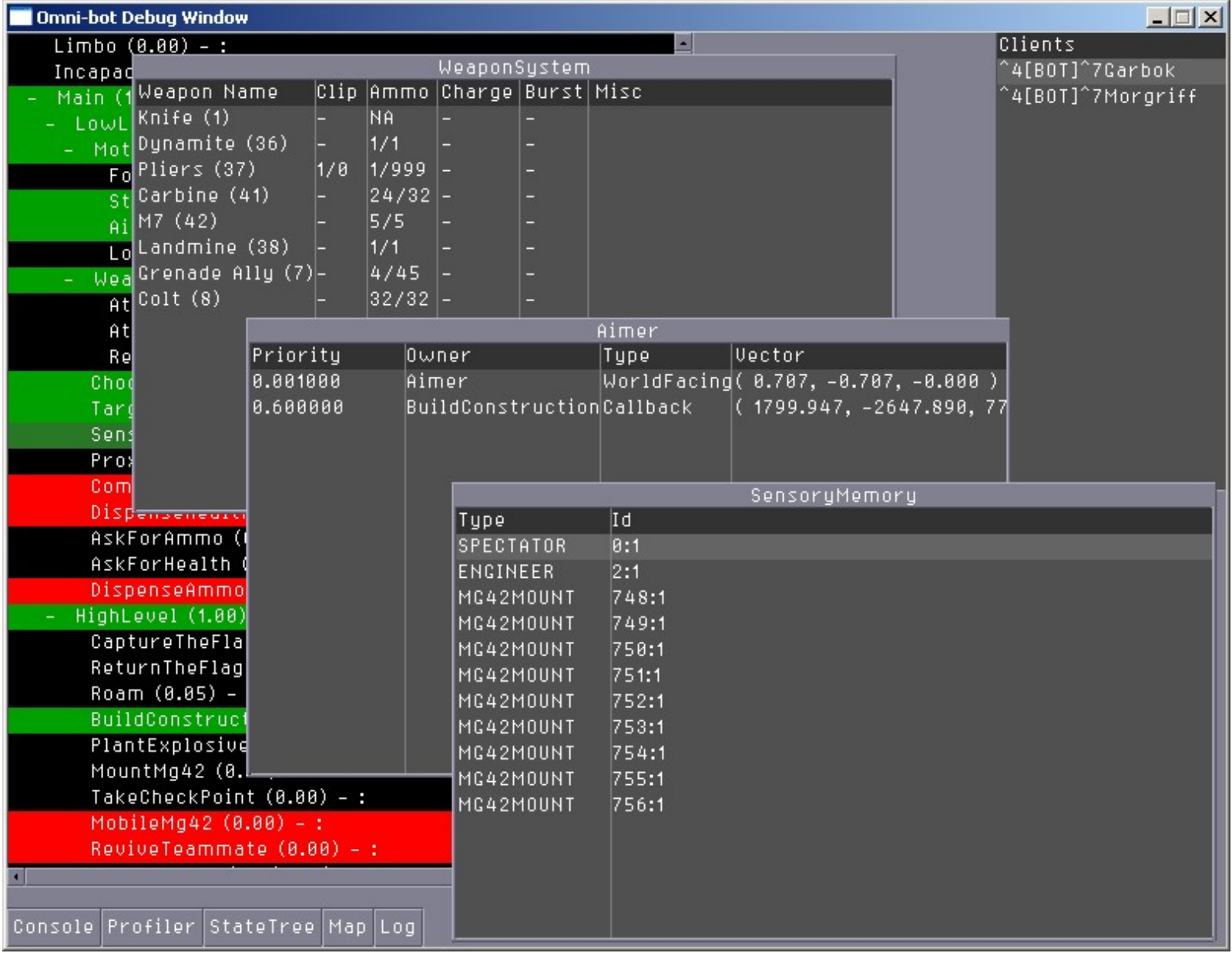
Nothing special about this. This is what you'll see before you actually click on a bot name in the client list to the upper right.



This shows the state tree for an ET bot. Some things to note about the state tree.

- Indentation shows hierarchy.
- Green lines are states that are currently running.
- Red lines are states that can't run because they don't meet a state requirement.
- Left click to show the states extra info, as shown below.
- Middle mouse click on state lines to enable their debug rendering if available.
- Right click on states to expand/collapse their children states.





Many states will have additional information they can display specific to the state. This screen shows the windows for 3 states open, to give you an idea of what information the states are responsible for looking at.

In this case, the weapon system tracks a list of the bots current weapons, and information about it. The SensoryMemory is the bots eyes and ears, and will show the current perception information about all known game entities. The Aimer state is responsible for prioritizing a list of aim requests from other states and preventing multiple states from fighting over control of the bots aim.

The content of these windows will likely change alot. Obviously there's nothing useful being shown in the SensoryMemory window at the moment, and the WeaponSystem window could use some more information shown. These windows are early versions, so they will evolve over time.

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Debug\\_Window](http://omni-bot.com/wiki/index.php?title=Debug_Window)"

This page has been accessed 302 times. This page was last modified 15:24, 12 November 2007.

# About Bots

From Omni-bot Wiki

## Contents

- 1 What is a bot ?
- 2 What are they good for ?
- 3 Do bots cheat ?
- 4 What is Omni-bot?
  - 4.1 Omni-bot Goals
  - 4.2 Omni-bot History
  - 4.3 How Omni-bot works
  - 4.4 What games does Omni-bot support?

## What is a bot ?

Bots are computer controlled opponents, typically designed to play a game in a similar manner as what you might encounter when playing online multiplayer with other players. Although it is very difficult to get bots to actually play like a human would, most bot authors invest considerable time giving the bots the ability to competently play the game, and be a fun opponent.

Anyone that has played FPS(First Person Shooter) games is probably familiar with bots.

Bots normally reside on the game server, meaning it is the game server that actually runs the bots, processes their navigation and AI, and ultimately controls the overall actions of the bots. Client side bots are commonly considered cheats, because they too are computer controlled and are often capable of giving individual clients unfair advantages. Aimbots could be considered a form of client side bot.

## What are they good for ?

Bots can be useful to players or server administrators for several reasons.

- Sometimes, for some people, playing against a bot can be more fun than playing a human. Humans are commonly idiots in multi-player games. They may cheat, complain, camp vehicles/spawns, teamkill, or generally do other things that ruin the enjoyment of other players. A well coded bot will play the game as designed. They don't complain. They don't cheat.

- They can be good practice. Playing against some bots, or even a hoard of bots, can be a good way to practice your death-matching skills, dodging, avoidance, sniping, etc.
- Bots are a good way for a server administrator to attract real players to their server. In a server browser, there are often many hundreds of servers listed. Few people will choose to join an empty server, so bots are often a good way to keep some gameplay going on a server even when no humans are around. Most bots have options that allow bots to be removed from the game as human players join, eventually resulting in a game of all or mostly humans.
- Playtesting maps. As a mapper building a map, it is often a good idea to playtest it. Sometimes it can be difficult to get enough humans together to properly test a map. If the mapper so desires, they can build bot navigation for their map as they develop it and test it with a full game of bots, or humans and bots, to test the balance, fun, and overall flow of the map. Foxbot for TFC was often used in this way.

## Do bots cheat ?

The answer to this varies from bot to bot. To be perfectly honest, technically some of the way bots have to be coded might be considered cheating by some, however most bot authors go through great lengths to reduce cheating to a level where the fun is not ruined.

Some examples of things that might be considered cheating

- Bots 'see' by doing collision tests with lines or volumes. This means that in order to determine if a bot can 'see' another player, a ray is shot from the bots eye to the target. If this ray doesn't collide with anything, such as walls, other players, props, etc, we would consider the object in the bots view. If the ray does hit something we consider the view blocked.

This concept is both a strength and a weakness of bot AI. The strength, and the potential for cheat accusations to come about, are often due to the reliance on collision to determine view. For example, many games have foliage in their outdoor maps. Grass, bushes, trees, etc. Often these sort of things do not have any kind of collision on them, so they don't obstruct the view of a bot at all. To counter this, some games go through extra efforts to simulate obstruction through foliage. Many bots don't make such compensations, so in maps that contain objects that to humans might be considered cover because they block your view with transparent, often the bot is able to see right through them. This also comes into play with game elements such as flashbangs, smoke grenades, tear gas, or other things that might be based around altering the rendering of a human players screen. These sort of effects are often faked with bots, in one way or another.

A weakness of this method, is that it gives the bot a very very rough view of the

environment. At the simplest level, a bot might cast a single ray from his eye to the enemy eye or enemy position to see if it can 'see' them. If the ray hits something, he can't see it, if it doesn't hit anything, he can see it. The problem with relying on a single raycast to make this determination, is that in reality that one spot on the target might be obstructed, while the rest of the target might be viewable. Due to this, sometimes bot authors do several raycasts at different positions on the target, to account for the possibility of partially obstructed targets. This results in a better view system, at the expense of requiring more processing. In extreme cases, a bot might cast a ray to every body part or bone on the target. This would give excellent resolution to the bot, but at a very high increase in cpu cost. Ultimately it depends on how accurate the bot author wants them to be. Most games typically don't notice slight inaccuracies in the resolution of the bots view system, while a slower paced tactical game might benefit more from it.

Also under the category of cheating, it is true that as a server side mod, the bot author technically has full access to where everything and everyone is in the game. It is fully within the capabilities of the bot coder to make the bots fully aware of everyone and everything. As a matter of fact, that would be the easiest way to make a bot. Fortunately, most bot authors don't like their bot to cheat that badly, which is why a bots knowledge of the world is heavily filtered, usually by parameters that mimic the view capabilities of a human player as close as possible, like the field of view, view distance, and whether the view is obstructed, usually by doing ray cast collision checks.

Finally, the subject most often accused of cheating is a bots aiming. It is very easy to make a bot shoot exactly on target every time. Some bots have implemented aiming systems that have resulted in an almost superhuman accuracy. These bots are rarely fun to play, and most bot coders also put a lot of effort into a more human-like aiming system, with properties such as a max turn rate, and adding acceleration/deceleration to aiming, both for smoothness and for less robotic behavior.

In the end, bot AI, and game AI for that matter almost always cheats to some degree. It isn't feasible, both in time and processing power, to make an AI work exactly like a human. The key to good AI is limiting the cheating of the bot to an acceptable level. Some cheating is inevitable, but a good bot will not make it obvious that they are cheating. It's all about having fun, and sometimes the code has to be written to work around the fact that bots don't see a rendered world at all like players, and sometimes have very limited or non existent hearing capabilities too. Ultimately it doesn't matter, as long as the gaming experience is fun.

## **What is Omni-bot?**

Omni-bot is a project with much bigger goals than just being a bot.

From the start of development, the intent was for Omni-bot to be a bot capable of being

easily ported to different engines, and different mods within those engines. Essentially it was to be the first cross game and cross mod bot. It was my desire from the start that the Omni-bot code would primarily be contained seperately from the game, and interact with the game in a standardized manner that could be easily ported to other engines and mods with minimal work. The vast majority of the Omni-bot code is shared between all supported games and mods.

## **Omni-bot Goals**

The Omni-bot goals were pretty ambitious.

- Create a bot that was easily portable to completely different engines and mods.
- Create a highly customizable bot, with data driven properties, and scripting support.
- Release an SDK, so that other mod authors can leverage the design of Omni-bot to easily enable bots in custom mods.

These goals have long since been achieved. Omni-bot supports Enemy Territory, Quake 4, ETF, early unreleased support for HL2:DM, and soon the official bots for Fortress Forever. Development continues, with improvements made to the code that all supported games can take advantage of, as well as plans for supporting more games.

The desired support for many games and many mods is where Omni-bot got its name.

## **Omni-bot History**

The primary Omni-bot developer, Jeremy(me), started out as one of the later bot coders of Foxbot, the popular bot for Valves Team Fortress Classic. As a huge TF fan, and a even bigger fan of bots, coding them became an addiction from early on. Jeremy started Omni-bot in winter of 2004, building up the shared framework and game specific interface code.

The first supported mod was ETF. Due to my love of team fortress, and the fact that ETF was a mod in development, on a mod friendly engine(Enemy Territory/Quake3), it made sense to use it as both a development platform of Omni-bot, but also as the first supported mod while I built up the Omni-bot code. ETF shipped with a very early playable version of Omni-bot. It had limited scripting support, but was capable of playing most of the Team Fortress classes decently. During development of ETF, Marc "Magik" joined on as an Omni-bot developer to develop support for Enemy Territory. Marc was crucial in early development of Enemy Territory support while Jeremy primarily worked on ETF.

After ETF released it's final version, Jeremy moved over to help support Enemy Territory, where a majority of the collective Omni-bot development was focused for quite some time. While Marc is currently kept busy with real life, he has helpfully provided a home for the Omni-bot website on the internet on a fast dedicated server. Hopefully in the future more

free time will become available for him to do some more bot work.

Throughout development of Enemy Territory, Omni-bot has grown leaps and bounds over it's initial release in ETF. The scripting capabilities have been dramatically improved and the overall capabilities of the bot are significantly more than they started. Today, there is enough scripting exposed in Omni-bot for a user to take complete control over the bots, implement custom goals, or just about anything they desire.

Additionally, Omni-bot support has been added to several Enemy Territory modifications, See `Supported_Games` for details.

## How Omni-bot works

A majority of the Omni-bot code is contained and isolated within a dynamic library. A small part of bot related code is compiled into the game. This code is called the interface code, because it's job is to act as the middle man between the engine, and bot. It translates the engine specific data into the standardized messages and implements functions that the bot can call based on common requirements.

Originally, the intent was for a single library(.dll, .so) to contain support for all mods, so every game loaded the same dll to run the bots. Early on, ETF and ET both shared the same dll, which was then simply omnibot.dll It became clear early on that this wasn't a practical way to set it up. Although it was relatively easy to do, lumping all bots into the same dll created several limiting factors that would have been hell to work with.

- The dll would eventually become very large. Even though much of the code is shared in the bot dll, custom game modes, weapons, and goals are implemented as part of the support for a particular mod.
- It would require each and every release to be an update for every supported mod. It was clear very quickly this wasn't going to happen. One person doesn't have the time to develop several mods to a releasable state at once, and even with multiple developers, it would be very difficult to synchronize releases.

Ultimately it was decided to split dlls per mod, although it is still entirely possible to add support for several mods and/or games to a single dll. Now, bots for a given game can be developed and released independent of each other, which simplifies the process. It also allows a distribution method to differ between mods. The desire at first was that all Omni-bot would reside in a central location, such as `C:\Program Files\Omni-bot` This is mostly how it works, but it is often desirable to be packaged another way, for dedicated servers, or for some mods, like ETF and Fortress Forever, where the bot must come with the mod and more or less be a part of it.

That's more or less a rundown of the goals and history of Omni-bot. Presently, development

moves ahead at full speed, with some interesting developments in the works. Follow the development updates for more details. The community has grown pretty large and users are often submitting navigation and scripts for the bot to share with others.

All feedback is taken into consideration. If you have ideas for improvement, or ideas that would simplify or expose more flexibility in the scripting, feel free to share them. Much of the scripting systems power got the way it is now from feedback from people that were using it. Many optimizations, many bug fixes, many functions, and many features were a direct result of user requests and questions about how to do something in particular.

## **What games does Omni-bot support?**

See [Supported\\_Games](#) for details.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=About\\_Bots](http://omni-bot.com/wiki/index.php?title=About_Bots)"

This page has been accessed 112 times. This page was last modified 16:18, 14 January 2008.

# Map Scripting Enemy Territory

(Redirected from Enemy Territory Map Scripting)

<a href="#">ET Main Page</a>	<b>Map Scripting</b>	<a href="#">Archives</a>
------------------------------	----------------------	--------------------------

## Contents

- 1 Standard Format
  - 1.1 Map Table
  - 1.2 OnMapLoad
  - 1.3 OnBotJoin
  - 1.4 Triggers
    - 1.4.1 Supported Triggers
- 2 Native Functions
  - 2.1 ChangeSpawnPoint
  - 2.2 GetGameTimeLeft
  - 2.3 GetGameType
  - 2.4 GetReinforceTime
  - 2.5 MaxViewDistance
  - 2.6 SetAvailableMapGoals
  - 2.7 SetGoalPriority
  - 2.8 SetMapGoalProperties
  - 2.9 TargetBreakableDist
- 3 Utility Functions
  - 3.1 ETUtilities
    - 3.1.1 ETUtil.ChangeClass
    - 3.1.2 ETUtil.ClearMainGoals
    - 3.1.3 ETUtil.ClearSecondaryGoals
    - 3.1.4 ETUtil.CountClass
    - 3.1.5 ETUtil.CountTeam
    - 3.1.6 ETUtil.DisableGoal
    - 3.1.7 ETUtil.EnableGoal
    - 3.1.8 ETUtil.LimitToClass
    - 3.1.9 ETUtil.NoSnipe
    - 3.1.10 ETUtil.RandomSpawn
    - 3.1.11 ETUtil.SelectWeapon
    - 3.1.12 ETUtil.SetPrimaryGoals
    - 3.1.13 ETUtil.ShowActiveGoals
    - 3.1.14 ETUtil.StopSniping
    - 3.1.15 ETUtil.SwitchWeapon
  - 3.2 Utilities
    - 3.2.1 Util.AddInvVehicle
    - 3.2.2 Util.AliveCount
    - 3.2.3 Util.DisableGroup
    - 3.2.4 Util.DistanceView
    - 3.2.5 Util.EnableGroup
    - 3.2.6 Util.GetGroup
    - 3.2.7 Util.OnTriggerPosition
    - 3.2.8 Util.RemoveGoal
    - 3.2.9 Util.SetGoalOffset
    - 3.2.10 Util.SetGoalPosition



- 3.2.11 Util.SetGroup
- 3.2.12 Util.ShowGroup
- 3.2.13 Util.SetMaxUsersInProgress
- 3.2.14 Util.SetPositionGoal
- 3.2.15 Util.ShowGoalInfo
- 3.2.16 Util.ShowGoalName
- 3.2.17 Util.ShowGoalOffset
- 3.2.18 Util.AddUsePoint
- 3.2.19 Util.AddUseWp
- 4 Conditionals
  - 4.1 Case Study
- 5 Scripting Tools
  - 5.1 makegm
- 6 Debugging Tools
  - 6.1 debugbot
  - 6.2 ScriptDebug
  - 6.3 GameMonkey interpreter
  - 6.4 print
  - 6.5 show\_goals
  - 6.6 show\_goalroutes
  - 6.7 draw\_goals
  - 6.8 draw\_goalroutes

## Standard Format

All official map scripts share a common layout. Having a standard for map scripts has several benefits; ease of documentation, debugging, and implementation of scripted goals are among those benefits. A typical ET map script will consist of one Map table and two functions that are automatically called by Omni-Bot:

```
global Map =
{
};

global OnMapLoad = function()
{
};

global OnBotJoin = function( bot )
{
};
```

Using makemapgm will ensure that a map script is in a standard format.

## Map Table

The Map Table has become an important part of an ET map script as some scripted goals rely on its existence. It also provides a 'safe' place to store map specific information. Typically it will consist of map variables, triggers / functions, and in some cases additional tables:

```
global Map =
```

```
{
    SomeGoal = "BUILD_some_construct",
    someVar = true,
    someTable =
    {
        someVar = false,
    },

    some_trigger = function(trigger)
    {
        print("some_trigger");
    },
};
```

It is important to note that when you add to a table you use a comma and not a semi-colon. Calling functions or referencing variables within the example table is done like this:

```
Map.someVar = false; //changes the value of someVar to false
print(Map.SomeGoal); //will print BUILD_some_construct
Map.someTable.someVar = true; //changes the value of someTable.someVar to true
```

The key thing to note is that you need to put Map. in front of whatever you want to call or reference within the Map table.

## OnMapLoad

The OnMapLoad function is used to initialize settings for the map. It is automatically called by Omni-Bot whenever a map loads (i.e. /map\_restart). Setting up triggers, goal availability, and goal bias' are the most common jobs performed by this function:

```
global OnMapLoad = function()
{
    OnTrigger( "Some Team did Something!", Map.some_trigger );
    SetAvailableMapGoals( TEAM.AXIS, false, Map.SomeGoal );
    SetGoalPriority( Map.SomeGoal, 1.1 );
};
```

## OnBotJoin

The OnBotJoin function is automatically called by Omni-Bot when a bot joins. Typical usage for OnBotJoin is for setting specific properties on the bot that you want to be map specific. View distance and breakable object distance are the two most commonly used:

```
global OnBotJoin = function( bot )
{
    bot.MaxViewDistance = 2500;
    bot.TargetBreakableDist = 150.0;
};
```

Note that it is not necessary to loop through the BotTable every time this is called as this function is called each time a bot joins.

## Triggers

A trigger is an event that is recognized by Omni-Bot in game. With map scripting, triggers can be set up to perform operations when a map event occurs. There are two steps to setting up triggers; defining the trigger and setting up the trigger function.

Triggers are most commonly defined in OnMapLoad using the OnTrigger function:

```
global OnMapLoad = function()  
{  
    OnTrigger("string", function);  
};
```

The "string" parameter is usually found using the wm\_announce messages seen during the game. This must be matched exactly including capitalization, punctuation, and color codes (if any). To see the wm\_announce message requires the waypointer to either play through the map and perform the objectives or look through the map script that is included inside a map's pk3 file.

If the wm\_announce messages appear in non-standard color, the latter method is often more convenient. The map's native script (not to be confused with the Omni-bot map script) is usually the file <mapname>.script in the /maps/ folder inside the pk3 file. Open the pk3 file with some archive utility, e.g. IZArc.

Another method of finding the string parameter is using the command **/bot debugtriggers**. This command will output to console everytime a recognized event occurs. The syntax will be similar to:

```
<---> Trigger: TagName: The Tank has been repaired! Action: announce Entity: 0x7016dd8c  
Activator: 0
```

The string just after TagName: is what the OnTrigger function will expect as the string parameter. For additional information about debugtriggers see the Omni-bot\_Command\_Reference#debugtriggers page. This method is required for maps that may not have wm\_announce messages that correspond to recognized map events.

Once you've collected the triggers you need, the contents of the console can be written to a plain text file by using the command /condump mytextfile. The file will be located in the omnibot folder under the game installation folder. By copying the trigger names from that file instead of typing them, the risk of typing errors is greatly reduced.

The function parameter of the OnTrigger function is the name of a function that the waypointer creates for the particular trigger. These can be named anything the waypointer wants, but should be somewhat intuitive.

Once the string parameter has been identified and a function name has been determined, the trigger definition will look something like this:

```
global OnMapLoad = function()  
{  
    OnTrigger("The Tank has been repaired!", Map.tank_repaired );  
};
```

The last part of the setup is to create the trigger function that will contain code for the game to process each time the event occurs. In all official map scripts, this is done inside the Map table:

```
global Map =  
{
```

```

tank_repaired = function( trigger )
{
    //things to do when this event occurs
    print("tank_repaired");
},
};

```

It is important to note is that the function is inside the map table and should have a comma outside the closing bracket and not a semi-colon. The print statement inside the trigger function is not necessary, but is a good way to test that the trigger is working. As an additional service to the users, you can comment these print statements out when you're done testing, so their console isn't flooded with information that doesn't mean much to them.

## Supported Triggers

Most events in ET will fall into the following Omni-Bot recognized categories:

- allied\_complete
- allied\_default
- allied\_failed
- announce
- announce\_icon
- axis\_complete
- axis\_default
- axis\_failed
- closed
- defused
- dynamited
- exploded
- mover\_goto x y z
- opened
- repair\_mg42
- returned
- round end
- stolen
- team\_announce
- thirty second
- two minute

If you are unsure if the event you want to set up a trigger for falls into one of these categories, use `/bot debugtriggers` in game.

**NOTE: For the returned trigger (i.e. documents returned) use `/bot debugtriggers` output for when the flag is returned by expiring (no player returns it). If the regular `wm_announce` message is used, the event that occurs when the flag expires will not be used. It is recommended to manually steal the flag, `/kill` and wait for them to be automatically returned. An example can be found in `radar.gm`. Most of the time it will be something like "Flag returned flag!"**

## Native Functions

This section consists of Omni-Bot functions specifically designed for use in map scripts.

### ChangeSpawnPoint

syntax: `bot.ChangeSpawnPoint(int spawnptid);`  
example: `bot.ChangeSpawnPoint(1);`

note: the number of the spawn point is map dependant and may require either searching for spawn selection configs or testing the numbers.

## GetGameTimeLeft

syntax: `GetGameTimeLeft();`  
returns: Time left in the game in seconds

example: can be used in maps where you may want the bots to focus on major objectives if there isn't much time left.

## GetGameType

syntax: `GetGameType();`  
returns: The current game type

## GetReinforceTime

syntax: `bot.GetReinforceTime();`  
returns: Time left before the bots next spawn

example: can be used in maps where you may want to exec a command if the bot is close to a new respawn.

## MaxViewDistance

syntax: `bot.MaxViewDistance = <distance>;`  
example: `bot.MaxViewDistance = 2500;`

note: typically set in OnBotJoin as this sets the property individually

## SetAvailableMapGoals

syntax: `SetAvailableMapGoals( Team, true / false, goalname );`  
example: `SetAvailableMapGoals( TEAM.AXIS, true, "MAP_FLAG_someflag" );`  
example: `SetAvailableMapGoals( TEAM.ALLIES, false, Map.Flag );`  
example: `SetAvailableMapGoals( TEAM.AXIS, false, "DEFEND.*" );`

note: the goalname parameter supports expressions

note: this command is the simpler alternative to getting a goal, then setting the availabilty.  
it's not necessary to do something like this:

```
somevar = GetGoal( "SOMEGOAL" )
if (somevar)
{
    somevar.SetAvailable( TEAM.ALLIES, true );
}
```

## SetGoalPriority

```
syntax: SetGoalPriority( goalname, priority, team, class, <optional persistent> );  
example: SetGoalPriority( Map.Flag, 1.0, 0, 0 ); //all teams and all classes  
example: SetGoalPriority( "DEFUSE_somedyno.*", 0.0, TEAM.AXIS, CLASS.ENGINEER, true ); //the  
persistent parameter is used for dynamic goals that may not have been created yet
```

## SetMapGoalProperties

```
syntax: SetMapGoalProperties( Goal, table )  
example: SetMapGoalProperties( "ATTACK_.*", {mincamptime=15, maxcamptime=30} );  
  
note: see OnMapLoad in goldrush.gm for example usage
```

## TargetBreakableDist

```
syntax: bot.TargetBreakableDist = <distance>;  
example: bot.TargetBreakableDist = 100;  
  
note: typically set in OnBotJoin as this sets the property individually  
note: used to allow bots to target breakables like windows. should be set to a relatively low  
number.
```

## Utility Functions

The functions listed in this section are custom functions created to support specific scenarios in game. They are located in ~/omni-bot/et/scripts and ~/omni-bot/global\_scripts respectively.

### ETUtilities

ET utility functions are located in the ~/omni-bot/et/scripts/et\_utilities.gm file.

### ETUtil.ChangeClass

```
syntax: ETUtil.ChangeClass( team, originalclass, newclass, revert, maxbots )  
example: ETUtil.ChangeClass( TEAM.ALLIES, CLASS.SOLDIER, CLASS.COVERTOPS, false, 1 );  
  
usage: used in braundorf_b4 to change one bot to covert ops for satcheling the side gate.  
Once satcheled, the function is called again with the revert flag set to true to  
have the bot change back to it's original class.
```

### ETUtil.ClearMainGoals

```
syntax: ETUtil.ClearMainGoals();  
usage: This function will deactivate all main goals for both teams.  
  
goals deactivated: AMMOCAB, CHECKPOINT, HEALTHCAB, BUILD, PLANT, FLAG, MOUNTMG42, MOVER
```

### ETUtil.ClearSecondaryGoals

```
syntax: ETUtil.ClearSecondaryGoals();
```

usage: This function will deactivate all secondary goals for both teams.

goals deactivated: AMMO, HEALTH, ARTILLERY, MOBILEMG42, REPAIRMG42, PLANTMINE

## ETUtil.CountClass

syntax: `ETUtil.CountClass( team, class )`

example: `ETUtil.CountClass( TEAM.ALLIES, CLASS.ENGINEER );`

usage: can be used to determine if a team has enough of a critical class for the map

## ETUtil.CountTeam

syntax: `ETUtil.CountTeam( team )`

example: `ETUtil.CountClass( TEAM.ALLIES );`

usage: used to count the number of bots on a give team. used in maps like et\_ice  
in a conditional statement for setting max users attacking / defending.

## ETUtil.DisableGoal

syntax: `ETUtil.DisableGoal(goalname, <optional true>);`

example: `ETUtil.DisableGoal("FLAG_someflag");`

usage: disables the goal for both teams. the optional true parameter is used to disable all goals except for ROUTE goals.

## ETUtil.EnableGoal

syntax: `ETUtil.EnableGoal(goalname);`

example: `ETUtil.EnableGoal("FLAG_someflag");`

usage: enables the goal for both teams.

## ETUtil.LimitToClass

syntax: `ETUtil.LimitToClass(goalname, team, class1, class2, class3);`

usage: used to limit specific goals to a specific class or classes

example: `ETUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER) //only soldiers on the allied team`

example: `ETUtil.LimitToClass("CHECKPOINT.*", TEAM.ALLIES, CLASS.SOLDIER, CLASS.MEDIC) //only soldiers and medics`

## ETUtil.NoSnipe

syntax: `ETUtil.NoSnipe(bot);`

usage: covert ops bots will not choose a garand or k43

note: this is typically called in OnBotJoin

## ETUtil.RandomSpawn

syntax: `ETUtil.RandomSpawn(team, spawnpoint);`

usage: used to have bots randomly choose to spawn at the given spawnpoint

## ETUtil.SelectWeapon

syntax: `ETUtil.SelectWeapon(bot, weapon);`

usage: the given bot will switch to the given weapon if it is the correct class for the weapon

note: typically used in OnBotJoin to have soldiers choose a specific weapon

example: `ETUtil.SelectWeapon(bot, WEAPON.PANZERFAUST);`

## ETUtil.SetPrimaryGoals

syntax: `ETUtil.SetPrimaryGoals(priority);`

usage: used for setting priorities of common goals.

note: this is typically called in OnMapLoad and effects the following goals (in order of priority):

CAPPOINT FLAGRETURN PLANT CHECKPOINT FLAG

## ETUtil.ShowActiveGoals

syntax: `ETUtil.ShowActiveGoals();`

usage: shows active goals for both teams

note: should only be used for debugging

## ETUtil.StopSniping

syntax: `ETUtil.StopSniping();`

usage: all bots currently using a mauser will switch to a different weapon

note: this is typically called inside trigger functions

## ETUtil.SwitchWeapon

syntax: `ETUtil.SwitchWeapon(weapon);`

usage: all qualifying bots will switch to the given weapon

example: `ETUtil.SwitchWeapon(WEAPON.PANZERFAUST); //all soldiers will switch to panzer`

note: typically used in trigger functions

## Utilities

Utility functions are located in the `~/omni-bot/global_scripts/utilities.gm` file.

## Util.AddInvVehicle

syntax: `Util.AddInvVehicle( goalname )`

usage: adds a `vehicle` to the Map.InvVehicle `table`



used for script goals in cases where the vehicle is invulnerable, but shows as "dead"

## Util.AliveCount

syntax: `Util.AliveCount( team, class )`

usage: returns the number of bots alive on a team with a given class

## Util.DisableGroup

syntax: `Util.DisableGroup(groupname, team);`

example: `Util.DisableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to disable a set of goals in the given group for a given team

## Util.DistanceView

syntax: `Util.DistanceView(<optional off|0>);`

usage: `/bot dist`

`/bot dist off`

`/bot dist 0`

note: used for determining bot.MaxViewDistance settings. aim at a position as far as you can see and type `/bot dist`

## Util.EnableGroup

syntax: `Util.EnableGroup(groupname, team);`

example: `Util.EnableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to enable a set of goals in the given group for a given team

## Util.GetGroup

syntax: `Util.GetGroup(groupname);`

example: `Util.GetGroup("somegroupname");`

usage: this function will return a table of goals belonging to the given group name

## Util.OnTriggerPosition

syntax: `Util.OnTriggerPosition( goalname, wpname, tolerance, wpfunction )`

example: `Util.OnTriggerPosition( Map.Mover_train1, "depotyard", 200.0, Map.tug_depotyard );`

note: used for setting up positional triggers for movers.

## Util.RemoveGoal

syntax: `Util.RemoveGoal( goalname );`

example: `Util.RemoveGoal( "MOVER_truck" );`

note: this command will remove the goal from the map goal table

## Util.SetGoalOffset

syntax: `Util.SetGoalOffset( x, y, z, GoalName );`

example: `Util.SetGoalOffset( 0, 0, 30, "BUILD_construct" )`

usage: the x y and z parameters are added to the origin of the goal to move the location of where the bots will look for the goal.

## Util.SetGoalPosition

syntax: `Util.SetGoalPosition( x, y, z, GoalName );`

example: `Util.SetGoalPosition( 4534, 2168, -199, "BUILD_construct" )`

usage: used to give the goal a new origin.

note: using /devmap to load the map and issuing the /viewpos command in console will give your origin in the map

## Util.SetGroup

syntax: `Util.SetGroup(goalname, groupname);`

example: `Util.SetGroup("somegoalname", "somegroupname");`

usage: this function will add a given goal to a given groupname

## Util.ShowGroup

syntax: `Util.ShowGroup(groupname);`

example: `Util.ShowGroup("somegroupname");`

usage: this function will list all goals in the given group in the console.

## Util.SetMaxUsersInProgress

syntax: `Util.SetMaxUsersInProgress( Users, GoalNames );`

example: `Util.SetMaxUsersInProgress( 15, "CHECKPOINT.*" );`

usage: used to set the maximum number of bots going for a particular goal(s).

## Util.SetPositionGoal

syntax: `Util.SetPositionGoal( goalname1, goalname2 );`

example: `Util.SetPositionGoal( Map.Build_tank_construct, Map.Mover_tank );`

usage: sets the origin of one goal to match the origin of another. useful for centering construct goals on movers

## Util.ShowGoalInfo

syntax: `Util.ShowGoalInfo(goalname);`

command: `/bot sgi 'goalname'`

usage: used to print a goals current information in console including the goals health, status of the DEAD flag,  
and the goals position

## Util.ShowGoalName

syntax: `Util.ShowGoalName(radius, showOffset);`  
command: `/bot sgn <radius> <optional true>`

usage: issue the command `/bot sgn` to find the goalname of any goals within 100 units.  
Optionally expand the radius and  
find your current offset from the goal with `/bot sgn 500 true`

## Util.ShowGoalOffset

syntax: `Util.ShowGoalOffset(goalname);`  
command: `/bot sgo 'somegoalname'`

usage: used to determine a players current offset from a goal. the values given in console can be used for goals that  
require positional offsets

## Util.AddUsePoint

syntax: `Util.AddUsePoint(goalname, position);`  
example: `Util.AddUsePoint( "somegoalname", Vector3(123, 456, 789) );`

usage: used to add a Use Point for a goal at a given position.

## Util.AddUseWp

syntax: `Util.AddUseWp(goalname, waypointname);`  
example: `Util.AddUseWp( "somegoalname", "somewaypointname" );`

usage: used to add a Use Point for a goal at a given waypoints position.

## Conditionals

Enemy Territory is a complex game type. Maps are not linear and events do not necessarily happen in a specific order. Conditional statements are necessary in some cases to ensure that availability of goals is consistent with the waypoints idea of a map flow. A conditional statement begins with the 'if' function. In script, it is used to check given paramaters before executing some code:

```
if ( a == b )  
{  
    //do something  
}
```

In this example it checks if something is equal to something else before executing any code in the following brackets. If something does not equal something else, anything inside the brackets will be ignored. This conditional can be extended to make something happen if the first conditional was not met with the 'else'

command:

```
if ( a == b )
{
    //do something
}
else
{
    //do something else
}
```

And finally, this can be extended even further using the 'else if' command:

```
if ( a == b )
{
    //do something
}
else if ( a == c )
{
    //do something else
}
else
{
    //do something else
}
```

Conditionals can get more complex by adding additional checks. The following example checks if both are true before executing the code in the example:

```
if ( a == b && c == d )
{
    //do something
}
```

If a check is dependant on one of many checks being true, the 'or' operator is used:

```
if ( a == b || c == d )
{
    //do something
}
```

To test if something is true or false, the syntax used is:

```
if ( !something )
{
    //do something
}
```

The apostrophe in front of the parameter checks if something is false while a `!=` checks if something doesn't equal something else:

```
if ( a != b )
{
```

```
    a = b;
}
```

Checking if a value is greater or lesser than another value is done as follows:

```
if ( a > b || c < d )
{
    //do something
}
```

In ET maps where the map flow is not pre-determined, it is important to add conditionals to some triggers in which goals are activated. While the bots order of objective completion can be set by the waypointer, the waypointer can not predict when a human player may complete a particular objective. So if goals are activated with a specific order in mind and a human player activates a goal out of turn, you may have goals available to bots earlier or later than intended; potentially breaking the gameplay.

## Case Study

**Scenario:** Map is Fueldump and goals are being activated or deactivated based on the Main Bridge status. When the bridge is built, attack and defend goals are activated and when it is destroyed, goals are deactivated. To add to the complexity, some of the same goals are activated / deactivated based on the status of the footbridge.

The first thing that needs to be done is to create some variables for use in the conditional statements. Official map scripts add these variables to the Map table:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,
};
```

The next step is to set these variables in the appropriate triggers:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built
    },
}
```

```

},

bridge_Destroyed = function( trigger )
{
    Map.BridgeStatus = 0; //not built
},

};

```

Now that we have status' updating correctly, conditional statements can be added to help ensure goals are active at the appropriate time:

```

global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built
        SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built

        //if both bridges destroyed, shift the defense back
        if ( Map.BridgeStatus == 0 )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
        }
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built
        SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
        SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Bridge.*" );
    },

    bridge_Destroyed = function( trigger )
    {
        Map.BridgeStatus = 0; //not built

        //if both bridges destroyed, shift the defense back
        if ( Map.FootBridgeStatus == 0 )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
        }
    },

};

```

All looks good at this point. At the beginning of the map, the allied bots will be shifting correctly based on both bridges status. There is a potential problem here though. What happens if someone builds or destroys a bridge after the tank is through the tunnel? The goals back by the bridge will be activated again. The solution is to pick a major event to check against and add that to our conditionals. In this case we will check the status of the Tunnel door before activating those goals:

```
global Map =
{
    //conditional variables
    BridgeStatus = 0, //not built
    FootBridgeStatus = 0,
    Tunnel_Doors = true, //doors intact

    tunneldoors_Destroyed = function( trigger )
    {
        Map.Tunnel_Doors = false;
    },

    footbridge_Built = function( trigger )
    {
        Map.FootBridgeStatus = 1; //built

        if ( Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
        }
    },

    footbridge_Destroyed = function( trigger )
    {
        Map.FootBridgeStatus = 0; //not built

        //if both bridges destroyed and doors intact, shift the defense back
        if ( Map.BridgeStatus == 0 && Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge_.*" );
        }
    },

    bridge_Built = function( trigger )
    {
        Map.BridgeStatus = 1; //built

        if ( Map.Tunnel_Doors )
        {
            SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Tunnel_Doors.*" );
            SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Bridge_.*" );
        }
    },

    bridge_Destroyed = function( trigger )
    {
```

```

Map.BridgeStatus = 0; //not built

//if both bridges destroyed and doors intact, shift the defense back
if ( Map.FootBridgeStatus == 0 && Map.Tunnel_Doors )
{
    SetAvailableMapGoals( TEAM.ALLIES, false, "ATTACK_Tunnel_Doors.*" );
    SetAvailableMapGoals( TEAM.ALLIES, true, "ATTACK_Bridge.*" );
}
},
};

```

For a complete example of conditional usage, look through any of the map scripts released with version 0.65.

## Scripting Tools

### makegm

makemapgm is a command that autogenerates a skeleton map script. Usage of this command will ensure a standard format and save time. Using the command is fairly straight forward:

Step 1: Load the map you want a script for

Step 2: Type /bot makegm in console. It will display a message if executed correctly

Step 3: Copy the mapname.gm file from ~/omni-bot/et/usr to ~/omni-bot/et/nav, Be sure to have backed up any mapname.gm file you may have been working on that exists in the nav folder.

The OnTriggers in OnMapLoad will need to be modified as makemap.gm does not find the information needed. Simply edit the "MISSING STRING" parameter of the OnTrigger function and the triggers should be working correctly.

## Debugging Tools

### debugbot

debugbot is a valuable tool to use if bots aren't behaving as expected. It will give success and failure messages in console for short and long term goals.

```

syntax: /bot debugbot all goals
or: /bot debugbot <bot-name> goals

```

It is recommended to have only one or two bots connected when issuing this command with the all parameter as it will give debug output for each bot.

### ScriptDebug

Script debugging is crucial as a high percentage of errors are syntax related. There are two ways to enable script debugging:

- /bot script\_debug 1
- [EnableScriptDebug\(true\)](#);

EnableScriptDebug(true); can be placed in et\_autoexec.gm while /bot script\_debug 1 will need to be executed in



console each time the map loads. If an error in script occurs while script debugging is enabled, the error will be listed in console in red text with the line number of the error.

note: in the omnibot.log, script errors will be written whether script debugging is enabled or not.

## GameMonkey interpreter

The GameMonkey download available at [www.somedude.net/gamemonkey/](http://www.somedude.net/gamemonkey/) contains an interpreter for gm files (gme.exe) that can be used as a syntax checker. With this, you won't have to start the game only to find you forgot a closing bracket or a semicolon.

Ideally, use this with an editor that can run command line applications and capture their output, such as SciTE or PSPad.

## print

print is a low level means of debugging a script. It can be used in cases where you want to test if a trigger is working correctly or if the map script has loaded:

```
global OnMapLoad = function()  
{  
    print("OnMapLoad");  
}
```

The print statement can also be used in-game via script\_run if you want to inspect the value of a variable:  
/bot script\_run "print(variable)"

## show\_goals

syntax: /bot show\_goals <optional>

usage: the optional parameter can be used to display one or several goals.

note: the optional parameter supports expressions. /bot show\_goals DEFEND.\* will give output for all DEFEND goals.

show\_goals can be useful to test whether or not goals are available when they are expected to be and that they have the expected bias setting. The output from show goals is similar to:

```
PLANT_sometarget -> 1000 serial 3 bias 1.00
```

The number just to the right of the arrow represents teams and status. For ET, only the first two matter. The first number is for Axis and the second is for Allies. 1 means it's available for that team while 0 means it is unavailable. In this case, the dynamite goal is currently available for the Axis team.

## show\_goalroutes

syntax: /bot show\_goalroutes <optional>

usage: used to list routes used for a give goal or goals

## draw\_goals

syntax: /bot draw\_goals [on|1|off|0] <optional>

usage: the optional parameter can be used to display one or several goals.

Note: the optional parameter supports regular expressions. /bot draw\_goals .\*BUILD.\* will draw BUILD goals.

This will draw the location of goals to your screen. Useful to see where bots "think" an object is located.

## **draw\_goalroutes**

syntax: draw\_goals on/off <optional goal name expression>

usage: used to draw the given goal(s) routes

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Map\\_Scripting\\_Enemy\\_Territory](http://omni-bot.com/wiki/index.php?title=Map_Scripting_Enemy_Territory)"

This page has been accessed 1,429 times. This page was last modified 21:31, 26 June 2008.

# Advanced Scripting Guide

From Omni-bot Wiki

## Contents

- 1 Basics
  - 1.1 Variable Type
  - 1.2 Operators
  - 1.3 Variable Scope
  - 1.4 Functions
  - 1.5 Loops
- 2 Omni-Bot Scripting
  - 2.1 Bot Functions
    - 2.1.1 General Functions
    - 2.1.2 Movement functions
    - 2.1.3 Weapon functions
      - 2.1.3.1 For all games
      - 2.1.3.2 ET-specific
  - 2.2 GM Functions
  - 2.3 Omni-Bot Event Callback Functions
    - 2.3.1 Spawn Function
    - 2.3.2 Voice Function
    - 2.3.3 Death Function
    - 2.3.4 Killedsomeone Function
    - 2.3.5 Feelpain Function

## Basics

You can skip this section if you already know GameMonkey or any other language that is similar to it.

### Variable Type

```
//a local variable that stores integer 1
var1 = 1;
//a local variable that stores string "1"
var1 = "1";
//a local empty table (similar to array)
table1 = table();
//a local non-empty table
table1 = {
    Id1 = "A",
    Id2 = "B"
};
//starting index of GM script is 0, unlike most of the 'mainstream' scripting languages'
starting index
print(table1[0]);
//output: A
print(table1[1]);
//output: B
```

```

print(table1[2]);
//output:  (null)

//using 'associated index' to retrieve values
print(table1.Id1);
//output:  A
print(table1.Id2);
//output:  B

//a local variable that has 2 implicit meanings:
var1 = somevar; //means copying the value of variable 'somevar' to 'var1', or var1 is int
converted from characters 'somevar'

```

## Operators

```

&& //conditional and
and //conditional and same as &&
|| //conditional or
or //conditional or same as ||
+ //plus or bitwise and
- //subtract
*// multiply
/ // divide
% //modulus
= //assignment
== // equal to (comparison)
! //Null or false check
!= //not equal to
< // less than
> // greater than
<= // less than or equal to
>= // greater than or equal to
& //bitwise and
| // bitwise or

```

## Variable Scope

the default variable scope of GM script is 'local', any declared variable is always 'local' variable unless you declared it with keyword 'global':

```

//a local variable 'outside' a function cannot be accessed within the function
var1 = 1;
somefunc = function()
{
    print("var1 is: ", var1);
};
somefunc();
//output: 'var1 is ' (var1 is null)

```

with 'global' keyword, it will work as intended

```

global var1 = 1;
somefunc = function()
{
    print("var1 is: ", var1);
};
somefunc();
//output: 'var1 is 1' (var1 is 1)

```

## Functions

You dont have to define the parameter value type/the return value type of the function when declaring function in GM:

```

//'local' function, not recommended, cuz it can be garbage-collected easily, and it's
potentially dangerous.

```

```

//use keyword 'global' to declare function to make them permanent. (immune to GC)

```

```

funcIsSet = function(a)
{
    if(a)
    {
        return true;
    }
    else
    {
        return false;
    }
};

```

```

myVar = 1;
if(funcIsSet(myVar) == true)
{
    print("true");
}
else
{
    print("false");
}
//output 'true'

```

```

func2x = function(a)
{
    if(a)
    {
        a *= 2;
        return a;
    }
    else
    {
        return null;
    }
};

```

```
myVar = 1;
myVar2 = func2x(myVar);
print(myVar2);
//output '2'
```

## Loops

while loop:

```
while(condition)
{
}
```

example:

```
i = 2;
while(i > 0)
{
    print("i: ", i);
    print("\n");
    i = i - 1;
    sleep(0.1);
}
print("finished");
//output:
2
1
finished
```

dowhile loop: (similar to while, but doesn't check condition before the first loop)

```
dowhile(condition)
{
}
```

example:

```
i = 0
dowhile(i > 0)
{
    print("i: ", i);
    print("\n");
    i = i - 1;
    sleep(0.1);
}
print("finished");
print(" now i is: ", i);
//output:
//0
```

```
//finished, now i is: -1
```

Important: while and dowhile require a 'sleep()' or yield() as interval, or it will freeze omni-bot.

for loop:

```
for(condition1;condition2;condition3;){
}
```

note: i++, i-- are not supported in GM script, use i = i + 1, i = i - 1 or i += 1, i -= 1 instead.

foreach loop:

```
foreach(index and value in tablename){
}
```

note: GM doesn't support switch, so you have to use if else if 'switch' instead:

```
number = 5;
if(number == 1)
{
    print("number is 1");
}
else if(number == 2)
{
    print("number is 2");
}
else if(number == 3)
{
    print("number is 3");
}
else if(number == 4)
{
    print("number is 4");
}
else if(number == 5)
{
    print("number is 5");
}
```

```
//output: number is 5
```

## Omni-Bot Scripting

You can script any bot behavior with the excellent combo of Omni-Bot and GM script.

## Bot Functions

These are properties of the bot object, you can gain full-control of the bot via these functions.

bot object: bot object is the bot object that gets passed to GM script, it stores most of the info of a specific bot, you can add additional variables to store useful info in bot object as well.

note: the keyword 'bot' becomes 'this' in def\_bot.gm. ('this' is pointing to 'bot object' in def\_bot.gm)

Here is a list of commonly used functions of 'bot':

### General Functions

bot.Say(); make the bot say the string gets passed to this function. example: bot.Say("Hello"); is equal to make the bot 'type' global message "Hello" in-game.

bot.SayTeam(); make the bot say the string gets passed to this function. example: bot.SayTeam("Hello"); is equal to make the bot 'type' team message "Hello" in-game.

bot.SayVoice(); make the bot say the voicemacro(int) gets passed to this function. example: bot.SayVoice(VOICE.G\_CHEER); is equal to make the bot 'use' voicemacro "G\_CHEER" in-game.

bot.GetTeam() returns the team of a specific bot as int. example:

```
foreach(Id and bot in BotTable)
{
    if(bot.GetTeam() == TEAM.ALLIES)
    {
        bot.Say("my team is Allies");
    }
}
```

then all Allies bots will say "my team is Allies" when this loop is being executed.

bot.ChangeTeam() change a specific bots team to the parameter being passed.

1 = TEAM.AXIS 2 = TEAM.ALLIES in ET 1 = TEAM.TEAM1 2 = TEAM.TEAM2 3 = TEAM.TEAM3 4 = TEAM.TEAM4 in other games that has more than 2 teams at 1 time.

bot.GetClass() returns the class of a specific bot as int.

example

```
foreach(Id and bot in BotTable)
{
    if(bot.GetClass() == CLASS.ENGINEER)
    {
        bot.SayVoice(VOICE.IMA_ENGINEER);
    }
}
```



```
}  
}
```

bot.ChangeClass() change a specific bots class to the parameter being passed.

1 = CLASS.SOLDIER 2 = CLASS.MEDIC 3 = CLASS.ENGINEER 4 = CLASS.FIELDOPS 5 = CLASS.COVERTOPS  
in ET returned values vary in different games/mods.

bot.HasTarget() returns true if bot has target, false if not.

bot.GetTarget() gets the game entity of the bots target.

bot.GetNearest(CAT.category, CLASS.class) gets the nearest gameentity.

bot.GetNearestAlly(CAT.category, CLASS.class) gets the nearest friendly gameentity.

bot.GetNearestEnemy(CAT.category, CLASS.class) gets the nearest hostile gameentity.

bot.GetAllType(CAT.category, CLASS.class, tablename) returns a table which contains all entities that matches CAT.category and CLASS.class.

example:

```
myTable = table()  
bot.GetAllType(CAT.PLAYER, CLASS.MEDIC, myTable);
```

then myTable will stores all medic class players (both human and bots) entities that the bot can sense.

bot.GetGameId() gets the game id of the bot(aka slot).

bot.GetGameEntity() gets the game entity of the bot.

bot.GetPosition() gets the bot position as Vector3.

bot.GetFacing() gets the bot facing as Vector3.

bot.DistanceTo(Position) calculate the distance (in in-game units) to a specific Vector3 position.

bot.HasLineOfSightTo(Position) returns true if bot can 'see' the Vector3 Position, return false if not.

bot.IsAllied() checks whether the Gameld or gameentity passed to this function is friendly. returns 'true' if friendly, 'false' if not/hostile.

## Movement functions

bot.EnableMovement() pass 'true' to enable bot movement, pass 'false' to disable bot movement (instantly).

bot.SetScriptControlled() set the bot to scriptcontrolled, unlike EnableMovement, it will wait for the previous 'GoTo sub-goal' to complete/getting overridden before taking over the bot movement.

bot.GoTo(Vector3 Position, Vector3 Facing) make the bot plan a path to a specific position with the specific facing. requires bot.SetScriptControlled(true):

bot.MoveTowards(Vector3 Position) make the bot movetowards a specific location for a very small amount of time without using pathplanner/without taking obstacles into consideration.requires bot.SetScriptControlled (true):

## Weapon functions

### For all games

bot.GetCurrentWeapon() returns the current weapon of the bot as int.

bot.HasWeapon() checks whether if the bot has a specific weapon(uses values from WEAPON table). return true or false if not.

bot.HasAmmo() checks whether if the bot has a specific type of ammo(uses values from WEAPON table). return true or false if not.

bot.SetScriptControlledWeapons() set the bot weapon system to script-controlled.

bot.SelectWeapon() make the bot select a specific weapon.Requires bot.SetScriptControlledWeapons(true)

bot.FireWeapon() make the bot press 'fire weapon' button.Requires bot.SetScriptControlledWeapons(true)

bot.PressButton() make the bot press one or more button at 1 time.Requires bot.SetScriptControlledWeapons (true)

bot.HoldButton(BTN.button, time) make the bot 'hold' one button for a fixed amount of time.Requires bot. SetScriptControlledWeapons(true)

bot.TurnToPosition(Vector3 Position) make the turn to a specific Vector3 Position for a very small amount of time.

bot.TurnToFacing(Vector3 Facing) make the bot turn to a specific Vector3 Facing for a very small amount of time.

bot.EnableShooting() pass 'true' to enable shooting, pass 'false' to disable shooting.

### ET-specific

bot.IsWeaponCharged() checks whether the weaponId being passed is charged(char bar)

bot.PickNewPrimaryWeapon() make the bot pick a specific primary weapon upon respawn.

bot.PickNewSecondaryWeapon() make the bot pick a specific secondary weapon upon respawn.

2.2 Entity Functions GetEntPosition(gameentity) returns the position of gameentity as Vector3

GetEntFacing(gameentity) returns the facing of gameentity as Vector3

GetEntityTeam(gameentity) returns the team of gameentity as int, returns null if the entity is neutral.

GetGameEntityFromId(Id) gets gameentity from GameId(slot number)

GetGameIdFromEntity(gameentity) gets GameId(slot number) from gameentity

## GM Functions

sleep(time) sleep for time seconds example: sleep(1.0); will delay the execution of the function for 1 sec.

yield() pause the execution for 1 server frame 1 server frame = 1 sec/sv\_fps It's not very reliable since some servers use sv\_fps 100 (ET default 20), which means it uses up to 5 times as normal as cpu time when executing infinite loops with yield(); a good alternative for yield() is sleep(0.05)

block(condition) will block the execution of a function until the condition(s) is met. example:

```
bot.GoTo( Vector3(0,0,0), GetEntFacing( bot.GetGameEntity() ) );
if( block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED) == EVENT.GOAL_SUCCESS )
{
    print("GoTo Goal Success");
}
else
{
    print("GoTo Goal Failed");
}
```

## Omni-Bot Event Callback Functions

Event Callback Functions get called every time the event triggers

```
bot.Events[EVENT.event] = [callback_function_name];
```

## Spawn Function

```
this.Events[EVENT.SPAWNED] = spawn_func; //now spawn_func gets called when 'SPAWNED' event triggers
```

```
global spawn_func = function()
{
    this.Say("Hello the world");
};
```

'this' bot will say "Hello the world" every time he spawns/respawns.

## Voice Function

```
this.Events[EVENT.TEAM_VOICE] = voice_func;
```

now voice\_func gets called when 'TEAMVOICE' or voicemacro(team) event triggers

\_source is the Gameld of the player or the bot who just said the voicemacro \_params is a table which contains the voicemacro name and other info etc.

```
global voice_func = function(_source, _params)
{
    if(_params.macro == VOICE.THANKS)
    {
        //get the source of the voice-thanks from '_source' player id
        gratefulguy = GetGameEntityFromId(_source);
        //check against the bot itself to avoid replying to its own voicechat
        if ( gratefulguy != this.GetGameEntity() )
        {
            Posgratefulguy = GetEntPosition( gratefulguy );
            if(Posgratefulguy)
            {
                distance = this.DistanceTo(Posgratefulguy);

                //the distance between the source of the voice (human player) and the bot <=200 in game
units
                //to ensure this bot is the bot who just revived/rearmed/healed the human player
                if (distance <= 200)
                {
                    //this bot must be either medic or fops
                    if (this.GetClass() == CLASS.MEDIC || this.GetClass() == CLASS.FIELDOPS)
                    {
                        sleep(1.0);
                        this.SayVoice(VOICE.WELCOME);
                    }
                }
            }
        }
    }
};
```

## Death Function

```
this.Events[EVENT.DEATH] = death_func;
```

now death\_func gets called whenever 'this' bot is killed

\_params is a table which contains: meansofdeath: the weapon/projectile/world name that killed the bot as string inflictor: gameld equal to \_source

```
global death_func = function(_source, _params)
{
    //if the bot gets killed by Artillery say something
    if(_params.meansofdeath == "MOD_ARTY")
    {
        print("stop arty spamming!");
    }
    //Get gameenti of the GameId who just killed 'this' bot
    sourceEnt = GetGameEntityFromId(_source);
```

```

if( sourceEnt )
{
    //Get position of the gameentity who just killed 'this' bot
    sourcePos = GetEntPosition(sourceEnt);
    if(sourcePos)
    {
        this.Say("you n00b is hiding at", sourcePos);
    }
}
};

```

## Killedsomeone Function

```

this.Events[EVENT.KILLEDsomeone] = killedsomeone_func;

```

now killedsomeone\_func gets called whenever 'this' bot killed someone

```

global killedsomeone_func = function(_source, _params)
{

    //uses this.killcounter a property of bot object to store number of kills
    //set it to 0 if it doesnt exist
    if (this.killcounter == null)
    {
        this.killcounter = 0;
    }

    //non-TK kill: killcounter + 1
    if (this.IsAllied( _source ) == false)
    {
        this.killcounter += 1;
    }

    //check if 'this' bot is already on killing spree or not using another property of the bot
    object 'this.OnKS'
    if (this.OnKS)
    {
        return;
    }

    //on killing spree
    this.OnKS = 1;

    //timeLimit 10 secs(10000 millisecs)
    endtime = GetTime() + 10000;
    while(GetTime() < endtime)
    {
        //say G_CHEER if the bot kills 3 enemies in a row in less than 10 secs
        if (this.killcounter > 2)
        {

```

```

        this.SayVoice(VOICE.G_CHEER);
        break;
    }
    sleep(0.5);
}
//reset killcounter
this.killcounter = 0;
//null OnKS
this.OnKS = null;
return;
};

```

## Feelpain Function

```
this.Events[EVENT.FEEL_PAIN] = feelpain_func;
```

now feelpain\_func gets called whenever 'this' bot gets hit/feels pain

\_params.inflictor: gameentity of the pain inflictor

```

global feelpain_func = function(_source, _params)
{
    //given this function a 5 secs duration to reduce the times this function gets called.
    //or it'll probably become a major resource hogger
    if (this.inthepain)
    {
        return;
    }
    this.inthepain = 1;

    //if this 'bot' has no target
    if( !this.HasTarget() )
    {
        //if inflictor exist
        if(_params.inflictor)
        {
            //if inflictor is hostile
            if( !this.IsAllied(_params.inflictor) )
            {
                //get inflictor position
                inflictorPos = GetEntPosition(_params.inflictor);
                if(inflictorPos)
                {
                    //turn to inflictor's position within .2 secs(200 millisecs)
                    endtime = GetTime() + 200;
                    dowhile(GetTime() < endtime)
                    {
                        this.TurnToPosition(inflictorPos);
                        sleep(0.03);
                    }
                }
            }
        }
    }
}

```

```
}  
sleep(5.0);  
//free inthepain after 5 secs, so that this function can be called every 5 sec.  
this.inthepain = null;  
return;  
};
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Advanced\\_Scripting\\_Guide](http://omni-bot.com/wiki/index.php?title=Advanced_Scripting_Guide)"

This page has been accessed 181 times. This page was last modified 23:00, 15 March 2008.

# Weapon sample.gm

From Omni-bot Wiki

```
// The new weapon instance is passed to this function as the 'this'.
// Properties can be set like this.Property = x, or the this prefix can be left off.

////////////////////////////////////

// Overall weapon properties.
this.Name = "Shotgun";
this.WeaponId = WEAPON.SHOTGUN;

// It is critical that these values correspond to the numeric id's that the game and bot use.
// Usually this is part of the global WPN table.
// Don't Defining AmmoType for modes that NeedsAmmo = false.
this.PrimaryFire.AmmoType = AMMO.SHELLS;

////////////////////////////////////
// Set some weapon properties that affect some internal logic.
this.PrimaryFire.WeaponType = "instant"; // The type of weapon this is. Helps determine
usage behavior. Valid values: "melee", "instant", "projectile"
this.PrimaryFire.ProjectileSpeed = 1000; // If the WeaponType is "projectile", this
projectile speed is used in the aim calculations
this.PrimaryFire.NeedsAmmo = true; // weapon requires ammo to fire. if false ammo
won't be checked, default is true
this.PrimaryFire.WaterProof = true; // weapon can work underwater. if false
desirability will default to 0 when underwater, default true
this.PrimaryFire.NeedsCharged = false; // weapon needs to be charged up. if true bot
internally checks if charged before considering for use, default false
this.PrimaryFire.SplashDamage = false; // weapon does splash damage where it hits, default
false
this.PrimaryFire.InheritsVelocity = false; // weapon inherits velocity from shooter, rather
than being shot with independent velocity. default is false
this.PrimaryFire.ProjectileGravity = 0.0; // What ratio of the games gravity is applied to
the projectile, default 0.0
this.PrimaryFire.NeedsInRange = false; // Only allow weapon to be used when the target is
within MinRange-MaxRange, default false
this.PrimaryFire.MinRange = 0.0; // MinRange used for NeedsInRange check
this.PrimaryFire.MaxRange = 1000.0; // MaxRange used for NeedsInRange check
this.PrimaryFire.DelayAfterFiring = 0.0; // Time the bot will ignore this weapon choice
after shooting.

this.PrimaryFire.MaxAimError = Vector2(0, 0); // Aim error to apply to aiming system. x =
horizontal, y = vertical, default Vector2(0,0)
this.PrimaryFire.AimOffset = Vector3(0, 0, 0); // 3d Offset the weapon should add to the
targets position, default Vector3(0,0,0)

// The default desirability when the bot a) Has no target, or b) No desirability window
matches. default is 0.0
this.PrimaryFire.DefaultDesirability = 0.0;

// Set some distance windows and the desirability they should use.
```



```

// Undefined ranges default to 0.
// Note: The order of these is important. The first matching range is used by the bot.
this.PrimaryFire.SetDesirabilityWindow(0, 64, 0.8);

// If this function is defined in the weapon, the bot will make script callbacks to support
more complex calculations
// to determine the desirability of the weapon. Frequent script callbacks can get expensive, so
use the above methods
// as much as possible to let the bot calculate the desirability internally based on parameters
set by script.
// Default behavior is 0 desirability.

this.PrimaryFire.CalculateDefaultDesirability = function(bot)
{
    return 0;
};

this.PrimaryFire.CalculateDesirability = function(bot, targetInfo)
{
    return 0;
};

// If this function is defined, it will be called to allow the script to calculate the point on
the enemy to aim at.
// Frequent script callbacks can get expensive, so use the above methods as much as possible to
let
// the bot calculate the desirability internally based on parameters set by script.
// Default behavior is to aim at the targets position, based on the WeaponType specified above.
// Projectile weapons will automatically lead the target, instant & melee will aim at the
targets position.
this.PrimaryFire.CalculateAimPoint = function(bot, targetInfo)
{
    return targetInfo.LastPosition;
};

////////////////////////////////////
// Secondary Fire Properties

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Weapon\\_sample.gm](http://omni-bot.com/wiki/index.php?title=Weapon_sample.gm)"

This page has been accessed 320 times. This page was last modified 21:41, 15 September 2007.

## Contents

- 1 ScriptGoal Properties
  - 1.1 Name
  - 1.2 Parent
  - 1.3 Disable
  - 1.4 AimVector
  - 1.5 AlwaysRecieveEvents
  - 1.6 Events
  - 1.7 Bot
  - 1.8 Priority
- 2 ScriptGoal Callbacks
  - 2.1 Initialize
  - 2.2 OnSpawn
  - 2.3 GetPriority
  - 2.4 Enter
  - 2.5 Exit
  - 2.6 Update
- 3 ScriptGoal Functions
  - 3.1 AddFinishCriteria
    - 3.1.1 Subject
    - 3.1.2 Expression
      - 3.1.2.1 Valid Operators
  - 3.2 Finished
  - 3.3 LimitTo
  - 3.4 LimitToClass
  - 3.5 LimitToTeam
  - 3.6 LimitToEntityFlag
  - 3.7 LimitToNoEntityFlag
  - 3.8 LimitToNoPowerup
  - 3.9 LimitToNoTarget
  - 3.10 LimitToPowerUp
  - 3.11 LimitToWeapon
  - 3.12 LimitToTargetClass
  - 3.13 LimitToTargetTeam
  - 3.14 LimitToTargetPowerUp
  - 3.15 LimitToTargetNoPowerUp
  - 3.16 LimitToTargetEntityFlag
  - 3.17 LimitToTargetNoEntityFlag
  - 3.18 LimitToTargetWeapon
  - 3.19 IsActive
  - 3.20 Goto
  - 3.21 DidPathSucceed
  - 3.22 DidPathFail
  - 3.23 AddAimRequest
    - 3.23.1 Priority

- 3.23.2 Aim Types
- 3.24 ReleaseAimRequest
- 3.25 AddWeaponRequest
  - 3.25.1 Priority
- 3.26 ReleaseWeaponRequest
- 3.27 BlockForWeaponChange
- 4 Script Goal Threading
- 5 Script Goal Performance

## ScriptGoal Properties

### Name

The name of the script goal. Important for other scripts to refer to by name, and the name it is displayed under in the Debug Window `this.Name = "MyGoalName";`

### Parent

The name of the parent goal. This should be set up once in the script and left alone. It controls where in the behavior tree the goal is inserted. The location of the goal in the state tree effects how it is evaluated.

Users should stick to one of the following 2 parents.

- ScriptGoals - Puts the goal under the ScriptGoals parent. Because the ScriptGoals parent is under the HighLevel state, only one state at a time can run. This is useful for goals that take control of the bot, that shouldn't be interrupted by other script goals or built in goals.
- LowLevel - Puts the goal under the LowLevel parent. All states under the LowLevel state are run simultaneously, and so is a good location for scripts that control things that don't necessarily take total control over the bot, or don't wish to interrupt the execution of other goals. Some examples might be calling for medic, calling for ammo, most communication scripts.

`this.Parent = "ScriptGoals";`

### Disable

Enables/Disables the goal for consideration. The goal will no longer be considered for execution. Disabled goals have specific coloring in the Debug Window

`this.Disable = true;`

### AimVector

This Vector3 is used by any active AimRequest the script goal has registered.

`this.AimVector = Vector3(x,y,z);`

### AlwaysRecieveEvents

Normally goals recieve events only when active. Setting this to true will allow the goal to recieve events even

when it's not running. This is useful for when you want to have the goal sit idle and only activate when receiving a certain event, or you want to track and store information in the goal from events.

```
this.AlwaysReceiveEvents = true;
```

## Events

Accesses the Events table of the goal, in order to set up script functions that will be called when the bot receives events. By default, goals only receive events when they are active. This can be overridden with AlwaysReceiveEvents, which will allow goals to receive events even when not active.

```
this.Events[EVENT.FEEL_PAIN] = function(Inflictor, PreviousHealth, CurrentHealth)
{
    whoDoneIt = GetEntName(Inflictor);
    print( format("%s: %s Hurt me: Was %d, now %d", this.Bot.GetName(), whoDoneIt,
PreviousHealth, CurrentHealth) );
};
```

## Bot

This provides an accessor to the bot that is running the goal. The 'this' within script goals is always the script goal itself, so in the frequent case of needing to access the bot, use this property.

```
this.Bot.Say("Wassup");
```

## Priority

Priority is a property rather than a return value of GetPriority in order to support the useful threading nature of script goals. It is possible to completely leave out a GetPriority function, and set the priority through this property somewhere else, such as in an event callback. This property will automatically be reset internally when the state exits, to prevent a script goal from activating in a loop from the user forgetting to reset the Priority.

```
this.Priority = 1;
```

## ScriptGoal Callbacks

These functions are available only in 0.7 and later.

### Initialize

This function is called exactly once per bot, when the bot is added and initialized. It is typically used to perform one time initialization code.

#### Example:

```
this.Initialize() = function()
{
};
```

---

## OnSpawn

This function is called every time the bot spawns. Spawning in Omni-bot means any time the bot re-enters his Main active state. This could mean an actual respawn, a revival from an incapacitated state, or something similar.

### Example:

```
this.OnSpawn() = function()  
{  
};
```

---

## GetPriority

This function is most often the place to evaluate whether you want the script goal to run. Take care in the complexity of the code here, as it is called pretty often, and can be a source of considerable performance overhead.

### Example:

```
this.GetPriority() = function()  
{  
};
```

---

## Enter

This function is called when a state is chosen to run, right before it begins running. This function is only called once when the state enters, and will not be called again until the state is exited.

### Example:

```
this.Enter() = function()  
{  
};
```

**See Also:** Exit

---

## Exit

This function is called when a state finishes running, or is overridden by a higher priority state. It is often used to release aim or weapon requests that were used by the goal.

### Example:

```
this.Exit() = function()  
{
```

```
};
```

**See Also:** Enter

---

## Update

This function is the meat of a script goal. Once a goal is chosen to run, and Enter is called, the Update function begins getting called at frequent regular intervals. The state stays active until another goal overrides it, or until Finished is called, after which the Exit function is called.

**Example:**

```
this.Update() = function()
{
    // do stuff
    this.Finished(); // when we're done running
};
```

**See Also:** Enter, Exit

---

## ScriptGoal Functions

These functions are available only in 0.7 and later.

---

### AddFinishCriteria

Finish Criteria are expressions set up from script that will be evaluated constantly in the code and when they are satisfied will cause the script goal to abort. These are useful when your script does alot of asynchronous operations(like Goto), but you want a way to abort the script goal, even if the main update script happens to be in an asynchronous or blocked call, sleep or yield.

**Parameters:** (subject, expression, value[optional])

**Parameters:** (expression, value[optional])

### Subject

Subject must be an entity, if no entity is provided as the first parameter, it is assumed that the 2nd version of the function is called, which takes only and expression and an optional value. The need for a value is determined by what the expression actually is.

### Expression

Expression is a string containing 1 or more keywords that use a natural syntax for defining the conditions of the criteria. Additionally, some keywords expect an operator to be defined

Keyword	Operator Required	Value

deleted	No	entity
health	Yes	health value to compare against(float or int)
hasentflag	No	entity flag to compare against. use not to negate test
weaponcharged	No	weapon Id(integer)
weaponequipped	No	weapon Id(integer)
velocity	Yes	vector3 velocity to compare against, OR float magnitude of velocity to compare against(speed)

### Valid Operators

- lessthan
- <
- greaterthan
- >
- equals
- ==

In addition, some expressions can contain negation keywords in order to check for the negative of an expression, such as

```
this.AddFinishCriteria(this.TargetEntity,"not hasentflag",ENTFLAG.DEAD); // as soon as the entity does not have the dead flag
```

**Returns:** true if criteria was created, false if not

### Example:

```
this.AddFinishCriteria(this.TargetEntity,"deleted"); // finish the goal of the target is deleted
this.AddFinishCriteria(this.TargetEntity,"health lessthan",1); // or if they are dead(health <= 1)
this.AddFinishCriteria(this.TargetEntity,"hasentflag",ENTFLAG.LIMBO); // or if they tap out
```

---

## Finished

Notifies the goal that it is finished running. This function is normally called from the Update function when the goal is desired to finish.

**Parameters:** (none)

**Returns:** none

### Example:

```
this.Finished();
```

---

## LimitTo

This function limits script execution based on a true value being returned from a user defined function. This function is useful if scripts should only be run based on some external logic; like a Map variable setting.

**Parameters:** (function, delay, onlyactive[optional - default false])

**Returns:** none

LimitTo functions are script functions that are called at user defined intervals that can be used to update from script whether or not the script goal should be able to run or not. Returning false from this function effectively blocks the script goals ability to activate. It will remain in this state as long as the LimitTo function is defined. The 2nd parameter controls how often the callback is performed. Calling script functions too often can adversely effect performance, so it is recommended to choose a delay that is as infrequently as you can get away with. The 'onlyactive' parameter is optional and defaults to false. If this parameter is set to true, the LimitTo function callback will ONLY be performed when the script goal is active. By default the LimitTo function will be called whether the goal is active or not, and returning false will prevent the goal from being able to activate. Passing true as the 'onlyactive' flag means the goal can activate, and only then is the LimitTo function called. This is useful when you need to check more information regarding specific targets of the goal to validate whether the goal should continue running. This callback is effectively a way to define your own finish criteria. It is recommended that you only use LimitTo callbacks when FinishCriteria can't be used, since FinishCriteria are much faster and don't involve frequent calls into script. LimitTo functions might commonly be used to check the status of a map goal or some other world state that the goal relies on.

**Example:**

```
//This function is typically set up in this.Initialize()
this.LimitTo(Map.SomeFunction, 1.0); //Call Map.SomeFunction every second
this.LimitTo(NULL); //clear the function

//The script goal object is referenced in the passed function as 'this'
Map.SomeFunction = function()
{
    if ( Map.SomeVariable == false )
        { this.Disable; } //disable the script
    else
        { return true; } //allow the script to run
};
```

---

## LimitToClass

This function will only allow the script goal to be created if the owning bot is one of the classes provided. This is much more efficient than checking class in script in common cases where a goal only applies to a certain class of bot.

**Parameters:** (classId, ...) Any number of class Ids

**Returns:** none

**Example:**



```
// This function can take any number of parameters.  
this.LimitToClass(CLASS.MEDIC);  
this.LimitToClass(CLASS.MEDIC, CLASS.ENGINEER);
```

---

## LimitToTeam

This function will only allow the script goal to be created if the owning bot is one of the teams provided. This is much more efficient than checking team in script in common cases where a goal only applies to a certain team of bots.

**Parameters:** (teamId, ...) Any number of class Ids

**Returns:** none

**Example:**

```
// This function can take any number of parameters.  
this.LimitToTeam(Team.BLUE);  
this.LimitToTeam(Team.BLUE, Team.RED);
```

---

## LimitToEntityFlag

This function will only allow the script goal to be created if the owning bot has one or more entity flags set. This is much more efficient than checking entity flags in script in common cases where a goal only applies to specific cases where the bot has certain entity flags.

**Parameters:** (ent flag, ...) Any number of entity flags

**Returns:** none

**Example:**

```
// This function can take any number of parameters.  
this.LimitToEntityFlag(ENTFLAG.INWATER);  
this.LimitToEntityFlag(ENTFLAG.INWATER, ENTFLAG.UNDERWATER);
```

---

## LimitToNoEntityFlag

This function will only allow the script goal to be created if the owning bot does not have one or more entity flags set. This is much more efficient than checking entity flags in script in common cases where a goal only applies to specific cases where the bot does not have certain entity flags.

**Parameters:** (ent flag, ...) Any number of entity flags

**Returns:** none

### Example:

```
// This function can take any number of parameters.
this.LimitToNoEntityFlag(ENTFLAG.INWATER);
this.LimitToNoEntityFlag(ENTFLAG.INWATER, ENTFLAG.UNDERWATER);
```

---

## LimitToNoPowerup

This function will only allow the script goal to be considered if the owning bot does not have the given powerups.

**Parameters:** (powerup flag, ...) Any number of power up flags

**Returns:** none

### Example:

```
// This function can take any number of parameters.
this.LimitToNoPowerUp(ENTFLAG.INVINCIBLE);
this.LimitToNoPowerUp(ENTFLAG.INVINCIBLE, ENTFLAG.QUADDAMAGE);
```

---

## LimitToNoTarget

This function will only allow the script goal to be considered if the owning bot does not have a target. This is much more efficient than checking for a target in the GetPriority function in script in cases where the bot should not have a target. The goal will not be evaluated unless the bot does not have a target.

**Parameters:** none

**Returns:** none

### Example:

```
// This function takes no parameters.
this.LimitToNoTarget();
```

---

## LimitToPowerUp

This function will only allow the script goal to be created if the owning bot has one or more power up flags set. This is much more efficient than checking power up flags in script in common cases where a goal only applies to specific cases where the bot has certain power up flags.

**Parameters:** (powerup flag, ...) Any number of power up flags

**Returns:** none

## Example:

```
// This function can take any number of parameters.  
this.LimitToPowerUp(ENTFLAG.INVINCIBLE);  
this.LimitToPowerUp(ENTFLAG.INVINCIBLE, ENTFLAG.QUADDAMAGE);
```

---

## LimitToWeapon

This function will only allow the script goal to be created if the owning bot has one or more weapons set by this function. This is much more efficient than checking the bots weapons in script in common cases where a goal only applies to specific cases where the bot has a certain weapon.

**Parameters:** (weapon Id, ...) Any number of weapon Ids

**Returns:** none

## Example:

```
// This function can take any number of parameters.  
this.LimitToWeapon(WEAPON.MOBILE_MG42);  
this.LimitToWeapon(WEAPON.MOBILE_MG42, WEAPON.MORTAR);
```

---

## LimitToTargetClass

This function will only allow the script goal to be considered if the owning bot has a target of a specified class. This is much more efficient than checking class in the GetPriority function in script in cases where a goal only applies to specific cases where the bots target is a specific class. The goal will not be evaluated unless the bot has a target and the target is on one of the provided classes.

**Parameters:** (classId, ...) Any number of target classes

**Returns:** none

## Example:

```
// This function can take any number of parameters.  
this.LimitToTargetClass(CLASS.ANYPLAYER); // don't run unless we have any player class  
this.LimitToTargetClass(CLASS.ENGINEER, CLASS.MEDIC); // look for specific classes
```

---

## LimitToTargetTeam

This function will only allow the script goal to be considered if the owning bot has a target on a specified team. This is much more efficient than checking team in the GetPriority function in script in cases where a goal only applies to specific cases where the bots target is on a specific team. The goal will not be evaluated unless the bot has a target and the target is on one of the provided teams.

**Parameters:** (teamId, ...) Any number of teams

**Returns:** none

**Example:**

```
// This function can take any number of parameters.  
this.LimitToTargetTeam(Team.RED); // don't run unless our target is on red team
```

---

## LimitToTargetPowerUp

This function will only allow the script goal to be considered if the owning bot has a target with one or more powerups. This is much more efficient than checking powerups in the GetPriority function in script in cases where a goal only applies to specific cases where the bots target has a powerup. The goal will not be evaluated unless the bot has a target and the target is on one of the provided powerups.

**Parameters:** (powerup id, ...) Any number of power up flags

**Returns:** none

**Example:**

```
// This function can take any number of parameters.  
this.LimitToTargetPowerUp(PowerUp.Quaddamage); // don't run unless our target has quad damage  
this.LimitToTargetPowerUp(EntFlag.Invincible, EntFlag.Quaddamage);
```

---

## LimitToTargetNoPowerUp

This function will only allow the script goal to be considered if the owning bot has a target without one or more powerups. This is much more efficient than checking powerups in the GetPriority function in script in cases where a goal only applies to specific cases where the bot's target hasn't a powerup. The goal will not be evaluated unless the bot has a target and the target doesn't have one of the provided powerups.

**Parameters:** (powerup id, ...) Any number of power up flags

**Returns:** none

**Example:**

```
// This function can take any number of parameters.  
this.LimitToTargetNoPowerUp(PowerUp.Quaddamage); // don't run if our target has quad damage  
this.LimitToTargetNoPowerUp(EntFlag.Invincible, EntFlag.Quaddamage);
```

---

## LimitToTargetEntityFlag

This function will only allow the script goal to be created if the target bot has one or more of the entity flags set. This is much more efficient than checking entity flags in script in common cases where a goal only applies to specific cases where the bot has certain entity flags.

**Parameters:** (ent flag, ...) Any number of entity flags

**Returns:** none

**Example:**

```
// This function can take any number of parameters.
this.LimitToTargetEntityFlag(ENTFLAG.INWATER);
this.LimitToTargetEntityFlag(ENTFLAG.INWATER, ENTFLAG.UNDERWATER);
```

---

## LimitToTargetNoEntityFlag

This function will only allow the script goal to be created if the target bot does not have one or more of the entity flags set. This is much more efficient than checking entity flags in script in common cases where a goal only applies to specific cases where the bot does not have certain entity flags.

**Parameters:** (ent flag, ...) Any number of entity flags

**Returns:** none

**Example:**

```
// This function can take any number of parameters.
this.LimitToTargetNoEntityFlag(ENTFLAG.INWATER);
this.LimitToTargetNoEntityFlag(ENTFLAG.INWATER, ENTFLAG.UNDERWATER);
```

---

## LimitToTargetWeapon

This function will only allow the script goal to be created if the bots has one or more weapons set by this function. This is much more efficient than checking the bots weapons in script in common cases where a goal only applies to specific cases where the bot's target has a certain weapon.

**Parameters:** (weapon Id, ...) Any number of weapon Ids

**Returns:** none

**Example:**

```
// This function can take any number of parameters.
this.LimitToTargetWeapon(WEAPON.MOBILE_MG42);
this.LimitToTargetWeapon(WEAPON.MOBILE_MG42, WEAPON.MORTAR);
```

---

## IsActive

Checks if the script goal is currently active. An active script goal means it is currently running. This check is useful mostly in cases where you want an active script goal to remain active and not necessarily keep performing complex checks in its GetPriority function.

**Parameters:** (none)

**Returns:** true if active, false if not

**Example:**

```
this.GetPriority = function()  
{  
    if(this.IsActive())  
    {  
        return 1.0;  
    }  
    else  
    {  
        // do some other check  
    }  
};
```

---

## Goto

Tells the bot to go to a specific location, planning and running a path if necessary. Typically a script would block after calling this to wait for a success or failure message.

**Parameters:** (Vector3 position, tolerance)

**Returns:** none

**Example:**

```
wpinfo = table();  
wp = Wp.GetWaypointByName("somewaypoint", wpinfo);  
this.Goto(wpinfo.position, wpinfo.radius);  
if( block(EVENT.PATH_SUCCESS, EVENT.PATH_FAILED) == EVENT.PATH_SUCCESS )  
{  
    print("Made it!");  
}
```

---

## DidPathSucceed

Checks if the last Goto command succeeded. Success means the bot has reached the end of the desired path. Typically a goal will block on EVENT.PATH\_SUCCESS or EVENT.PATH\_FAILED events. DidPathSucceed and

DidPathFail are included for cases where the script wishes to remain active and doing something during the path traversal, and it can poll the path success/failure during. If neither function returns true, then the bot should be moving along the path normally.

**Parameters:** (none)

**Returns:** true if path succeeded, false if not

**Example:**

```
if( this.DidPathSucceed() )
{
}
```

**See Also:** Goto, DidPathSucceed

---

## DidPathFail

Checks if the last Goto command failed. Failure means the path was either blocked(no path exists), or the bot has gotten stuck on the way. Typically a goal will block on EVENT.PATH\_SUCCESS or EVENT.PATH\_FAILED events. DidPathSucceed and DidPathFail are included for cases where the script wishes to remain active and doing something during the path traversal, and it can poll the path success/failure during. If neither function returns true, then the bot should be moving along the path normally.

**Parameters:** (none)

**Returns:** true if path failed, false if not

**Example:**

```
if( this.DidPathFail() )
{
}
```

**See Also:** Goto, DidPathSucceed

---

## AddAimRequest

Adds a priority based aim request into the bots aiming system. Aim requests are favored based on their priority, with the highest priority aim request chosen at any given point. It is a much simpler and easier to manage setup in the cases of many competing goals that wish to control the bots aim. Each goal can add an aim request and will receive notifications when the bot chooses and executes their request. This functions works together with the AimVector property to allow setting the aim type at any point. The script can receive notification of a successful aiming by blocking on the EVENT.AIM\_SUCCESS event.

Aim requests can be viewed in the Debug Window under the Aimer state. They will take on the name of the script goal they were added in.

**Parameters:** (priority, aim type<optional, default position>, aim position<optional>)

## Priority

Most normal goals use a `Priority.Medium` priority, which leaves 3 higher levels of priority usable by scripts or important goals.

Priority determines how to select a request when there are multiple requests active. Higher priorities are selected over lower priorities.

- `Priority.Min`
- `Priority.Idle`
- `Priority.VeryLow`
- `Priority.Low`
- `Priority.Medium`
- `Priority.High`
- `Priority.VeryHigh`
- `Priority.Override`

## Aim Types

- `movedirection` - Face in the direction of movement. Useful if you want to override the bots normal combat aiming, as in for a disguise goal where you don't want them to engage the target.
- `facing` - Face a given `facing(direction)` vector. In this case `AimVector` represents a facing vector, and not a position vector

If one of the above is not specified, the aim request is a positional request, and the `AimVector` of the script goal will be used as a raw position to aim at.

**Returns:** none

### Example:

```
aimPos = someAimPosition;
aimPos2 = someAimPosition2;

this.AddAimRequest(Priority.High, "movedirection"); // requests the bot to face along its move
direction at a 0.8 priority
// OR
this.AddAimRequest(Priority.High); // leave off optional parameters, defaults to AimPosition,
so set it
this.AimVector = aimPos; // this can be called at any time in the script goal

if(block(EVENT.AIM_SUCCESS) == EVENT.AIM_SUCCESS)
{
    // we're on target, do something cool
}
```

**See Also:** `ReleaseAimRequest`

---

## ReleaseAimRequest

This function will release the Aim Request within the script goal; allowing other potential aim requests to have



control.

**Parameters:** none

**Returns:** none

**Example:**

```
this.ReleaseAimRequest();
```

**See Also:** AddAimRequest

---

## AddWeaponRequest

Adds a priority based weapon equip request into the bots weapon system. Weapon requests are favored based on their priority, with the highest priority request chosen as the desired weapon. It is a much simpler and easier to manage setup in the cases of many competing goals that wish to control the bots equipped weapon. Each goal can add an weapon request and will receive notifications when the bot chooses and executes their request.

Weapon requests can be view in the Debug Window under the Aimer state. They will take on the name of the script goal they were added in.

**Parameters:** (priority, weapon Id)

### Priority

Most normal goals use a Priority.Medium priority, which leaves 3 higher levels of priority usable by scripts or important goals.

Priority determines how to select a request when there are multiple requests active. Higher priorities are selected over lower priorities.

- Priority.Min
- Priority.Idle
- Priority.VeryLow
- Priority.Low
- Priority.Medium
- Priority.High
- Priority.VeryHigh
- Priority.Override

**Returns:** none

**Example:**

```
this.AddWeaponRequest(Priority.High, WEAPON.MEDKIT);
```

**See Also:** ReleaseWeaponRequest

---

## ReleaseWeaponRequest

This function will release the weapon request within the script goal; allowing other potential weapon requests to have control.

**Parameters:** none

**Returns:** none

**Example:**

```
this.ReleaseWeaponRequest();
```

**See Also:** AddWeaponRequest

---

## BlockForWeaponChange

This function provides specialized script thread blocking functionality to wait for a weapon change to a certain weapon. This function takes the place of a call to block in the case you want to wait until the bot switches to a specific weapon.

**Parameters:** (weaponId)

**Returns:** none

**Example:**

```
this.BlockForWeaponChange(WEAPON.MEDKIT);  
this.Say("Medkit Equipped!");
```

---

## Script Goal Threading

Script goals have been designed to be friendly to script threading, by providing some built in functionality to make it easier to manage multiple threads to be associated with the script goal.

The script goal keeps track of the thread created when calling all the following functions.

- Initialize
- OnSpawn
- GetPriority
- Enter
- Exit
- Update

You may use thread functions (block, yield, sleep), in any of the script goal functions, though you must understand that for efficiency, and to prevent runaway threads, the script goal will internally kill the threads associated with the function callbacks at certain times.

- All threads will be killed when the state exits.
- All threads will be killed when the state is disabled. See Disable
- All threads will be killed when the state becomes unselectable. This is normally due to the limit functions.

Note that this only applies to the thread that is created for the function calls themselves. If you really need a thread to run longer than these specifications allow, you can create your own thread normally with the `thread()` function. It is highly recommended that you script your goal within these functions, as it is easy to *leak* threads when creating them arbitrarily.

---

## Script Goal Performance

Executing script is much much slower than executing native code, so you should be aware that a script can have significant adverse effects on the performance of the bot as a whole. Here are some tips to keep performance as high as possible with script goals.

- **Only define what you need** - None of the callback functions are required, so if you don't use them, don't define them. It is wasteful to have an empty function defined. If you don't need an Enter function, leave it out, if you don't need an Update function, leave it out. Same for Exit, GetPriority, Initialize.
- **Use limit functions as much as possible** - If the goal should only be considered for certain player classes, or for bots with certain powerups or entity flags, or for when the bot has a target that matches similar specific flags, use the Limit functions to set that up. They are much much faster than checking those properties in the GetPriority function. If the limit functions fail, the script will never be called, which is a huge optimization.
- **Use thread functions** - Normally, GetPriority is called often. You can control this behavior by using thread functions to keep a thread running, and delay priority evaluations with a loop and calls to sleep. GetPriority will not be called if there is still an active GetPriority thread.
- **Keep your GetPriority function as lightweight as possible** - The less work it does the better, because GetPriority is called frequently by the bot. This is especially important if the goal is placed under the *LowLevel* parent because all child states under *LowLevel* are evaluated and allowed to run simultaneously.
- **Ditto for the Update function** - The same applies to the Update function especially. Keep it as simple as possible to achieve the desired behavior, and use thread functions.
- **Cache as much as possible** - Take advantage of the fact that Initialize is called only once when the bot is added. Use this function to cache values the script goal might need. For example, if the script goal does things based on specific waypoints, it would be expensive to call functions like `Wp.GetWaypointByName()` in the priority or update functions. Instead collect all you might need up front in the Initialize function and store them in a table on the script goal so you have faster access to them later.
- **Use events, and block() as much as possible** - Most common operations that previously required expensive loops to perform have been converted to events. Choosing a weapon, aiming a weapon, going to some location, etc. Most can be followed by a call to `block()` which will suspend the execution of the script until the event is recieved. This is the single easiest to use and most useful for performance features available. See `BlockForWeaponChange`, `Goto`, `AddAimRequest`.
- **Beware of loops** - Be careful when looping, as they can silently cause large performance drains. Be wary of how much work is being done in a loop, and on how many elements. Expensive functionality can be updated less often, or split up over time. Use `yield()` or `sleep(x)` within loops to spread out the calculations over time.

# RTCW:Script Goals

From Omni-bot Wiki

## Contents

- 1 RTCW Script Goals
  - 1.1 goal\_airstrike.gm
  - 1.2 goal\_askforammo.gm
  - 1.3 goal\_askforhealth.gm
  - 1.4 goal\_combatmovement.gm
  - 1.5 goal\_deliversupplies.gm
  - 1.6 goal\_dispenseammo.gm
  - 1.7 goal\_dispensehealth.gm
  - 1.8 goal\_grenadetarget.gm
  - 1.9 goal\_supplyself.gm
  - 1.10 goal\_useswitch.gm

## RTCW Script Goals

### goal\_airstrike.gm

This script goal checks for a table of waypoints set up for the bot to go to and throw an airstrike cannister. Typical usage would be in cases where there is good chance enemies may be, but the bot may not have a line of sight to. Bots throwing airstrikes over the gate on mp\_assault is an example of the utilization.

### goal\_askforammo.gm

This script goal checks the bots ammo situation occasionally and looks for ammo packs on the ground before calling for ammo using the ET voice macro "Need Ammo!".

### goal\_askforhealth.gm

This script goal checks the bots health situation occasionally and looks for health packs on the ground before calling for health using the ET voice macro "Need Medic!".

### goal\_combatmovement.gm

This script goal runs while the bot has a target and implements some dodging and strafing movement while in combat.

### **goal\_deliversupplies.gm**

This script goal listens for teammate requests for health and ammo, and if the bot is a field ops or medic capable of answering the request, the bot will travel to the callers location and provide them with the requested health/ammo. Useful for mobile mg42 and other defensive positions to be able to replenish ammo or health in the field.

### **goal\_dispenseammo.gm**

This is a simple reimplementaion of the old behavior of dropping ammo shortly after spawn.

### **goal\_dispensehealth.gm**

This is a simple reimplementaion of the old behavior of dropping health shortly after spawn.

- requires mapgoal configuration

### **goal\_grenadetarget.gm**

Gives bots the ability to grenade targets such as mg42's, barbed barriers, etc.

### **goal\_supplyself.gm**

This script goal provides medics and fieldops the ability to give themselves health or ammo when they need it.

### **goal\_useswitch.gm**

Allows bots to treat hitting switches in a map as a goal, also used heavily in railgun.

- requires mapgoal configuration

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=RTCW:Script\\_Goals](http://omni-bot.com/wiki/index.php?title=RTCW:Script_Goals)"

This page has been accessed 38 times. This page was last modified 02:58, 10 June 2008.

# Autowaypoint Script

From Omni-bot Wiki

This is a gm script that was made by Bladen and [GNU]FeelGood/Nobody. Basically, the most important functionality is that it can be used to leave a trail of biconnected waypoints behind you while you run through a map. It can be used in conjunction with a waypointing config file, sample attached below. Put the gm script into your <omni-bot>\et\scripts or <omni-bot>\global\_scripts folder.

See the comments at the top of the script for details.

- Et\_waypointing.gm
- Waypointing\_cfg.zip

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Autowaypoint\\_Script](http://omni-bot.com/wiki/index.php?title=Autowaypoint_Script)"

This page has been accessed 155 times. This page was last modified 18:07, 24 February 2008.

# Chatty bot profile for 0.61

From Omni-bot Wiki

Feel free to use or add useful functions.

TODO on page: syntax-highlighting, currently working on Linux (CR problem)

```
// chatty ET bot profile
// based on default_bot.gm 0.61 and forum ideas
// This bot is not Chuck enough !!!

// print( "Chatty ET bot profile" );

// TODO on first start:
// set properties f.e class, team, FieldOfView
// add your killmessages
// add your onliners
// important: set your custom vsay sounds !!!

// Load some scripts we might use.
// ExecScript( "health_goal.gm" );
// ExecScript( "ammo_goal.gm" );
// ExecScript( "adrenaline.gm");
ExecScript( "disguise_goal.gm" );
ExecScript( "mountvehicle_goal.gm" );
ExecScript( "dispenseammo_goal.gm" );
ExecScript( "dispensehealth_goal.gm" );
ExecScript( "escortvehicle_goal.gm" );
ExecScript( "ridevehicle_goal.gm" );
ExecScript( "useswitch_goal.gm" );

////////////////////////////////////
// Config
////////////////////////////////////

// FieldOfView is the angle(in degrees) that the bot can 'see' in front of them.
this.FieldOfView = 90.0;

// ReactionTime is the time delay(in seconds) from when a bot first see's a target, to when
// the bot will begin to react and target them.
this.ReactionTime = 0.0;

// MemorySpan is how long it takes(in seconds) for a bot to consider his memory of someone or
something
// 'out of date' and not considered for targeting and such
this.MemorySpan = 5.0;

// AimPersistance is how long the bot will aim in the direction of a target after the target
has gone out of view.
// This is useful for keeping the bot aiming toward the target in the event of brief
obstructions of their view.
```

```

this.AimPersistance = 5.0;

// MaxViewDistance is the maximum distance(in game units) the bot is capable of seeing
something.
// This could be tweaked lower for maps with fog or for a closer to human view distance
// Typically this value is best set in the map script in the OnBotJoin callback.
this.MaxViewDistance = 10000.0;

// These 3 values are aim properties. Care must be taken when tweaking aim properties, since
// improper values can produce aim oscillations and hurt the bots combat abilities.
this.MaxTurnSpeed = 720.0; // degree's / second
this.AimStiffness = 75.0;
this.AimDamping = 10.0;

// customize this for your server !!!
// enter keys from sounddefinition for your custom voices here
// played for kill of bot
this.killcustomvoice =
{
    "vsay FTAttack ^7Attack!",
    // example, add more
    //"vsay fu_sod ^7Bock auf GANGBANG ?",
};

// same as killcustomvoice
this.deathcustomvoice =
{
    // add custom vsays
    // "vsay fu_scheterbe ^7Isch kann net scheterbe !!!",
    // "vsay fu_verrueckt ^7Da ist er wieder ... !!!",
};

// killmessages
this.killmessages =
{
    "^7Need more cannon-food!",
    "^7I have nothing to offer but blood, tears and sweat",
    // add more here
};

// Put all wanted voices for kill in this
this.killshouts =
{
    VOICE.LETS_GO,
    VOICE.MOVE,
    VOICE.ON_OFFENSE,
    VOICE.FOLLOW_ME,
    VOICE.COVER_ME,
    VOICE.G_GREATSHOT,
    VOICE.G_ENEMY_WEAK,
    VOICE.G_OOPS,
    VOICE.G_CHEER,
    VOICE.G_GOODGAME,

```



```

};

// probability bot is using voice or message at kill, value from 0 to 100
// most annoying value for kills is 100 :)
this.say_at_kill = 10;
// using messages or voices at kill
this.say_at_kill_mess_or_shout = 49;

// probability bot is using voice or message at kill, value from 0 to 100
// annoying value for death is 100 :)
this.say_at_death = 10;
// using message or voices at death
this.say_at_death_mess_or_shout = 49;

// put in more if wanted
this.deathmessages =
{
    // add more
    "^7Gonna kill you next time [d]^7!!!!",
};

// put in more if wanted
this.deathshouts =
{
    VOICE.G_GREATSHOT,
    VOICE.G_ENEMY_WEAK,
    VOICE.COVER_ME,
    VOICE.G_HOLD_FIRE,
    VOICE.G_NEGATIVE,
    VOICE.REINFORCE_OFF,
    VOICE.NEED_BACKUP,
};

////////////////////////////////////
// END config
////////////////////////////////////

////////////////////////////////////
// Set Weapon & Item properties.
////////////////////////////////////

// Utility Callbacks
this.SelectTeam = function()
{
    return null;
};

this.SelectClass = function()
{
    return null;
};

this.SelectWeapons = function()
{
    myteam = this.GetTeam();

```

```

myclass = this.GetClass();

//~print("SelectWeapons: Class", myclass, "Team", myteam);

weaponSelection = {};

if ( myclass == CLASS.SOLDIER )
{
    // only purpose for this is to choose something other than the mortar, until a
working mortar behavior can be devised.

    if (GetModName() == "noquarter") {
        if ( myteam == TEAM.ALLIES )
        {
            soldierWeapons =
            {
                WEAPON.VENOM,
                WEAPON.BAZOOKA,
                WEAPON.MOBILE_BROWNING,
                WEAPON.FLAMETHROWER,
            };
        }
        else
        {
            soldierWeapons =
            {
                WEAPON.VENOM,
                WEAPON.PANZERFAUST,
                WEAPON.MOBILE_MG42,
                WEAPON.FLAMETHROWER,
            };
        }
        // set RandInt(0,2) to disable Flamer
        weaponSelection.Primary = soldierWeapons[ RandInt( 0, 3 ) ];
    }
    else
    {
        soldierWeapons =
        {
            WEAPON.PANZERFAUST,
            WEAPON.MOBILE_MG42,
            WEAPON.FLAMETHROWER,
        };
        weaponSelection.Primary = soldierWeapons[ RandInt( 0, 2 ) ];
    }
}

if ( weaponSelection.Primary )
    { this.PickNewPrimaryWeapon( weaponSelection.Primary ); }

if ( weaponSelection.Secondary )
    { this.PickNewSecondaryWeapon( weaponSelection.Secondary ); }

```

```
};
```

```
////////////////////////////////////
```

```
global saySkills = function()  
{  
    this.mySkills = table();  
    if(this.GetSkills(this.mySkills))  
    {  
        this.Say("Battle Sense: ", this.mySkills[SKILL.BATTLE_SENSE]);  
        this.Say("Engineering: ", this.mySkills[SKILL.ENGINEERING]);  
        this.Say("First Aid: ", this.mySkills[SKILL.FIRST_AID]);  
        this.Say("Signals: ", this.mySkills[SKILL.SIGNALS]);  
        this.Say("Lght Wpns: ", this.mySkills[SKILL.LIGHT_WEAPONS]);  
        this.Say("Hvy Wpns: ", this.mySkills[SKILL.HEAVY_WEAPONS]);  
        this.Say("Cov Ops: ", this.mySkills[SKILL.COVERTOPS]);  
    }  
    else  
    {  
        this.Say("Don't know my skills!");  
    }  
};
```

```
// TODO  
// turn/combat  
turn_func = function(_source, _params)  
{  
    // TODO get old state ...  
    // get control, ws still on  
    this.SetScriptControlled(true);  
  
    curPos = this.GetPosition();  
    // ... TODO movement etc  
    // ...  
    this.SetScriptControlled(false);  
};
```

```
// How about a simple script for field ops and medics when they spawn to drop some ammo/health  
spawn_func = function(_source, _params)  
{  
    //~ yield();  
  
    // clear any previously scripted goals.  
    this.scriptgoal = null;  
    this.SetScriptControlled(false);  
    this.SetScriptControlledWeapons(false);  
  
    this:thread(ScriptGoals.mount_vehicle.watchForMountableVehicles);  
    this:thread(ScriptGoals.escort_vehicle.watchForEscortableVehicles);  
    ScriptGoals.RideVehicle.Begin(this);  
    ScriptGoals.UseSwitch.Begin(this);  
  
    // Sample Usage of health_goal.gm
```

```

// this:thread(ScriptGoals.watch_health.watchHealthFunc);

// this:thread(ScriptGoals.watch_ammo.watchAmmoFunc);

// Check if this bot is a Medic, and drop some Medkits for a few seconds.
if(this.GetClass() == CLASS.MEDIC)
{
    this:thread(ScriptGoals.dispense_health.DispenseHealth);
}

// Check if this bot is a Field-Ops, and drop some Ammo for a few seconds.
if(this.GetClass() == CLASS.FIELDOPS)
{
    this:thread(ScriptGoals.dispense_ammo.DispenseAmmo);
}

// Sample usage of disguise_goal.gm
//if( this.GetClass() == CLASS.COVERTOPS )
//{
//    this:thread(ScriptGoals.disguise_goal.LookForDisguiseGoalThread);
//}

//~ this.SetDebugFlag(DEBUG.GOALS, true);
};

```

```

// called on death
death_func = function(_source, _params)
{

    if( RandRange(0, 99) < this.say_at_death )
    {
        if ( RandRange(0,99) < this.say_at_death_mess_or_shout )
        {
            // shouts 50/50 using common from voicetable/custom-voices
            if ( RandRange(0,99) < 49 )
            {
                rand1 = RandInt( 0, tableCount(this.deathshouts) -1 );
                this.SayVoice( this.deathshouts[rand1]);
            }
            else
            {
                rand1 = RandInt(0, tableCount(this.deathcustomvoice) -1 );
                this.ExecCommand( this.deathcustomvoice[rand1] );
            }
        }
        else
        {
            rand1 = RandInt( 0, tableCount(this.deathmessages) -1 );
            this.Say( this.deathmessages[rand1] );
        }
    }
}

```

```
};
```

```
// priv-chat does not work with NQ 1.1.1 /ETPUB
```

```
privchat_msg_func = function(_source, _params)
```

```
{
```

```
    this.SayVoice(VOICE.G_HI);
```

```
    this.Say("Hi there");
```

```
    if(_params.msg == "snipe here")
```

```
    {
```

```
        if( this.CanSnipe() )
```

```
        {
```

```
            this.SayVoice(VOICE.AFFIRMATIVE);
```

```
            sourceEntPos = GetEntPosition(_source);
```

```
            sourceEntFacing = GetEntFacing(_source);
```

```
            if(sourceEntPos && sourceEntFacing)
```

```
            {
```

```
                this.Snipe(sourceEntPos, sourceEntFacing);
```

```
                status = block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED);
```

```
                if(status == EVENT.GOAL_FAILED)
```

```
                {
```

```
                    this.Say("Can't do it!");
```

```
                }
```

```
            }
```

```
        }
```

```
        return;
```

```
    }
```

```
    if( _params.msg == "skills" )
```

```
    {
```

```
        this:saySkills();
```

```
        return;
```

```
    }
```

```
// Restrict it to only when waypoint mode is on for now.
```

```
if( Wp.IsWaypointViewOn() )
```

```
{
```

```
    if(_params.msg == "come")
```

```
    {
```

```
        this.SayVoice(VOICE.AFFIRMATIVE);
```

```
        sourceEntPos = GetEntPosition(_source);
```

```
        sourceEntFacing = GetEntFacing(_source);
```

```
        if(sourceEntPos && sourceEntFacing)
```

```
        {
```

```
            this.SetScriptControlled(true);
```

```
            this.GoTo(sourceEntPos, sourceEntFacing);
```

```
        }
```

```

        status = block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED);
        if(status == EVENT.GOAL_SUCCESS)
        {
            this.Say("Now What?");
        }
    }
    else if(_params.msg == "roam")
    {
        this.SayVoice(VOICE.AFFIRMATIVE);
        this.SetScriptControlled(false);
    }
}

};

// called on kills
killed_someone_func = function(_source, _params)
{
    if(this.IsAllied(_source)== false)
    {
        // killed enemy
        this.randSay = RandRange(0,99);
        // print("Say value ", this.randSay);
        if ( this.randSay < this.say_at_kill )
        {
            if ( RandRange(0,99) < this.say_at_kill_mess_or_shout )
            {
                // 50/50 voicetable or custom
                if (RandRange(0,99) < 49)
                {
                    rand1 = RandInt( 0, tableCount(this.killshouts)-1 );
                    this.SayVoice(this.killshouts[rand1]);
                }
                else
                {
                    rand1 = RandInt(0, tableCount(this.killcustomvoice) -1 );
                    this.ExecCommand( this.killcustomvoice[rand1] );
                }
            }
            else
            {
                rand1 = RandInt( 0, tableCount(this.killmessages)-1 );
                this.Say(this.killmessages[rand1]);
            }
        }
    }
    else
    {
        // killed team mate
        victimID = _source;
        if(victimID)
        {
            if(victimID != this.GetGameId() && _params.meansofdeath != "MOD_LANDMINE" && _params.
meansofdeath != "MOD_DYNAMITE")

```

```

        {
            if(!this.inguilt)
            {
                this.inguilt = 1;
                sleep(1.0);
                this.SayVoice(VOICE.SORRY);
                this.inguilt = null;
            }
        }
    }
};

// test event-function
test_func = function(_source, _params)
{
    this.Say("All ok.");
    print("TEST EXECUTED.");
};

// test event-function
hearsound_func = function(_source, _params)
{
    this.Say("Heard sound:", _params.soundId);
    print("Heard sound: ", _params.soundId);
};

// test event-function
receive_weapon_func = function(_source, _params)
{
    // this.Say("Get:", _params.weaponId);
    print("RECEIVE : ", _params.weaponId);
    print("RECEIVE : ", _source);
};

// EVENT-handling
// on spawn. Comment this out if you dont want them to do this.
this.Events[EVENT.SPAWNED] = spawn_func;
// on death
this.Events[EVENT.DEATH] = death_func;
// on priv chat
this.Events[EVENT.PRIV_CHAT_MSG] = test_func;
this.Events[EVENT.GLOBAL_CHAT_MSG] = test_func;
this.Events[EVENT.TEAM_CHAT_MSG] = test_func;
// on sound
this.Events[EVENT.HEAR_SOUND] = hearsound_func;
// on kill someone
this.Events[EVENT.KILLEDSOMEONE] = killed_someone_func;
// this.Events[EVENT.ADDWEAPON] = receive_weapon_func;
// this.Events[EVENT.FEEL_PAIN] = check_saymedic_func;

// no use for, props set up in this script
// difficulties.InitBotDifficultyProperties(this);

```

# Custom NQ-shrubbot-cmds

From Omni-bot Wiki

## Custom NQ-shrubbot-cmds

### Bot custom command examples that can be used

**[command]**

**command** = addbot

**exec** = bot addbot [1] [2] [3]

**desc** = Adds bot to game. Syntax: !addbot <team> <class> <name>. Don't use whitespaces in <name>.

**levels** = 5

**[command]**

**command** = kickbot

**exec** = bot kickbot [1] [2] [3]

**desc** = Kicks a bot. Syntax: !kickbot <name>.

**levels** = 5

**[command]**

**command** = minbots

**exec** = bot minbots [1]

**desc** = Sets minimum count of bots in game.

**levels** = 5

**[command]**

**command** = maxbots

**exec** = bot maxbots [1]

**desc** = Sets maximum count of bots in game.

**levels** = 5

**[command]**

**command** = kickallbots

**exec** = bot minbots 0; bot maxbot 0; bot kickall

**desc** = Cleans server from bots.

**levels** = 5

Make sure you don't set maxbots/minbots in gm-scripts.



## Bot custom command examples that can be used in combination with scripts

Add script-snippets from page Custom bot command extensions to your <\$install-path>/omni-bot/et/scripts/et\_commands.gm to run these cmds.

**[command]**

**command** = say

**exec** = bot say [1] [2] [3] [4] [5] [6] [7] [8] [9]

**desc** = Lets first bot from table talk inputs.

**levels** = 5

**[command]**

**command** = sayall

**exec** = bot say [1] [2] [3] [4] [5] [6] [7] [8] [9]

**desc** = Lets all bots talk inputs.

**levels** = 5

**[command]**

**command** = botreset

**exec** = bot resetxp

**desc** = Resets XPs from all connected bots.

**levels** = 5

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Custom\\_NQ-shrubbot-cmds](http://omni-bot.com/wiki/index.php?title=Custom_NQ-shrubbot-cmds)"

This page has been accessed 527 times. This page was last modified 06:41, 28 November 2007.

# Custom bot command extensions

From Omni-bot Wiki

## Contents

- 1 Script command extensions
  - 1.1 sayall-function
  - 1.2 say-function
  - 1.3 say-function

## Script command extensions

### Installation

Method 1:

- Add script-snippets from this page to your <\$install-path>/omni-bot/et/scripts/et\_commands.gm file to run these cmds.

Method 2:

- Add ExecScript("et\_commands\_ext.gm") at beginning of your et\_commands.gm and put this file et\_commands\_ext.gm into same path.

Caution: Wiki currently accepts only capital letters at beginning of an uploaded-file. Please rename file 'Et\_commands\_ext.gm' to 'et\_commands\_ext.gm' on linux-/unix-systems.

### sayall-function

Lets all bots talk input.

Example: /bot sayall WE LOVE OMNI-BOT!

```
// > 0.61
Commands["sayall"] = function( _params )
{
    // set color
    out = "^7";
```

```

// we get 9 shrubbot params (1..9)
counter = 0;
while ( counter <= 9 )
{
    if ( _params[counter] != null )
    {
        out += _params[counter];
        out += " ";
    }
    counter +=1;
    yield();
}

foreach ( gameId and bot in BotTable )
{
    bot.Say(out );
}
};

```

## say-function

Lets first bot in table talk inputs.

Example: /bot say I'm not a borg!

```

// > 0.61
// supports up to 9 words/parameters
Commands["say"] = function( _params )
{
    // set color
    out = "^7";
    counter = 0;

    // we get 9 shrubbot params (1..9)
    while (counter <= 9)
    {
        if ( _params[counter] != null )
        {
            out += _params[counter];
            out += " ";
        }
        counter +=1;
        yield();
    }
}

```

```

    }

    foreach (gameId and bot in BotTable )
    {
        bot.Say(out );
        // just first bot in table
        break;
    }
};

```

## say-function

Resets bot-XPs via cmd. Use in combination with shrubbot.

```

// > 0.61 Commands["resetxp"] = function( _params ) {

    foreach (gameId and bot in BotTable )
    {

bot.Say("!resetmyxp");

    }

};

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Custom\\_bot\\_command\\_extensions](http://omni-bot.com/wiki/index.php?title=Custom_bot_command_extensions)"

This page has been accessed 448 times. This page was last modified 11:05, 2 October 2007.

# Difficulty sample.gm

From Omni-bot Wiki

```
// Difficulty_sample.gm Omni-bot Wiki 2007
// 12-10-2007 slightly edited version by Stoned-Aimlezz & Kim Carson.
// This script tested with ET 2.60b + Jaymod 2.1.6 + Omni-bot 0.66.
// Changes : some (re)naming, Difficulty to dif.
// Example of console command : /bot dif one|two|three|four|five|six|uber.
// This script only defines 6 total difficulty levels (one-five + uber).
// It isn't limited to 6 though, and can be easily set up with more.
// General usage :
// Type in console : /bot dif : states the current difficulty level.
// Type in console : /bot dif four : sets difficulty to 'level' four.

print( "Initializing Difficulties" );
// The default starting difficulty
// global (difficulties[CurrentDifficulty]) = three;
global CurrentDifficulty = one;
global AdjustAim = true;
global difficulties =

{
// bot dif one
one =
{
FieldOfView = 90.0,
ReactionTime = 3.0,
MemorySpan = 2.0,
AimPersistance = 1.0,
MaxTurnSpeed = 700.0,
AimStiffness = 75.0,
AimDamping = 11.0,
AimErrorX = 17,
AimErrorY = 17,
AimOffsetX = -3,
AimOffsetY = -11,
AimOffsetZ = -3,
},

// bot dif two
two =
```

```
{  
FieldOfView = 90.0,  
ReactionTime = 3.0,  
MemorySpan = 2.0,  
AimPersistance = 1.0,  
MaxTurnSpeed = 720.0,  
AimStiffness = 75.0,  
AimDamping = 10.0,  
AimErrorX = 15,  
AimErrorY = 15,  
AimOffsetX = 0,  
AimOffsetY = -10,  
AimOffsetZ = 0,  
},
```

```
// bot dif three
```

```
three =  
{  
FieldOfView = 90.0,  
ReactionTime = 2.0,  
MemorySpan = 2.0,  
AimPersistance = 1.0,  
MaxTurnSpeed = 720.0,  
AimStiffness = 75.0,  
AimDamping = 10.0,  
AimErrorX = 10,  
AimErrorY = 10,  
AimOffsetX = 0,  
AimOffsetY = -7,  
AimOffsetZ = 0,  
},
```

```
// bot dif four
```

```
four =  
{  
FieldOfView = 90.0,  
ReactionTime = 1.0,  
MemorySpan = 2.0,  
AimPersistance = 1.0,  
MaxTurnSpeed = 720.0,  
AimStiffness = 75.0,  
AimDamping = 10.0,  
AimErrorX = 7,  
AimErrorY = 7,
```

```
AimOffsetX = 0,  
AimOffsetY = -7,  
AimOffsetZ = 0,  
},
```

```
// bot dif five
```

```
five =  
{  
FieldOfView = 90.0,  
ReactionTime = 0.75,  
MemorySpan = 2.5,  
AimPersistance = 1.5,  
MaxTurnSpeed = 720.0,  
AimStiffness = 75.0,  
AimDamping = 10.0,  
AimErrorX = 5,  
AimErrorY = 5,  
AimOffsetX = 0,  
AimOffsetY = -5,  
AimOffsetZ = 0,  
},
```

```
// bot dif six
```

```
six =  
{  
FieldOfView = 100.0,  
ReactionTime = 0.5,  
MemorySpan = 3.0,  
AimPersistance = 2.0,  
MaxTurnSpeed = 720.0,  
AimStiffness = 75.0,  
AimDamping = 10.0,  
AimErrorX = 5,  
AimErrorY = 5,  
AimOffsetX = 0,  
AimOffsetY = 0,  
AimOffsetZ = 0,  
},
```

```
// bot dif uber
```

```
uber =  
{  
FieldOfView = 105.0,  
ReactionTime = 0.0,
```

```

    MemorySpan = 5.0,
    AimPersistance = 5.0,
    MaxTurnSpeed = 720.0,
    AimStiffness = 75.0,
    AimDamping = 10.0,
    AimErrorX = 0,
    AimErrorY = 0,
    AimOffsetX = 0,
    AimOffsetY = 0,
    AimOffsetZ = 0,
    },
};

// debug display flag
global difficulties_Debug = true;

// A helper function that sets a bot up with the current difficulty settings.
difficulties.InitBotDifficultyProperties = function(bot)
{
    Debug = true;

    if(difficulties[CurrentDifficulty])
    {
        bot.FieldOfView = difficulties[CurrentDifficulty].FieldOfView;
        bot.ReactionTime = difficulties[CurrentDifficulty].ReactionTime;
        bot.MemorySpan = difficulties[CurrentDifficulty].MemorySpan;
        bot.AimPersistance = difficulties[CurrentDifficulty].AimPersistance;
        bot.MaxTurnSpeed = difficulties[CurrentDifficulty].MaxTurnSpeed;
        bot.AimStiffness = difficulties[CurrentDifficulty].AimStiffness;
        bot.AimDamping = difficulties[CurrentDifficulty].AimDamping;

        if ( AdjustAim )

        { difficulties.AdjustWeaponProperties(bot); }

        if ( Debug )

        {
            print("Current Difficulty", CurrentDifficulty);
            print("Difficulty FOV", bot.FieldOfView);
            print("Difficulty ReactionTime",bot.ReactionTime);
            print("Difficulty MemorySpan", bot.MemorySpan);
            print("Difficulty AimPersistance",bot.AimPersistance);
            print("DifficultyMaxTurnSpeed",bot.MaxTurnSpeed);

```



```

        print("Difficulty AimStiffness", bot.AimStiffness);
        print("Difficulty AimDamping", bot.AimDamping);
    }
}

else

{
    print("Invalid Current Difficulty", CurrentDifficulty);
}

};

// A helper function that sets aim offsets for weapons
difficulties.AdjustWeaponProperties = function(bot)
{
    Debug_weaps = false;

    AdjustableWeapons =
    {
        WEAPON.THOMPSON,
        WEAPON.MP40,
        WEAPON.LUGER,
        WEAPON.COLT,
        WEAPON.MOUNTABLE_MG42,
        WEAPON.STEN,
    };
    foreach ( id and weap in AdjustableWeapons )
    {
        w = bot.GetWeapon(weap);
        if ( w )
        {
            errorX = difficulties[CurrentDifficulty].AimErrorX;
            errorY = difficulties[CurrentDifficulty].AimErrorY;
            offsetX = difficulties[CurrentDifficulty].AimOffsetX;
            offsetY = difficulties[CurrentDifficulty].AimOffsetY;
            offsetZ = difficulties[CurrentDifficulty].AimOffsetZ;

            w.PrimaryFire.MaxAimError = Vector2(errorX, errorY);
            w.PrimaryFire.AimOffset = Vector3(offsetX, offsetY, offsetZ);

            if ( Debug_weaps )
            {

```

```
    print(w.Name, "weapon properties set");
    print("offset:", w.PrimaryFire.AimOffset);
    print("error:", w.PrimaryFire.MaxAimError);
}
```

```
//return;
```

```
}
```

```
}
```

```
};
```

```
// Register a command that can be used to change the difficulty in-game
```

```
// Example usage:
```

```
// bot dif two
```

```
// bot dif uber
```

```
// bot dif five
```

```
Commands["dif"] = function(_params)
```

```
{ if(_params[0] != null)
```

```
{
```

```
// Check if this difficulty level exists.
```

```
if(difficulties[_params[0]])
```

```
{
```

```
    global CurrentDifficulty = _params[0];
```

```
    foreach ( gameId and bot in BotTable )
```

```
    {
```

```
        if(bot)
```

```
        {
```

```
            difficulties.InitBotDifficultyProperties(bot);
```

```
        }
```

```
    }
```

```
    print("Difficulty Changed", CurrentDifficulty);
```

```
}
```

```
else
```

```
{
```

```
    print("Difficulty level not found", _params[0]);
```

```
}
```

```
}
```

```
print("Difficulty:", CurrentDifficulty);  
};
```

```
print( "Initializing Difficulties Complete" );
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Difficulty\\_sample.gm](http://omni-bot.com/wiki/index.php?title=Difficulty_sample.gm)"

This page has been accessed 1,127 times. This page was last modified 12:58, 13 October 2007.

# ETPub: Kicking all bots

From Omni-bot Wiki

## Kicking Bots in ETPub

ETPub has a separate cvar setting for bot presence on the server: `g_bot_minPlayers`. If this cvar is not set to 0 in your `etpub.cfg`, kicking bots will always lead to adding new bots after the kick.

### Settings to kick bots

Make sure not to set `MinBots` and `MaxBots` in your `et_autoexec.gm`.

- Add **nobot.cfg** to your `etpub` folder on your server

#### **nobot.cfg**

```
//Nobot.cfg
MinBots(0);
MaxBots(0);
kickall;
set g_bot_minPlayers "0"
```

- Add **!nobot** command to your `shrubbot.cfg`

```
[command]
command = nobot
exec = exec nobot.cfg
desc = Kicks all bots
levels = 5
```

### Settings to re-add bots

- Add **bot.cfg** to your `etpub` folder on your server

#### **bot.cfg**

```
//Bot.cfg
MinBots(0);
MaxBots(X);
```

```
set g_bot_minPlayers "X"
```

- Add **!bot** command to your shrubbot.cfg

**[command]**

**command** = bot

**exec** = exec bot.cfg

**desc** = Adds bots to total the number of **X** players

**levels** = 5

*Where **X** is the number of players on your server, to which total you want bots added.*

Papagali 10:50, 23 January 2008 (MST)

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ETPub:\\_Kicking\\_all\\_bots](http://omni-bot.com/wiki/index.php?title=ETPub:_Kicking_all_bots)"

This page has been accessed 161 times. This page was last modified 17:50, 23 January 2008.

# MG42 Aim Adjustments

From Omni-bot Wiki

DATE: 28 June 2007

BOT VERSION: 0.66

AUTHOR: Crapshoot

DESCRIPTION: This script makes bots less deadly when mounting mg42's

CREDITS: Demetrius for requesting and Bladen for some script inspiration

FILE: mg\_mount.gm

## Installation:

Place mg\_mount.gm in the ~/omni-bot/et/scripts folder

Add ExecScript( "mg\_mount.gm" ); to the top of your def\_bot.gm

Add WatchForMGMount.Begin(this); to the spawn\_func of your def\_bot.gm

## Script

```
if ( WatchForMGMount== null )
{
    global WatchForMGMount =
    {
        Begin = function( bot )
        {
            //don't start if in Limbo or etmain
            if ( (GetEntFlags( bot.GetGameEntity(), ENTFLAG.LIMBO)) )
            {
                return;
            }

            if(bot.WatchForMGThread)
            { threadKill(bot.WatchForMGThread); }

            bot.WatchForMGThread = bot:thread(WatchForMGMount.ThreadFunc);
        },

        ThreadFunc = function()
        {
            if ( !this.BotMGPropertiesSet )
            {
                //print("setting original bot properties");
                this.MaxView = this.MaxViewDistance;
                this.FOV = this.FieldOfView;
                this.Reaction = this.ReactionTime;
                this.Persistance = this.AimPersistance;
                this.Tolerance = this.AimTolerance;
                this.TurnSpeed = this.MaxTurnSpeed;
                this.Stiffness = this.AimStiffness;
                this.Damping = this.AimDamping;
                this.mounted_mg = false;
            }
        }
    }
}
```

```

        this.BotMGPropertiesSet = true;
    }

    while (1)
    {
        if ( this.HasEntityFlag(ENTFLAG.MOUNTED) && !this.mounted_mg )
        {
            //print("setting MG42 properties");
            this.FieldOfView = this.FOV / 2;
            this.ReactionTime = this.Reaction * 2;
            this.AimPersistance = this.Persistance / 5;
            this.AimTolerance = this.Tolerance * 3;
            this.MaxTurnSpeed = this.Turnspeed / 3;
            this.AimStiffness = 25.0;
            this.AimDamping = 5.0;
            this.mounted_mg = true;
        }

        if ( !this.HasEntityFlag(ENTFLAG.MOUNTED) && this.mounted_mg )
        {
            //print("setting original properties");
            this.FieldOfView = this.FOV;
            this.ReactionTime = this.Reaction;
            this.AimPersistance = this.Persistance;
            this.AimTolerance = this.Tolerance;
            this.MaxTurnSpeed = this.TurnSpeed;
            this.AimStiffness = this.Stiffness;
            this.AimDamping = this.

            this.mounted_mg = false;
        }

        if ( this.HasEntityFlag(ENTFLAG.LIMBO) && !this.mounted_mg )
        {
            threadKill(this.WatchForMGThread);
        }

        //properties set and bot still mounted, so script some
additional behaviors

        if ( this.HasEntityFlag(ENTFLAG.MOUNTED) && this.mounted_mg )
        {
            //possibly extend here
        }

        sleep(1.0);
    }
},
};
}

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=MG42\\_Aim\\_Adjustments](http://omni-bot.com/wiki/index.php?title=MG42_Aim_Adjustments)"

This page has been accessed 565 times. This page was last modified 01:05, 25 July 2007.

# Team Door Script

From Omni-bot Wiki

This is a modified version of the disguise goal script by d00d. It contains two different scripted goals: A disguise goal that is meant to make the bots steal uniforms, and a team door goal that is started from the disguise goal thread once the bot is disguised, provided the map script of the current map defines team doors the bots should use.

The team door goal is a slightly modified version of the useswitch goal. It needs at least one named waypoint at the team door, and a Map.Teamdoors table in the map script.

Everything the bots should do after opening the team door goes in the ExitConditions. Please refer to the useswitch documentation.

All this zipped up neatly, with a sample map script for Caen2: caen2\_disguise.zip

## Map table in the map script

Here's an example what a Map table could look like:

```
global Map =
{
    //Table of teamdoors for covert ops
    Teamdoors =
    {
        // Let's use only one teamdoor for now
        teamdoor1 =
        {
            WaypointName = "teamdoor1", //the name of the waypoint at the team door
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            //LimitDistance = 500,
            LimitBots = 1, //only one bot at a time
            //ExitConditions is a table of functions
            //that must return true if the script
            //should release control over the bot:
            ExitConditions =
            {
                lostCover = function(bot)
                {
                    //If we lose disguise, abort.
                    //Shouldn't be necessary ideally, but ...
                    if ( !GetEntFlags(bot.GetGameEntity(), ENTFLAG.DISGUISED) )
                    {
                        return true;
                    }
                },
                stealDocs = function(bot)
                // This is what the bot should normally do after opening the door.
                // Highly map-specific.
                {
```





```

if ( Wp.GetWaypointByName( teamdoorTable.WaypointName,
    teamdoorTable.Waypoint ) )
{
    print( "Teamdoor Data Succeeded, Waypoint:",
        teamdoorTable.WaypointName );

    if ( teamdoorTable.Waypoint.facing.IsZero() )
        { print( "Error: No Facing Defined, Waypoint:",
            teamdoorTable.WaypointName ); }
}
else
{ print( "Error: Teamdoor Data Failed, Waypoint:",
    teamdoorTable.WaypointName ); }
}
},
};

```

## UpdateTeamdoors function

I've put the UpdateTeamdoors function into the Map table because I didn't want to confuse potential users with so many files. In the OnMapLoad function, you have to call this once:

```

global OnMapLoad = function()
{
    Map.UpdateTeamdoors();
};

```

## The Bot Script

Now here's the script for the bots:

```

// This script contains functionality to allow covert ops to steal uniforms
// and open team doors. Tested with OB .66 and ET only.
// It shouldn't harm your bots' functionality if you use this instead of
// the disguise goal that comes with OB.

//=====
// Disguise goal. This will start the team door goal if there are team
// doors configured in the map script.
//=====

// The null check is just in case the script is executed multiple times
if(ScriptGoals.disguise_goal == null)
{
    // For cleanliness, we're going to store all our functions and variables inside a table,
    // within the ScriptGoals global table.
    // The ScriptGoals is a global table created by the bot by default, as a standardized storage
    location
    // for goals implemented in script. Script goals should make their own table under the
    ScriptGoals global table.
}

```

```

ScriptGoals.disguise_goal = table();

// The name for this goal
ScriptGoals.disguise_goal.Name = "steal_uniform";

// A debug flag for enabling/disabling printed messages. Useful for debugging this goal.
ScriptGoals.disguise_goal.Debug = false;

// We'll store the goal check delay for the thread function here, just to put most of the
// configuration options neatly at the top of the script for readability.
ScriptGoals.disguise_goal.ThreadCheckDelay = 2;

//Should return a game entity: the last player this bot has killed
ScriptGoals.disguise_goal.GetLastVictim = function( _bot )
{
    mytarget = _bot.GetTarget(); //get target entity
    if (mytarget == null)
    {
        mytarget = _bot.GetLastTarget();
        //      if (GetTime() > this.lasttargettime+4000)
        //      {
        //          print("^6returning null");
        //          return null;
        //      }
        //print(this.Name, GetEntityName(this.target));
    }
    else
    {
        // we have a fresh target
        this.lasttargettime = GetTime();
        this.lastvictim_invalid == null;
    }
    //print("Target: ",mytarget);
    if (!(mytarget == null))
    {
        //targetname = GetEntityName(mytarget);
        //print("Name:",targetname);
        //print("^6EyPos:",GetEntEyePosition(mytarget));
        healthArmorTable = GetEntHealthAndArmor( mytarget );
        if ( !healthArmorTable || (healthArmorTable.Health > 0) || GetEntEyePosition(mytarget)
==null )
        {
            //print("^6",targetname,"^6<- ZERO HEALTH!!!!!!!!!!!!!!");
            //print("^6EyPos:",GetEntEyePosition(mytarget));
            mytarget = null;
        }
    }
    //print("^1returning",mytarget);
    return mytarget;
};

// This function just holds the functionality to search a corpse table for the closest one.
// It returns a gameentity
ScriptGoals.disguise_goal.LookForClosestCorpse = function( _bot )

```

```

{
    // Get a table containing all players and corpses.
    corpseList = table();
    //_bot.GetAllType( CAT.PLAYER, CLASS.ANYPLAYER, corpseList );
    _bot.GetAllType( CAT.MISC, CLASS.CORPSE, corpseList );

    // Some debug messages
    if( ScriptGoals.disguise_goal.Debug )
    {
        //print("tableCount(corpseList) before health check", tableCount(corpseList));
    }

    botPos = _bot.GetPosition();

    nearestCorpse = null;
    nearestCorpseDist = 10000;

    // Filter out all players that are alive, or on the other team.
    foreach ( index and entity in corpseList )
    {
        if( ScriptGoals.disguise_goal.Debug )
        {
            DrawEntityAABB ( entity, COLOR.RED );
        }

        // What's their current health/armor.
        healthArmorTable = GetEntHealthAndArmor( entity );

        // Skip invalid corpses.
        if( ( healthArmorTable.Health <= 0 ) &&
            !GetEntFlags( entity, ENTFLAG.DISABLED ) &&
            !_bot.IsAllied( entity ) )
        {
            entPos = GetEntPosition( entity );

            distSq = (entPos - botPos).LengthSq();
            if( nearestCorpse == null || distSq < nearestCorpseDist )
            {
                nearestCorpse = entity;
                nearestCorpseDist = distSq;
            }
        }
    }

    // Some debug messages
    if( ScriptGoals.disguise_goal.Debug )
    {
        print("tableCount(corpseList) after health check", tableCount(corpseList));
    }

    return nearestCorpse;
};

// This function should be called when the goal is finished so that it can do cleanup.
ScriptGoals.disguise_goal.LookForDisguiseGoalThread = function()

```

```

{
    // Check if the bot already has this goal.
    if(this.WatchForDisguiseThread)
    {
        threadKill(this.WatchForDisguiseThread);
    }
    this.WatchForDisguiseThread = threadId();

    // The loop condition can check the class, so if the bot changes class this script goal
    will end cleanly.
    while( this.GetClass() == CLASS.COVERTOPS )
    {
        // Some debug messages
        if( ScriptGoals.disguise_goal.Debug )
        {
            print("ScriptGoals.disguise_goal");
        }

        // Check if this bot is already disguised. If so, we can skip a big section of this code.
        botEnt = this.GetGameEntity();
        if( !GetEntFlags( botEnt, ENTFLAG.DISGUISED ) && !GetEntFlags( botEnt, ENTFLAG.
CARRYINGGOAL )
            && !this.HasTarget() )
        {
            // Gib our last victim and steal his uniform.
            // Check if lastVictim entity is still in the game.
            lastVictim = ScriptGoals.disguise_goal.GetLastVictim( this );
            if (lastVictim)
            {
                healthArmorTable = GetEntHealthAndArmor( lastVictim );
                // If so, tapped out / revived / too far? Set to null if yes and skip the rest.
                if ( !healthArmorTable || healthArmorTable.Health > 0 || GetEntFlags(lastVictim,
ENTFLAG.LIMBO) || this.IsAllied(lastVictim) || this.DistanceTo(lastVictim) > 400 )
                {
                    lastVictim = null;
                }
            }
        }

        if( lastVictim )
        {
            if( ScriptGoals.disguise_goal.Debug )
            {
                // Some debug messages
                //print("^6Last Victim returned:",lastVictim);
                //print("Heading to last victim.", lastVictim);
            }

            // Set a script goal for this function, so other script goals can not attempt to
            // take control.
            this.scriptgoal = ScriptGoals.disguise_goal.Name;

            // Go to the goal.

            corpsePosition = GetEntPosition( lastVictim );
            if(!corpsePosition){ break; }

```

```

this.SetScriptControlled( true );
this.start = GetTime();
this.GoTo( corpsePosition, Vector3(0,0,0) );
if( block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED) == EVENT.GOAL_SUCCESS )
{
    // Some debug messages
    if( ScriptGoals.disguise_goal.Debug )
    {
        print("^6Attempting to gib", lastVictim);
    }
    // Take control of the weapons now.
    this.SetScriptControlledWeapons( true );
    this.SelectWeapon(WEAPON.KNIFE);
    if (GetTime() > this.start+10000)
    {
        print("^6timer expired");
        this.scriptgoal = null;
        this.SetScriptControlled( false );
        this.SetScriptControlledWeapons( false );
    }

    // Keep doing this until the victim is gibbed
    this.now = GetTime();
    dowhile ( !GetEntFlags( lastVictim, ENTFLAG.LIMBO ) && !GetEntFlags( botEnt,
ENTFLAG.CARRYINGGOAL )
        && !this.HasTarget() && this.DistanceTo(lastVictim) < 100 && GetTime() < this.
now+5000 )
    {
        if( ScriptGoals.disguise_goal.Debug )
        {
            DrawEntityAABB ( lastVictim, COLOR.RED );
        }

        pos = GetEntPosition(lastVictim);
        this.MoveTowards( pos );
        this.TurnToPosition( pos );

        this.PressButton( BTN.ATTACK1 );
        yield();
    }
}
else
{
    //this.lastvictim_invalid = true;
    this.scriptgoal = null;
    this.SetScriptControlled( false );
    this.SetScriptControlledWeapons( false );
}
// Now the victim should be gibbed.
// Try to steal the uniform until we're disguised or 5 seconds are elapsed
if ( GetEntFlags( lastVictim, ENTFLAG.LIMBO ))
{
    this.now = GetTime();
    //print("trying to steal uniform");
}

```

```

        dowhile ( !GetEntFlags( botEnt, ENTFLAG.DISGUISED ) && !GetEntFlags( botEnt,
ENTFLAG.CARRYINGGOAL )
            && !this.HasTarget() && GetTime() < this.now+5000)
        {
            if( ScriptGoals.disguise_goal.Debug )
            {
                DrawEntityAABB ( lastVictim, COLOR.RED );
            }

            this.MoveTowards( pos );
            this.TurnToPosition( pos );

            this.PressButton( BTN.USE );
            yield();
        }
    }
else
{
    this.scriptgoal = null;
    this.SetScriptControlled( false );
    this.SetScriptControlledWeapons( false );
}
if( GetEntFlags( botEnt, ENTFLAG.DISGUISED) )
{
    // Some debug messages
    this.SayTeam("Ready to infiltrate!");
    if( ScriptGoals.disguise_goal.Debug )
    {
        print("^6Stole uniform.");
    }
    this.SelectBestWeapon();
    this.SetScriptControlledWeapons( false );
    this.SetScriptControlled( false );
    this.scriptgoal = null;
    //Start the Teamdoor thread if there are any teamdoors in the map script
    if (Map.Teamdoors)
    {
        ScriptGoals.Teamdoor.Begin(this);
    }
}
else
{
    this.scriptgoal = null;
    this.SetScriptControlled( false );
    this.SetScriptControlledWeapons( false );
}
}

if( lastVictim == null)
{
    nearestCorpse = ScriptGoals.disguise_goal.LookForClosestCorpse( this );
}

if( nearestCorpse )
{

```

```

// Some debug messages
if( ScriptGoals.disguise_goal.Debug )
{
    print("Heading to a corpse.", nearestCorpse);
}

// Set a script goal for this function, so other script goals can not attempt to
// take control.
this.scriptgoal = ScriptGoals.disguise_goal.Name;

// Go to the goal.
corpsePosition = GetEntPosition( nearestCorpse );

this.SetScriptControlled( true );
this.GoTo( corpsePosition, Vector3(0,0,0) );
if( block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED) == EVENT.GOAL_SUCCESS )
{
    // Some debug messages
    if( ScriptGoals.disguise_goal.Debug )
    {
        print("^6Attempting to steal uniform.", nearestCorpse);
    }

    // Take control of the weapons now.
    this.SetScriptControlledWeapons( true );

    // Keep doing this until we're disguised
    dowhile ( !GetEntFlags( botEnt, ENTFLAG.DISGUISED ) && !GetEntFlags( botEnt,
ENTFLAG.CARRYINGGOAL )
        && !this.HasTarget() /* && !GetEntFlags( nearestCorpse, ENTFLAG.DISABLED )*/ )
    {
        if( ScriptGoals.disguise_goal.Debug )
        {
            DrawEntityAABB ( nearestCorpse, COLOR.RED );
        }

        this.MoveTowards( corpsePosition );
        this.TurnToPosition( corpsePosition );

        this.PressButton( BTN.USE );
        yield();

        if ( GetEntPosition(nearestCorpse) == null || this.DistanceTo(GetEntPosition
(nearestCorpse)) >30 )
        {
            this.now = GetTime();
            dowhile(GetTime() < this.now+2000)
            {
                this.MoveTowards( corpsePosition );
                this.TurnToPosition( corpsePosition );
                this.PressButton( BTN.USE );
                yield();
            }
            //print("^6Aborted due to pos.");
            break;

```



```

    }
    if ( !GetEntFlags( botEnt, ENTFLAG.DISGUISED ) && GetEntFlags( nearestCorpse,
ENTFLAG.DISABLED ) )
    {
        //print("^6What?! Coulda sworn ... Did someone beat me to it? Second try.");
        this.now = GetTime();
        dowhile(GetTime() < this.now+2000 && !GetEntFlags( botEnt, ENTFLAG.DISGUISED ) )
        {
            this.MoveTowards( corpsePosition );
            this.TurnToPosition( corpsePosition );
            this.PressButton( BTN.USE );
            yield();
        }
        //print("^6Giving up.");
        break;
    }
    yield();
}

if( GetEntFlags( botEnt, ENTFLAG.DISGUISED ) )
{
    // Some debug messages
    this.SayTeam("Ready to infiltrate!");
    if( ScriptGoals.disguise_goal.Debug )
    {
        print("^6Stole uniform.");
    }
    if (Map.Teamdoors)
    {
        ScriptGoals.Teamdoor.Begin(this);
    }
}

// Null the script goal, so other script goals can possibly use this bot.
this.scriptgoal = null;
this.SetScriptControlledWeapons( false );
this.SetScriptControlled( false );
}
}
else
{
    // We are disguised, so do whatever.
}

sleep( ScriptGoals.disguise_goal.ThreadCheckDelay );
}

// Clear the goal on the way out.
if( this.scriptgoal == ScriptGoals.disguise_goal.Name )
{
    this.scriptgoal = null;
}
};
}

```

```

//=====
// The team door goal.
//=====
ScriptGoals.Teamdoor =
{
    // The name for this goal
    Name = "teamdoor",

    // A debug flag for enabling/disabling printed messages. Useful for debugging this goal.
    Debug = false,
    DebugScriptGoal = false,

    // We'll store the goal check delay for the thread function here, just to put most of the
    // configuration options neatly at the top of the script for readability.
    ThreadCheckDelay = 3,

    Begin = function(bot)
    {
        yield();
        if ( GetEntFlags(bot.GetGameEntity(), ENTFLAG.LIMBO ))
        {
            return;
        }

        //ScriptGoals.Teamdoor.Finish(bot);

        while ( bot.scriptgoal == ScriptGoals.dispense_ammo.Name ||
            bot.scriptgoal == ScriptGoals.dispense_health.Name )
            { yield(); }

        // Make sure there are some team doors configured.
        if(!Map || !Map.Teamdoors)
        {
            print("^1Teamdoor goal error: No Teamdoor table in global Map table.");
            return;
        }

        foreach ( i and teamdoorTable in Map.Teamdoors )
        {
            if ( !Map.Teamdoors[ i ].LimitBots )
                { Map.Teamdoors[ i ].LimitBots = 2; }
        }
        print("Teamdoor.Begin, existing", bot.TeamdoorThread);
        if(bot.TeamdoorThread)
        {
            print("Killing Thread.", bot.TeamdoorThread);
            threadKill(bot.TeamdoorThread);
        }
        bot.TeamdoorThread = bot:thread(ScriptGoals.Teamdoor.ThreadFunc);

        print("Creating Thread.", bot.TeamdoorThread);
    },

    ThreadFunc = function()
    {

```

```

//setFinalFunction(ScriptGoals.Teamdoor.Finish);

firstCall = true;

while( true )
{
    if( this.scriptgoal == null )
    {
        myTeamMask = (1<<this.GetTeam());

        foreach ( i and teamdoorTable in Map.Teamdoors )
        {
            myBots = Util.BotsWithGoal( ScriptGoals.Teamdoor.Name );

            if( ( teamdoorTable.LimitTeam & myTeamMask ) &&
                ( myBots < Map.Teamdoors[ i ].LimitBots ) &&
                ( !GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO) ) &&
                ( !GetEntFlags(this.GetGameEntity(), ENTFLAG.CARRYINGGOAL) ) &&
                ( GetEntFlags(this.GetGameEntity(), ENTFLAG.DISGUISED) ) )
            {
                if ( teamdoorTable.UseOnce )
                {
                    //create or clear the table when first called (each spawn)
                    if (!teamdoorTable[ this ] || firstCall )
                    {
                        teamdoorTable[ this ] = table();
                        firstCall = false;
                    }

                    if ( teamdoorTable[ this ].Used )
                    {
                        if ( ScriptGoals.Teamdoor.Debug || teamdoorTable.Debug )
                        { print(this.Name, "^2has already used", teamdoorTable.WaypointName,
"skipping"); }

                        continue;
                    }
                }

                radius = teamdoorTable.Waypoint.radius;
                wpPos = teamdoorTable.Waypoint.position;
                wpFacing = teamdoorTable.Waypoint.facing;

                if ( teamdoorTable.LimitDistance > 0 )
                {
                    dist = this.DistanceTo(wpPos);

                    if ( ScriptGoals.Teamdoor.Debug || teamdoorTable.Debug )
                        { print(this.Name, "^2is", dist, "away from", teamdoorTable.WaypointName); }

                    if ( dist > teamdoorTable.LimitDistance )
                        { continue; }
                }

                this.SetScriptControlled(true);
            }
        }
    }
}

```

```

this.EnableShooting(false);
this.scriptgoal = ScriptGoals.Teamdoor.Name;

if ( ScriptGoals.Teamdoor.Debug || teamdoorTable.Debug )
    { print("^6", this.Name, "is going to", teamdoorTable.WaypointName); }

if (teamdoorTable.Routel && this.DistanceTo(wpPos)>4000)
{
    routelinfo = table();
    wp1 = Wp.GetWaypointByName(teamdoorTable.Routel, routelinfo);
    if (!routelinfo.facing)
    {
        routelinfo.facing = Vector3(0,0,0);
    }
    assert(routelinfo.position && routelinfo.facing);
    this.GoTo(routelinfo.position, routelinfo.facing);
    if( block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED) == EVENT.GOAL_SUCCESS )
    {
        print("Made it to Routel!");
    }
}

assert(radius && wpPos && wpFacing);
this.GoTo(wpPos, wpFacing);
if( block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED) == EVENT.GOAL_SUCCESS )
{
    this.SetDesiredFacing(wpFacing);

    counter = 0;
    exitCondition = false;
    while( (teamdoorTable.LimitTeam & myTeamMask) && !exitCondition )
    {
        this.MoveTowards( wpPos, radius );
        if( this.TurnToFacing(wpFacing) )
        {
            if(counter & 2)
            {
                this.PressButton( BTN.USE );

                if ( teamdoorTable.ExitConditions )
                {
                    foreach ( id and _function in teamdoorTable.ExitConditions )
                    {
                        if ( _function(this) )
                        {
                            if ( ScriptGoals.Teamdoor.Debug || teamdoorTable.Debug )
                                { print(this.Name, "^2exit condition met for", teamdoorTable.
WaypointName); }

                            if ( teamdoorTable.UseOnce )
                                { teamdoorTable[ this ].Used = true; }
                            exitCondition = true;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    counter += 1;
    yield();
}

ScriptGoals.Teamdoor.Finish(this);
break;
}
else
{
    ScriptGoals.Teamdoor.Finish(this);
    break;
}

if ( GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO ) )
{
    ScriptGoals.Teamdoor.Finish(this);
}
}
yield();
}
}

sleep( ScriptGoals.Teamdoor.ThreadCheckDelay );
}

ScriptGoals.Teamdoor.Finish(this);
},

Finish = function(bot)
{

    if( bot && bot.scriptgoal == ScriptGoals.Teamdoor.Name )
    {
        print("^6Bot set non-script-controlled:", bot.Name);
        bot.SetScriptControlled(false);
        bot.EnableShooting(true);
        bot.scriptgoal = null;
    }
    else if ( bot && ScriptGoals.Teamdoor.DebugScriptGoal )
    {
        if ( bot.IsScriptControlled() )
        {
            displaytime = 5;
            playerEnt = Util.GetPlayerEntity();

            output = bot.Name + " ^1Teamdoor Finish Error: mismatched ScriptGoal ";
            EchoToScreen( playerEnt, displaytime, output );
            print(bot.Name, "^1Teamdoor Finish Error: mismatched ScriptGoal");
            print(bot.Name, "scriptgoal =", bot.scriptgoal);
        }
    }
}

```

```
if( this && this.scriptgoal == ScriptGoals.Teamdoor.Name )
{
    this.SetScriptControlled(false);
    this.scriptgoal = null;
}
},
};

if( !Map )
{
    global Map = table();
}
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Team\\_Door\\_Script](http://omni-bot.com/wiki/index.php?title=Team_Door_Script)"

This page has been accessed 274 times. This page was last modified 20:41, 2 August 2007.

# Voice Chat Script

## From Omni-bot Wiki

Those who have been with Omni-bot a little longer might find this reminiscent of what used to be the default behavior in earlier versions. I don't remember which versions I'm talking about here, maybe 0.231. The bots used to talk a lot then. In these olden days, the bots' aim was extremely poor when they used grenades. It was very common then that a bot threw a grenade at himself, shouted "Fire in the hole!", got hurt, shouted "Hold your fire!" (because being a teammate is a reflexive relation), "Taking fire!", and eventually "Need a medic!" Though this was slightly funny, it soon got annoying. I hope I have removed most of the annoying parts of this, and kept the fun.

Download: [Voice\\_chat\\_script.gm](#)

```
//=====
// Talking Bot Profile for Omni-bot 0.66
// by Jan Schreiber (d00d).

// This is not a part of the official Omni-bot distribution.
// Errors in this file should be (please) reported to user
// d00d via PM or whatever means of communication seems appropriate.
// They shouldn't be considered bugs in the Omni-bot core;
// the Omni-bot dev team is in no way responsible for the contents of
// this file. Please don't use the bug report forum to discuss any
// problems you encounter with this script.

// Thanks to: crapshoot, Bladen, Jaskot, DrEvil, IRATA [*] and many
// others I fail to remember atm.

// Based on various sources provided by the Omni-bot community, including
// Bladen's scripts and a forum thread about a Chuck Norris bot:
// http://www.omni-bot.de/e107/e107_plugins/forum/forum_viewtopic.php?6918

// This will most likely work with 0.65, but has been reported to crash 0.61!
// All care has been taken to make this work without unwanted side effects,
// but you know how it is ...

// This is a JUST FOR FUN script!
// If you are seriously concerned about cheating, you'd better not use it--
// or only selected portions of it, then it's your own fault. ;-P
// Actually, you'd better use no bots at all then, for that matter, because
// bots can easily make a map imbalanced, depending on the quality of the
// scripts, waypoints, and various other parameters.
// There are no agreed-upon standards how to assess bot usage in competitive
// environments, AFAIK, so better leave bots out of this.
// This script is by no means intended as a cheat or anything close to a cheat,
// but it may well use techniques that are considered dubious by some players.
// Bots escorting players and the disguised warning used here are potentially
// critical. Use responsibly, and with common sense.

// To do:
```

```

// =====
// - Obviously, this is almost impossible to maintain as it stands.
// DEFINITELY NEEDS to be modularized somehow.
// - The "Feel pain" function is apparently based on a totally wrong
// assumption. Doesn't do much harm, but still ...
//=====

// How many bots are escorting a player who has
// called for cover, backup, or asked to follow him:
global voice_escortbots = 0; //DO NOT EDIT! (initialization)
// Load some scripts we might use.
// ExecScript( "health_goal.gm" );
// ExecScript( "ammo_goal.gm" );
ExecScript( "disguise_goal.gm" );
ExecScript( "mountvehicle_goal.gm" );
ExecScript( "dispenseammo_goal.gm" );
ExecScript( "dispensehealth_goal.gm" );
ExecScript( "escortvehicle_goal.gm" );
ExecScript( "ridevehicle_goal.gm" );
ExecScript( "useswitch_goal.gm" );
ExecScript( "poisoned.gm" );

// Various bot properties
// FieldOfView is the angle (in degrees) that the bots can 'see' in front of them.
this.FieldOfView = 130.0;

// ReactionTime is the time delay (in seconds) from when a bot first see's a target, to when
// the bot will begin to react and target them.
this.ReactionTime = 0.0;

// MemorySpan is how long it takes (in seconds) for a bot to consider his memory of someone or
something
// 'out of date' and not considered for targeting and such
this.MemorySpan = 4.0;

// AimPersistance is how long the bot will aim in the direction of a target after the target
has gone out of view.
// This is useful for keeping the bot aiming toward the target in the event of brief
obstructions of their view.
this.AimPersistance = 3.0;

// MaxViewDistance is the maximum distance(in game units) the bot is capable of seeing
something.
// This could be tweaked lower for maps with fog or for a closer to human view distance
// Typically this value is best set in the map script in the OnBotJoin callback.
//this.MaxViewDistance = 10000.0;

// These 3 values are aim properties. Care must be taken when tweaking aim properties, since
// improper values can produce aim oscillations and hurt the bots combat abilities.
this.MaxTurnSpeed = 720.0; // degree's / second
this.AimStiffness = 75.0;
this.AimDamping = 10.0;

// Leave server when team-killed by the same player too often.

```



```

this.leaveWhenTK = true;
//Number of deaths by a sniper rifle for this bot (initialization)
this.sniper_deaths = 0;

// don't accuse players of team kill when this bot was killed by any of these:
this.noblameweapons =
{
    "MOD_SATCHEL",
    "MOD_MORTAR",
    "MOD_GRENADE",
    "MOD_AIRSTRIKE",
    "MOD_ARTY",
    "MOD_DYNAMITE",
    "MOD_WORLD",
    "MOD_LANDMINE",
    "MOD_EXPLOSIVE",
    "MOD_CONSTRUCTION",
    "MOD_CRUSH_CONSTRUCTION",
    "MOD_CRUSH",
    "MOD_TRIGGER_HURT",
    "MOD_WATER",
    "MOD_FALLING",
};

// check ammo status of these weapons:
this.checkammoweapons =
{
    WEAPON.THOMPSON,
    WEAPON.MP40,
    WEAPON.STEN,
    WEAPON.LUGER_AKIMBO,
    WEAPON.LUGER_AKIMBO_SILENCED,
    WEAPON.LUGER,
    WEAPON.COLT,
    WEAPON.COLT_AKIMBO,
    WEAPON.COLT_AKIMBO_SILENCED,
    //WEAPON.AXIS_GRENADE,
    //WEAPON.ALLY_GRENADE,
    WEAPON.FG42_SCOPE,
    WEAPON.FG42,
    WEAPON.K43,
    WEAPON.K43_SCOPE,
    WEAPON.KAR98,
};

// remain silent (no "Taking fire!" etc.)
// when killed by any of these:
this.nocryweapons =
{
    "MOD_SATCHEL",
    "MOD_MORTAR",
    "MOD_ARTY",
    "MOD_DYNAMITE",
    "MOD_WORLD",
    "MOD_LANDMINE",
};

```

```

    "MOD_EXPLOSIVE" ,
    "MOD_CONSTRUCTION" ,
    "MOD_CRUSH_CONSTRUCTION" ,
    "MOD_CRUSH" ,
    "MOD_TRIGGER_HURT" ,
    "MOD_WATER" ,
    "MOD_FALLING" ,
};

// When attacked by a disguised enemy, do NOT warn team mates  by saying
// "enemy in disguise" if the attacker used any of these:
this.nodisguisewarnweapons =
{
    "MOD_SATCHEL" ,
    "MOD_K43_SCOPE" ,
    "MOD_FG42SCOPE" ,
    "MOD_GARAND_SCOPE" ,
    "MOD_WORLD" ,
    "MOD_LANDMINE" ,
    "MOD_EXPLOSIVE" ,
};

// no "great shot" vsay if killed by:
this.nopraiseweapons =
{
    "MOD_PANZERFAUST" ,
    "MOD_FLAMETHROWER" ,
    "MOD_BROWNING" ,
    "MOD_MACHINEGUN" ,
    "MOD_SATCHEL" ,
    "MOD_MORTAR" ,
    "MOD_GRENADE" ,
    "MOD_GRENADE_LAUNCHER" ,
    "MOD_AIRSTRIKE" ,
    "MOD_ARTY" ,
    "MOD_DYNAMITE" ,
    "MOD_WORLD" ,
    "MOD_MINE" ,
    "MOD_LANDMINE" ,
    "MOD_EXPLOSIVE" ,
    "MOD_CONSTRUCTION" ,
    "MOD_CRUSH_CONSTRUCTION" ,
    "MOD_CRUSH" ,
    "MOD_TRIGGER_HURT" ,
    "MOD_WATER" ,
    "MOD_FALLING" ,
};

// don't say a oneliner after killing somebody with these:
this.notauntweapons =
{
    "MOD_BROWNING" ,
    "MOD_MACHINEGUN" ,
    "MOD_SATCHEL" ,
    "MOD_MORTAR" ,

```

```

    "MOD_DYNAMITE" ,
    "MOD_WORLD" ,
    "MOD_MINE" ,
    "MOD_LANDMINE" ,
    "MOD_EXPLOSIVE" ,
    "MOD_CONSTRUCTION" ,
    "MOD_CRUSH_CONSTRUCTION" ,
    "MOD_TRIGGER_HURT" ,
    "MOD_WATER" ,
    "MOD_CRUSH" ,
    "MOD_FALLING" ,
};

// don't say sorry after killing a teammate with these:
this.nosorryweapons =
{
    "MOD_MORTAR" ,
    "MOD_DYNAMITE" ,
    "MOD_WORLD" ,
    "MOD_MINE" ,
    "MOD_LANDMINE" ,
    "MOD_EXPLOSIVE" ,
    "MOD_CONSTRUCTION" ,
    "MOD_CRUSH_CONSTRUCTION" ,
    "MOD_CRUSH" ,
    "MOD_TRIGGER_HURT" ,
    "MOD_WATER" ,
};

// team chat thanks
// Chat messages to which the bot should reply with
// "You're welcome" etc.
this.teamchatthanks =
{
    "Thanks" ,
    "Thank you" ,
    "Thank you." ,
    "Thank you!" ,
    "THANK YOU!" ,
    "Thanks!" ,
    "Thanks." ,
    "THANKS!" ,
    "thanks!" ,
    "thanks" ,
    "thnaks" ,
    "thx" ,
    "thx." ,
    "thx!" ,
    "THX" ,
    "THX!" ,
    "Thx, m8!" ,
    "thx m8" ,
    "thx m8!" ,
    "Thx m8!" ,
    "Thx, mate!" ,

```

```

"Thanks, mate!",
"Thanks mate!",
"Thanks mate",
"thanks mate",
"Danke, Kamerad!",
"danke",
"danke!",
"danke!!",
"danke!!!",
"Danke",
"Danke!",
"Danke!!",
"Danke!!!",
"DANKE!!!",
"DANKE!!",
"DANKE!",
"DANKE",
};

// Keep track of team killers in a table.
// Might be interesting.
// Not exactly sure what to do with it though.
// Currently only used for shouting "That was the 5th time" etc.,
// and leaving the server if a single player "specializes"
// on tk'ing bots.
this.teamkilltable = table({entity="",name="",kills=0});

// Randomly say some nonsense, especially when killing someone.
// Feel free to add a lot of customized messages here.
this.oneliners =
{
    "Oh, I'm sorry. Did I break your concentration?",
    "We should have shotguns for this.",
    "Shocking! Positively shocking!", //this is from an early James Bond movie ;-)
    "Hasta la vista, baby!",
    "You missed, Mr. Bond!",
    "Good bye, Mr. Bond!",
    "I like big fat men like you. When they fall they make more noise!",
    "Never send a human to do a machine's job.",
    "Dodge this!",
    "Zed's dead, baby, Zed's dead.",
    "He, he! pwnage!",
    "Call in Aunt Hilary!",
    "Ze enemy is vegan!",
    "Heh heh heh -- right in the butt. That was great.",
    "You know that hard look I get in my eyes? They saw that and ran like school girls with
tails between their legs.",
    "I thought I had an appetite for destruction, but all I wanted was a club sandwich.",
    "I don't have to be careful. I got a gun.",
    "Oh, they're defending themselves somehow!",
    "Shhhhhh. Be vewwwy qwiet ... I'm hunting wabbits.",
    "Need more cannon-food!",
    "Ya look like swiss cheese!",
    "^1BUAHAAAAHARRR!",
};

```

```

// Auxiliary function for kicking bots safely (DrEvil)
global AsyncKickBot = function(bot)
{
    kickfunc = function(bot)
    {
        yield();
        KickBot(bot.Name);
    };
    thread(kickfunc, bot);
};

////////////////////////////////////

// Utility Callbacks
this.SelectTeam = function()
{
    return null;
};

this.SelectClass = function()
{
    return null;
};

////////////////////////////////////

global ClassPrimaryWeaponChoice = function(bot)
{
    myteam = bot.GetTeam();
    myclass = bot.GetClass();

    if(GetModName() == "noquarter")
    {
        if ( myclass == CLASS.SOLDIER )
        {
            if ( myteam == TEAM.ALLIES )
            {
                wpns =
                {
                    WEAPON.VENOM,
                    WEAPON.BAZOOKA,
                    WEAPON.MOBILE_BROWNING,
                    WEAPON.FLAMETHROWER,
                };
                return wpns[ RandInt( 0, 3 ) ];
            }
        }
        else
        {
            wpns =
            {
                WEAPON.VENOM,
                WEAPON.PANZERFAUST,
                WEAPON.MOBILE_MG42,
                WEAPON.FLAMETHROWER,
            };
        }
    }
}

```

```

        return wpns[ RandInt( 0, 3 ) ];
    }
}
}
else if(GetModName() == "jaymod")
{
    if ( myclass == CLASS.SOLDIER )
    {
        wpns =
        {
            WEAPON.PANZERFAUST,
            WEAPON.MOBILE_MG42,
            WEAPON.FLAMETHROWER,
            //WEAPON.MORTAR // dont choose this for now
        };
        return wpns[ RandInt( 0, 2 ) ];
    }
}
else // etmain
{
    if ( myclass == CLASS.SOLDIER )
    {
        wpns =
        {
            WEAPON.PANZERFAUST,
            WEAPON.MOBILE_MG42,
            WEAPON.FLAMETHROWER,
            //WEAPON.MORTAR // dont choose this for now
        };
        return wpns[ RandInt( 0, 2 ) ];
    }
}
return null;
};

global ClassSecondaryWeaponChoice = function(bot)
{
    return null;
};

this.SelectWeapons = function()
{
    myteam = this.GetTeam();
    myclass = this.GetClass();

    //~ print("SelectWeapons: Class", myclass, "Team", myteam);

    weaponSelection = {};
    weaponSelection.Primary = ClassPrimaryWeaponChoice(this);
    weaponSelection.Secondary = ClassSecondaryWeaponChoice(this);

    if ( weaponSelection.Primary )
    { this.ChangePrimaryWeapon( weaponSelection.Primary ); }

    if ( weaponSelection.Secondary )

```

```

    { this.ChangeSecondaryWeapon( weaponSelection.Secondary ); }
};

////////////////////////////////////

global saySkills = function()
{
    this.mySkills = table();
    if(this.GetSkills(this.mySkills))
    {
        this.Say("Battle Sense: ", this.mySkills[SKILL.BATTLE_SENSE]);
        this.Say("Engineering: ", this.mySkills[SKILL.ENGINEERING]);
        this.Say("First Aid: ", this.mySkills[SKILL.FIRST_AID]);
        this.Say("Signals: ", this.mySkills[SKILL.SIGNALS]);
        this.Say("Lght Wpns: ", this.mySkills[SKILL.LIGHT_WEAPONS]);
        this.Say("Hvy Wpns: ", this.mySkills[SKILL.HEAVY_WEAPONS]);
        this.Say("Cov Ops: ", this.mySkills[SKILL.COVERTOPS]);
    }
    else
    {
        this.Say("Don't know my skills!");
    }
};

// The following is executed on each spawn
spawn_func = function()
{
    // clear any previously scripted goals.
    this.scriptgoal = null;
    this.scriptedgoal = null;
    this.scriptedeskort = null;
    this.EnableMovement(true);
    this.SetScriptControlled(false);
    this.SetScriptControlledWeapons(false);

    // Save the spawn ("home") position
    this.HomePos = this.GetPosition();

    this:thread(ScriptGoals.mount_vehicle.watchForMountableVehicles);
    this:thread(ScriptGoals.escort_vehicle.watchForEscortableVehicles);
    ScriptGoals.RideVehicle.Begin(this);
    ScriptGoals.UseSwitch.Begin(this);
    WatchForPoison.Begin(this);

    // Sample Usage of health_goal.gm
    //this:thread(ScriptGoals.watch_health.watchHealthFunc);

    //some class specific behavior on spawn follows:

    if(this.GetClass() == CLASS.COVERTOPS)
    {
        //This might help at times:
        //print(this.Name);

        //Look for corpses to steal uniforms from:

```

```

this:thread(ScriptGoals.disguise_goal.LookForDisguiseGoalThread);

spwnrnd = RandInt(1,200);
if (spwnrnd < 2 && this.GetNearestEnemy(CAT.PLAYER,CLASS.ANYPLAYER) == null)
{
    this.SayVoice(VOICE.ALL_CLEAR);
}
else if (spwnrnd < 3)
{
    this.SayVoice(VOICE.CLEAR_PATH);
}
}

if(this.GetClass() == CLASS.ENGINEER)
{
    //print(this.Name);
    spwnrnd = RandInt(1,100);
    if (spwnrnd < 2)
    {
        this.SayVoice(VOICE.IMA_ENGINEER);
    }
    else if (spwnrnd < 4)
    {
        this.SayVoice(VOICE.MOVE);
    }
    else if (spwnrnd < 5)
    {
        this.SayVoice(VOICE.LETS_GO);
    }
}

if(this.GetClass() == CLASS.SOLDIER)
{
    //print(this.Name);
    spwnrnd = RandInt(1,100);
    if (spwnrnd < 2)
    {
        this.SayVoice(VOICE.IMA_SOLDIER);
    }
    else if (spwnrnd < 4)
    {
        this.SayVoice(VOICE.ON_OFFENSE);
    }
    else if (spwnrnd < 5)
    {
        this.SayVoice(VOICE.ON_DEFENSE);
    }
}

// Check if this bot is a Medic, and drop some Medkits for a few seconds.
if(this.GetClass() == CLASS.MEDIC)
{
    this:thread(ScriptGoals.dispense_health.DispenseHealth);
}

```



```

// Check if this bot is a Field-Ops, and drop some Ammo for a few seconds.
if(this.GetClass() == CLASS.FIELDOPS)
{
    this:thread(ScriptGoals.dispense_ammo.DispenseAmmo);
}

//this.SetDebugFlag(DEBUG.GOALS, true);
};

death_func = function( Inflictor, MeansOfDeath )
{
    //~ The following might be useful for scripters:
    //print("MeansOfDeath:",MeansOfDeath);
    //print("Inflictor (game entity):",Inflictor);
    //print("Inflictor name:",GetEntityName(Inflictor));

    if (this==null || Inflictor==null || MeansOfDeath==null )
    {
        return;
    }

    // In case it was suicide:
    if (Inflictor == this.GetGameEntity())
    {
        //print("^1SUICIDE!");
        if (MeansOfDeath=="MOD_GRENADE" && RandInt(1,10)<4)
        {
            this.Say("Sigh. Some day I'll learn how to use grenades ...");
        }
        if ((MeansOfDeath=="MOD_CRUSH_CONSTRUCTION" || MeansOfDeath=="MOD_CONSTRUCTION") && RandInt
(1,10)<4)
        {
            this.Say("Stop laughing! This is more difficult than you think.");
        }
        else if (RandInt(1,10) < 2)
        {
            this.SayTeam("Oh my, that's embarrassing ...");
        }
        // Leave early:
        return;
    }

    //General team kill check:
    if ( !(this==null) && this.IsAllied(Inflictor) )
    {
        foreach ( weapon in this.noblameweapons )
        // Team kills with some weapons (e.g. dynamite) can't be considered a 'real' team kill ...
        {
            if (weapon == MeansOfDeath)
            {
                // ... so we set our dont_blame_them var here
                dont_blame_them = true;
            }
        }
    }
}

```

```

// get the bastard's name:
infname = GetEntityName(Inflictor);
//print("Team kill Inflictor:", infname);

//Keep track of the number of teamkills for each player in a table
this.found = false;
if ( dont_blame_them == false && infname != null && infname != "tank" && infname != "truck" )
{
    foreach ( teamkiller in this.teamkilltable )
    {
        if ( teamkiller != null )
        {
            if ( !(this==null) && teamkiller.entity == Inflictor )
            {
                this.found = true;
                // increment kills for teamkiller
                teamkiller.kills = teamkiller.kills + 1;
                // update name if they are trying to outsmart us by changing their name: ;-)
                teamkiller.name = infname;
                // Leave server when team-killed by the same player too often,
                // as if protesting against the rough treatment.
                // POTENTIAL SOURCE OF INSTABILITIES!
                // Set this.leaveWhenTK to false if you don't want this.
                rand = RandInt(1,10);
                if (this.leaveWhenTK && teamkiller.kills > 11 && rand < 8)
                {
                    if (rand < 2)
                    {
                        this.SayTeam("So long, suckers!");
                    }
                    else if (rand < 3)
                    {
                        this.SayTeam("I don't play with team killers!");
                    }
                    else if (rand < 4)
                    {
                        this.SayTeam("Bastards! I'll leave!");
                    }
                    else if (rand < 5)
                    {
                        this.SayTeam("Enjoy your death trap, ladies!");
                    }
                    else if (rand < 6)
                    {
                        this.SayTeam("Wait a minute, there's something bothering me about this place.");
                        this.SayTeam("I know! This lesbian bar doesn't have a fire exit!");
                        this.SayTeam("Enjoy your death trap, ladies!");
                    }
                    else if (rand < 7)
                    {
                        this.SayTeam("You can't fire me, I quit!");
                    }
                    if (rand < 6)
                    {
                        this.SayVoice(VOICE.G_BYE);
                    }
                }
            }
        }
    }
}

```

```

    }
    sleep(1);
    // Do NOT use the following:
    //this.ExecCommand("disconnect");
    // DrEvil's AsyncKickBot has so far proven a safe way to kick a bot
    // from "outside" its own code:
    AsyncKickBot(this);
    sleep(2);
}
if (this == null) { return; }
if (!this == null && teamkiller.kills > 3 && RandInt(1,10) < 6)
{
    tmpStr = infname+"^1, that was the "+teamkiller.kills+"th time!";
    this.SayTeam(tmpStr);
}
else if (!this == null && teamkiller.kills > 6 && RandInt(1,10) < 9)
{
    tmpStr = "^1STOP IT, WILL YOU!!!!";
    this.SayTeam(tmpStr);
}
else if (!this == null && teamkiller.kills > 7 && RandInt(1,10) == 10)
{
    tmpStr = "I guess some people never change. Or, they quickly change and then
quickly change back.";
    this.SayTeam(tmpStr);
}
else if (!this == null && teamkiller.kills > 8 && RandInt(1,10) == 2)
{
    tmpStr = infname+"^2, you must have some terrible emotional problems.";
    this.SayTeam(tmpStr);
}
}
}
}
// if not in the teamkill table already, add inflictor to the table
if ( !this.found && this.IsAllied(Inflictor) && Inflictor != this.GetGameEntity() )
{
    i = tableCount(this.teamkilltable);
    //print("i =",i);
    this.teamkilltable[i] = table({entity="",name="",kills=0});
    this.teamkilltable[i].name = infname;
    this.teamkilltable[i].kills = 1;
    this.teamkilltable[i].entity = Inflictor;
    //this.teamkilltable[tableCount(this.teamkilltable)].kills = 1;
    //teamkiller.name = infname;
    //teamkiller.kills = teamkiller.kills + 1;
}
//print("Team kill table for",this.Name,":");
//foreach ( teamkiller in this.teamkilltable )
//{
    //print(teamkiller,teamkiller.name,teamkiller.kills,teamkiller.entity);
//}
}
}
dont_blame_them = null;

```

```

// Sniper/knife tk check.
// If this bot was killed by knife or sniper rifle, it can't be an accident, so let's flame ...
if( Inflictor != this.GetGameEntity() && this.IsAllied(Inflictor) && (MeansOfDeath ==
"MOD_K43_SCOPE" || MeansOfDeath == "MOD_FG42SCOPE" || MeansOfDeath == "MOD_KNIFE" ||
MeansOfDeath == "MOD_GARAND_SCOPE" ) )
{
    rand = RandInt(1,10);
    if (rand < 4)
    {
        this.SayTeam("Bastard!");
    }
    else if (rand < 7)
    {
        this.SayTeam("Asshole!");
    }
}

if (Inflictor != this.GetGameEntity() && !(GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO)))
{
    //this.SayVoice(VOICE.HEAL_ME);
    //this.Say("@*&!!");
}

// (Enemy) sniper check.
if( !this.IsAllied(Inflictor) && (MeansOfDeath == "MOD_K43_SCOPE" || MeansOfDeath ==
"MOD_FG42SCOPE" || MeansOfDeath == "MOD_GARAND_SCOPE" ) && this.GetNearestAlly(CAT.PLAYER,CLASS.
ANYPLAYER) == null )
{
    this.sniper_deaths = this.sniper_deaths + 1;
    rand = RandInt(1,10);
    if (rand < 4 && this.sniper_deaths > 5 && this.GetTeam == TEAM.AXIS)
    {
        this.SayTeam("Achtung, Scharfschuetze!");
        this.sniper_deaths = this.sniper_deaths - 1;
    }
    else if (rand < 4 && this.sniper_deaths > 5 && this.GetTeam == TEAM.ALLIES)
    {
        this.SayTeam("Sniper out there, beware!");
        this.sniper_deaths = this.sniper_deaths - 1;
    }
}

// "Great shot" only when certain weapons were used.
// Edit the nopraiseweapons table accordingly
foreach ( weapon in this.nopraiseweapons )
// don't say anything when killed by dynamite etc.
{
    if (weapon == MeansOfDeath)
    {
        return;
    }
}

if( this.IsAllied(Inflictor) == false && RandRange(1, 100) < 3 )
{

```

```

        this.SayVoice(VOICE.G_GREATSHOT);
    }
else if( this.IsAllied(Inflictor) == false && RandRange(1, 150) > 149 )
{
    this.SayVoice(VOICE.G_OOPS);
}
};

// On team chat event:
team_chatmsg_func = function( WhoSaidIt, Message )
{
    if( this==null || WhoSaidIt == null || WhoSaidIt == this.GetGameEntity || this.GetGameEntity
    == null )
    { return; }

    // Medics answering welcome to written thanks.
    if ( WhoSaidIt != this.GetGameEntity && this.IsAllied(WhoSaidIt) && this.Health > 0 && (this.
    GetClass() == CLASS.MEDIC || this.GetClass() == CLASS.FIELDOPS) )
    {
        if (this.answering)
        {
            return;
        }
        Posgratefulguy = GetEntPosition( WhoSaidIt );
        if( Posgratefulguy )
        {
            distance = this.DistanceTo(Posgratefulguy);
            if ( distance < 200 )
            {
                foreach (tablemessage in this.teamchatthanks)
                {
                    if (Message == tablemessage)
                    {
                        this.saywelcome = true;
                    }
                }
            }
            rndtmp = RandInt(1,10);
            if (this.saywelcome && rndtmp < 8)
            {
                this.answering = 1;
                sleep(0.5);
                this.TurnToPosition(Posgratefulguy);
                if (rndtmp < 3)
                {
                    this.SayVoice(VOICE.WELCOME);
                }
                else if (rndtmp < 5)
                {
                    this.SayTeam("np!");
                }
                else if (rndtmp < 8)
                {
                    this.SayTeam("You're welcome!");
                }
                sleep(2.0);
            }
        }
    }
}

```

```

        this.answering = null;
    }
    this.saywelcome = null;
}
}

if( Message == "snipe here" )
{
    if( this.CanSnipe() )
    {
        this.SayVoice(VOICE.AFFIRMATIVE);

        sourceEntPos = GetEntPosition(WhoSaidIt);
        sourceEntFacing = GetEntFacing(WhoSaidIt);

        if(sourceEntPos && sourceEntFacing)
        {
            this.Snipe(sourceEntPos, sourceEntFacing);

            status = block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED);
            if(status == EVENT.GOAL_FAILED)
            {
                this.Say("Can't do it!");
            }
        }
    }
    return;
}

if( Message == "skills" )
{
    this:saySkills();
    return;
}

// Restrict it to only when waypoint mode is on for now.
if( Wp.IsWaypointViewOn() )
{
    if(Message == "come")
    {
        this.SayVoice(VOICE.AFFIRMATIVE);

        sourceEntPos = GetEntPosition(WhoSaidIt);
        sourceEntFacing = GetEntFacing(WhoSaidIt);

        if(sourceEntPos && sourceEntFacing)
        {
            this.SetScriptControlled(true);
            this.GoTo(sourceEntPos, sourceEntFacing);
        }

        status = block(EVENT.GOAL_SUCCESS, EVENT.GOAL_FAILED);
        if(status == EVENT.GOAL_SUCCESS)
        {

```

```

        this.Say("Now What?");
    }
}
else if(Message == "roam")
{
    this.SayVoice(VOICE.AFFIRMATIVE);
    this.SetScriptControlled(false);
}
}
};

voice_func = function(sourceEnt, VoiceId)
{
    if(this==null || sourceEnt==null || VoiceId==null )
    {
        return;
    }
    if(VoiceId && sourceEnt)
    {
        //the gratefulguy thing (by Bladen)
        if(!(this==null) && VoiceId == VOICE.THANKS)
        {
            //get the source of the voice-thanks from sourceEnt
            gratefulguy = sourceEnt;

            //check against the teamkill table
            foreach ( teamkiller in this.teamkilltable )
            {
                if ( teamkiller != null )
                {
                    if ( !(this==null) && teamkiller.entity == gratefulguy )
                    {
                        if (teamkiller.kills > 8)
                        {
                            this.SayTeam("Pah, teamkiller!");
                            return;
                        }
                    }
                }
            }
        }

        if ( !(this==null) && gratefulguy != this.GetGameId() && this.IsAllied(gratefulguy) &&
this.Health > 0 )
        {
            Posgratefulguy = GetEntPosition( gratefulguy );
            //print(Posgratefulguy);
            if( !(this==null) && Posgratefulguy )
            {
                distance = this.DistanceTo(Posgratefulguy);
                if ( distance < 80 )
                {
                    //this bot must be either medic or fops
                    if ( !(this==null) && this.GetClass() == CLASS.MEDIC || this.GetClass() == CLASS.
FIELDOPS && !(GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO)) )
                    {

```

```

        sleep(1.0);
        this.TurnToPosition(Posgratefulguy);
        if (this.answering)
        {
            return;
        }
        this.answering = 1;
        if (!(this==null) && RandInt(0,12) > 11)
        {
            vname = GetEntityName(gratefulguy);
            this.SayTeam(vname, ", I think this is the beginning of a wonderful
friendship.");
        }
        else
        {
            this.SayVoice(VOICE.WELCOME);
        }
        sleep(2.0);
        if (!(this==null))
        {
            this.answering = null;
        }
        return;
    }
}
}
}

//Medics answering to "need medic"
if(!(this==null) && VoiceId == VOICE.NEED_MEDIC && this.GetNearestAlly(CAT.PLAYER,CLASS.
ANYPLAYER))
{
    rand = RandInt(1,10);
    if ( !(this==null) && this.GetClass() == CLASS.MEDIC && this.GetGameEntity() != sourceEnt
&& rand < 3 && this.Health > 0 && !(GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO)) && !
(GetEntFlags(this.GetGameEntity(), ENTFLAG.CARRYINGGOAL)) )
    {
        if (this.answering)
        {
            return;
        }
        else if (rand < 3)
        {
            this.answering = 1;
            sleep(1.5);
            this.SayVoice(VOICE.IMA_MEDIC);
            sleep(0.5);
            if (this.GetTeam() == TEAM.AXIS && rand < 2)
            {
                this.SayTeam("Halte aus, Kamerad!");
            }
            if (this.GetTeam() == TEAM.ALLIES && rand < 2)
            {
                this.SayTeam("Medic on the way!");
            }
        }
    }
}

```



```

        }
        sleep(2.0);
    }
    this.answering = null;
}

// Field ops answering to "need ammo"
if(!(this==null) && VoiceId == VOICE.NEED_AMMO)
{
    rand = RandInt(1,10);
    if ( !(this==null) && this.GetClass() == CLASS.FIELDOPS && this.GetGameEntity() !=
sourceEnt && this.Health > 0 && !(GetEntFlags(this.GetGameEntity(), ENTFLAG.LIMBO)) && !
(GetEntFlags(this.GetGameEntity(), ENTFLAG.CARRYINGGOAL)) )
    {
        pos = GetEntPosition( sourceEnt );
        if (this.answering)
        {
            return;
        }
        else if (rand < 5 && this.DistanceTo(pos) < 350)
        {
            this.answering = 1;
            sleep(1.5);
            this.SayVoice(VOICE.IMA_FIELDOPS);
            sleep(2.0);
        }
        this.answering = null;
    }
}

if( VoiceId == VOICE.ALL_CLEAR && this.GetTarget() == null && this.GetNearestEnemy(CAT.
PLAYER,CLASS.ANYPLAYER) == null )
{
    if (RandInt(1,10) < 2)
    {
        this.SayVoice(VOICE.AFFIRMATIVE);
    }
    else if (RandInt(1,10) > 8)
    {
        this.SayTeam("Affirmative: No enemy in sight!");
    }
}

//Avoid friendly hand grenades and airstrike markers if warned by "fire
//in the hole"
//Based on an idea by Bladen
if(VoiceId == VOICE.FIRE_IN_THE_HOLE && sourceEnt != this.GetGameEntity())
{
    //print("^6watchForProjectiles");
    timer = GetTime() + 6000;
    while(GetTime() < timer)
    {
        distancetohome = this.DistanceTo(this.HomePos);

```

```

AirStrikeMarker = this.GetNearest(CAT.PROJECTILE, CLASS.SMOKEMARKER);
if(AirStrikeMarker)
{
    //print("it's an airstrike marker");
    strikePos = GetEntPosition(AirStrikeMarker);
    if(strikePos)
    {
        distance = this.DistanceTo(strikePos);
    }

    if(distance < 600)
    {
        this.SetScriptControlled(true);
        endtime = GetTime() + 3000;
        dowhile( GetTime() < endtime )
        {
            this.MoveTowards(this.HomePos);
            this.PressButton(BTN.SPRINT);
            this.PressButton(BTN.JUMP);
            yield();
        }
        this.SetScriptControlled(false);
        continue;
    }
    else if(distance < 1500)
    {
        this.EnableMovement(false);
        this.HoldButton(BTN.PRONE,1);
        sleep(1.0);
        this.EnableMovement(true);
        continue;
    }
    else
    {
        sleep(1.0);
        continue;
    }
}

nadeEnt = this.GetNearest(CAT.PROJECTILE, CLASS.GRENADE);
if(nadeEnt==null)
{
    nadeEnt = this.GetNearest(CAT.PROJECTILE, CLASS.GPG40_GRENADE);
}
if(nadeEnt)
{
    //print("it's a grenade");
    nadePos = GetEntPosition(nadeEnt);
    if(nadePos)
    {
        distance = this.DistanceTo(nadePos);
    }

    if(distance < 600)
    {

```

```

        this.SetScriptControlled(true);
        endtime = GetTime() + 800;
        dowhile( GetTime() < endtime )
        {
            this.MoveTowards(this.HomePos);
            this.PressButton(BTN.SPRINT);
            yield();
        }
        this.SetScriptControlled(false);
        continue;
    }
    else if(distance < 1200)
    {
        this.HoldButton(BTN.PRONE, 5.0);
        sleep(5.0);
        continue;
    }
    else
    {
        sleep(1.0);
        continue;
    }
}
sleep(0.2);
}
}

// State own class in response to somebody else's
if(VoiceId == VOICE.IMA_SOLDIER && sourceEnt != this.GetGameEntity && RandInt(1,100)<10)
{
    if (this.answering == true)
    {
        return;
    }
    this.answering = true;
    if (this.GetClass == CLASS.FIELDOPS)
    {
        this.SayVoice(VOICE.IMA_FIELDOPS);
    }
    if (this.GetClass == CLASS.COVERTOPS)
    {
        this.SayVoice(VOICE.IMA_COVERTOPS);
    }
    if (this.GetClass == CLASS.MEDIC)
    {
        this.SayVoice(VOICE.IMA_MEDIC);
    }
    if (this.GetClass == CLASS.ENGINEER)
    {
        this.SayVoice(VOICE.IMA_ENGINEER);
    }
    sleep(3);
    this.answering = null;
}

```

```

// State own class in response to somebody else's at the start of game
if((VoiceId == VOICE.IMA_SOLDIER || VoiceId == VOICE.IMA_ENGINEER || VoiceId == VOICE.
IMA_FIELDOPS || VoiceId == VOICE.IMA_COVERTOPS) && sourceEnt != this.GetGameEntity && RandInt
(1,100)<10)
{
    //print("Class-Dings!");
    if (this.answering == true)
    {
        return;
    }
    currenttime = GetTime();
    if (currenttime > 20000) { return; }
    this.answering = true;
    if (this.GetClass == CLASS.FIELDOPS)
    {
        this.SayVoice(VOICE.IMA_FIELDOPS);
    }
    if (this.GetClass == CLASS.COVERTOPS)
    {
        this.SayVoice(VOICE.IMA_COVERTOPS);
    }
    if (this.GetClass == CLASS.MEDIC)
    {
        this.SayVoice(VOICE.IMA_MEDIC);
    }
    if (this.GetClass == CLASS.ENGINEER)
    {
        this.SayVoice(VOICE.IMA_ENGINEER);
    }
    sleep(3);
    this.answering = null;
}

//Some escort behavior (by Bladen)
if(VoiceId == VOICE.COVER_ME || VoiceId == VOICE.NEED_BACKUP || VoiceId == VOICE.FOLLOW_ME)
{
    if (!(this==null) && voice_escortbots < 10 && this.GetClass() != CLASS.ENGINEER && this.
GetGameEntity() != sourceEnt && GetEntFlags(this.GetGameEntity(), ENTFLAG.CARRYINGGOAL) ==
false)
    {
        if(sourceEnt)
        {
            //print("^6voice_escort: ", voice_escortbots);
            sourcePos = GetEntPosition(sourceEnt);
            if(sourcePos)
            {
                if (this.scriptedgoal || this.DistanceTo(sourcePos) > 1500)
                {
                    //print("Cover me: returned distance too long or busy");
                    sleep(0.5);
                    tmpRan = RandInt(1,10);
                    if (tmpRan < 2)
                    {
                        this.SayVoice(VOICE.COMMAND_DECLINED);
                    }
                }
            }
        }
    }
}

```

```

    else if (tmpRan > 9)
    {
        this.SayVoice(VOICE.NEGATIVE);
    }
    return;
}
sleep(1.0);
//print("^1voice_escort: ", voice_escortbots);
if (voice_escortbots > 2)
{
    //print("^6break");
    //sleep(2.0);
    return;
}
if (RandInt(1,10) < 3)
{
    this.SayVoice(VOICE.COMMAND_ACK);
}
else
{
    this.SayVoice(VOICE.AFFIRMATIVE);
}
this.scriptedgoal = coverme;
this.scriptedgoal = "voice coverme";

// store the # of escorting bots in a global variable
global voice_escortbots = voice_escortbots + 1;

timer = GetTime() + 45000;
while(!(this==null) && GetTime() < timer && GetEntHealthAndArmor(sourceEnt).Health >
0)
{
    if(!(this==null) && !(this.HasTarget()) )
    {
        endtime = GetTime() + 300;
        dowhile(!(this==null) && GetTime() < endtime)
        {
            this.TurnToFacing(GetEntFacing(sourceEnt));
            //sleep(1);
            yield();
        }
    }
    sourcePos = GetEntPosition( sourceEnt );
    if(sourcePos)
    {
        //print("sourcePos:", sourcePos);
        //print("distance to sourceEnt is:", distance);
        if (!(this==null) && this.DistanceTo(sourcePos) > 800)
        {
            this.scriptedescort = escortteammate;
            //random offset (x,y) x=200 to 600 y=200 to 600
            randomizerX = RandInt(-200,200);
            randomizerY = RandInt(-200,200);
            randomPos = sourcePos + Vector3(randomizerX,randomizerY,0);
            this.EnableMovement(true);

```

```

        this.SetScriptControlled(true);
        this.GoTo(randomPos, GetEntFacing(sourceEnt) );
        sleep(8.0);
        this.SetScriptControlled(false);
        this.scriptedescort = null;
        sleep(2.0);
        continue;
    }
    sleep(1.0);
}
else
{
    //print("no source pos.!!!!");
    // store the # of escorting bots in a global variable
    global voice_escortbots = voice_escortbots - 1;
    yield();
    return;
}
}
//print("done");
if (!(this==null))
{
    this.SetScriptControlled(false);
    this.EnableMovement(true);
    this.scriptedgoal = null;
    this.scriptedwp = null;
    global voice_escortbots = voice_escortbots - 1;
    //escortedEntity.escortbots = escortedEntity.escortbots - 1;
    //print("^2Escort: ",voice_escortbots);
    //print("^2Escort: ",escortedEntity.escortbots);
}
}
}
}
else
{
    this.SetScriptControlled(false);
    return;
}
this.SetScriptControlledWeapons(false);
this.SetScriptControlled(false);
return;
}
}
return;
};

```

```

global_voice_func = function (WhoSaidIt,VoiceId)
{
    if (VoiceId == VOICE.G_GREATSHOT && WhoSaidIt == this.lastkill)
    {
        sleep(1.0);
        now = GetTime();
        if (now < this.lastkilltime + 5000 && RandInt(1,10) < 8)
        {

```

```

        this.SayVoice(VOICE.G_THANKS);
    }
}

// Next 2 IFs: General global talk behavior (Bye, hello, etc.)
if(VoiceId == VOICE.G_BYE && sourceEnt != this.GetGameEntity && RandInt(1,100)<3)
{
    this.SayVoice(VOICE.G_BYE);
}

if(VoiceId == VOICE.G_HI && sourceEnt != this.GetGameEntity && RandInt(1,100)<3)
{
    this.SayVoice(VOICE.G_HI);
}

};

killed_someone = function( _source, _params )
// "the bot killed someone" event
{
    if(this==null || _source==null || _params==null )
    {
        return;
    }

    //keep the last victim for a while
    this.lastkill = _source;
    this.lastkilltime = GetTime();
    //print(this.lastkill);
    //print(this.lastkilltime);

    //print("killed_someone: _params:", _params);
    foreach ( weapon in this.notauntweapons )
    {
        if (weapon == _params)
        {
            dont_taunt_them = true;
        }
    }
    if (this.IsAllied( _source ) == false && dont_taunt_them == false)
    // if not a team kill, say one of the this.oneliners or some other
    // bullshit
    {
        rand = RandInt(0, tableCount(this.oneliners)+180); //only in some cases
        if (rand == tableCount(this.oneliners)+1)
        {
            vteam = GetEntTeam(_source);
            if (vteam == TEAM.ALLIES)
            {
                this.Say("Is that a .45 in your pocket, or are you just pleased to see me?");
            }
            if (vteam == TEAM.AXIS)
            {
                this.Say("Is that a Luger in your pocket, or are you just pleased to see me?");
            }
        }
    }
}

```

```

}
if (rand == tableCount(this.oneliners)+2)
{
    vname = GetEntityName(_source);
    if (vname != null && vname != "tank" && vname != "truck")
    {
        this.Say("^2Come on, ",vname,"^2, stop ^wtrying^2 to hit me and ^whit^2 me.");
    }
}
else if (rand < tableCount(this.oneliners))
{
    this.Say(this.oneliners[rand]);
}
}

else if (this.IsAllied(_source)) // say 'sorry' when guilty of a team kill
{
    vname = GetEntityName(_source);
    if ( vname == null || vname == "tank" || vname == "truck" )
    {
        dont_say_sorry = true;
    }
    foreach ( weapon in this.nosorryweapons )
    <font color="green" st

```



# AMMO

From Omni-bot Wiki

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=AMMO>"

This page has been accessed 220 times. This page was last modified 16:30, 20 May 2007.

# BB

From Omni-bot Wiki

## Common Blackboard Keys

BB.DELAY\_GOAL

BB.IS\_TAKEN

BB.RUN\_AWAY

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=BB>"

This page has been accessed 166 times. This page was last modified 20:12, 26 May 2007.

# BONE

From Omni-bot Wiki

## Common Bone Ids

BONE.HEAD

BONE.LEFTARM

BONE.LEFTFOOT

BONE.LEFTHAND

BONE.LEFTLEG

BONE.PELVIS

BONE.RIGHTARM

BONE.RIGHTFOOT

BONE.RIGHTHAND

BONE.RIGHTLEG

BONE.TORSO

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=BONE>"

This page has been accessed 159 times. This page was last modified 16:33, 15 September 2007.

# BTN

From Omni-bot Wiki

## Common Buttons

BTN.AIM  
BTN.ATTACK1  
BTN.ATTACK2  
BTN.BACKWARD  
BTN.CROUCH  
BTN.DROP  
BTN.FORWARD  
BTN.JUMP  
BTN.LEAN\_L  
BTN.LEAN\_R  
BTN.PRONE  
BTN.RELOAD  
BTN.SPRINT  
BTN.STRAFE\_L  
BTN.STRAFE\_R  
BTN.USE  
BTN.WALK

## Fortress Forever Buttons

BTN.GREN1  
BTN.GREN2  
BTN.GREN\_THROW  
BTN.DROP\_AMMO  
BTN.DROP\_ITEM

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=BTN>"

This page has been accessed 182 times. This page was last modified 20:14, 26 May 2007.

# BUY

From Omni-bot Wiki

## Quake 4 Buy Menu

BUY.AMMO\_REFILL  
BUY.AMMO\_REGEN  
BUY.ARMOR\_RED  
BUY.ARMOR\_YELLOW  
BUY.DAMAGE\_BOOST  
BUY.GRENADELAUNCHER  
BUY.HEALTH\_REGEN  
BUY.HYPERBLASTER  
BUY.LIGHTNINGGUN  
BUY.MACHINEGUN  
BUY.NAILGUN  
BUY.NAPALMGUN  
BUY.RAILGUN  
BUY.ROCKETLAUNCHER  
BUY.SHOTGUN

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=BUY>"

This page has been accessed 152 times. This page was last modified 20:15, 26 May 2007.

# CAT

From Omni-bot Wiki

## Common Categories

CAT.AUTODEFENSE

CAT.AVOID

CAT.MISC

CAT.MOUNTED\_WPN

CAT.MOVER

CAT.PICKUP

CAT.PLAYER

CAT.PROJECTILE

CAT.SHOOTABLE

CAT.TRIGGER

CAT.STATIC

CAT.VEHICLE

## Fortress Forever Categories

CAT.BUILDABLE

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=CAT>"

This page has been accessed 160 times. This page was last modified 20:16, 26 May 2007.

# CLASS

From Omni-bot Wiki

## Contents

- 1 Common Classes
- 2 Doom 3 Classes
- 3 Enemy Territory Classes
- 4 Quake 4 Classes
- 5 Fortress Forever Classes
- 6 Return to Castle Wolfenstein Classes

## Common Classes

CLASS.AMMO  
CLASS.ARMOR  
CLASS.BUTTON  
CLASS.EXPLODING\_BARREL  
CLASS.HEALTH  
CLASS.LADDER  
CLASS.LIFT  
CLASS.MOVER  
CLASS.PLAYERSTART  
CLASS.TELEPORTER

## Doom 3 Classes

CLASS.ANYPLAYER  
CLASS.PLAYER

## Enemy Territory Classes

CLASS.AIRSTRIKE  
CLASS.AMMOCABINET  
CLASS.ANYPLAYER  
CLASS.ARTY  
CLASS.BREAKABLE  
CLASS.CORPSE  
CLASS.COVERTOPS  
CLASS.DYNAMITE

CLASS.ENGINEER  
CLASS.FLAME  
CLASS.FIELDOPS  
CLASS.GPG40\_GRENADE  
CLASS.GRENADE  
CLASS.HEALTHCABINET  
CLASS.INJUREDPLAYER  
CLASS.MEDIC  
CLASS.MG42MOUNT  
CLASS.LANDMINE  
CLASS.MORTAR  
CLASS.M7\_GRENADE  
CLASS.ROCKET  
CLASS.SATCHEL  
CLASS.SMOKEBOMB  
CLASS.SMOKEMARKER  
CLASS.SOLDIER  
CLASS.TREASURE  
CLASS.VEHICLE  
CLASS.VEHICLE\_NODAMAGE  
CLASS.VEHICLE\_HVY

## Quake 4 Classes

CLASS.AMMO\_REGEN  
CLASS.ANYPLAYER  
CLASS.DEADZONE  
CLASS.DOUBLER  
CLASS.GRENADE  
CLASS.GUARD  
CLASS.HASTE  
CLASS.INVISIBILITY  
CLASS.MARINE\_FLAG  
CLASS.ONE\_FLAG  
CLASS.PLAYER  
CLASS.QUAD\_DAMAGE  
CLASS.REGENERATION  
CLASS.Scout  
CLASS.STROGG\_FLAG

## Fortress Forever Classes

CLASS.ANYPLAYER



CLASS.BACKPACK  
CLASS.BACKPACK\_AMMO  
CLASS.BACKPACK\_ARMOR  
CLASS.BACKPACK\_GRENADES  
CLASS.BACKPACK\_HEALTH  
CLASS.CALTROP  
CLASS.CIVILIAN  
CLASS.CONC\_GRENADE  
CLASS.DEMOMAN  
CLASS.DETPACK  
CLASS.DISPENSER  
CLASS.EMP\_GRENADE  
CLASS.ENGINEER  
CLASS.GAS\_GRENADE  
CLASS.GLGREN  
CLASS.GRENADE  
CLASS.HWGUY  
CLASS.MEDIC  
CLASS.MIRV\_GRENADE  
CLASS.MIRVLET\_GRENADE  
CLASS.NAIL\_GRENADE  
CLASS.NAPALM\_GRENADE  
CLASS.PIPEBOMB  
CLASS.PYRO  
CLASS.ROCKET  
CLASS.SCOOT  
CLASS.SENTRY  
CLASS.SNIPER  
CLASS.SOLDIER  
CLASS.SPY  
CLASS.TURRET

## Return to Castle Wolfenstein Classes

CLASS.AIRSTRIKE  
CLASS.ANYPLAYER  
CLASS.ARTY  
CLASS.BREAKABLE  
CLASS.CORPSE  
CLASS.DYNAMITE  
CLASS.ENGINEER  
CLASS.FLAME  
CLASS.GRENADE  
CLASS.INJUREDPLAYER

CLASS.LIEUTENANT  
CLASS.MEDIC  
CLASS.MG42MOUNT  
CLASS.ROCKET  
CLASS.SMOKEBOMB  
CLASS.SMOKEMARKER  
CLASS.SOLDIER  
CLASS.TREASURE  
CLASS.VEHICLE  
CLASS.VEHICLE\_NODAMAGE  
CLASS.VEHICLE\_HVY

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=CLASS>"

This page has been accessed 212 times. This page was last modified 04:07, 28 April 2008.

# CONTENT

From Omni-bot Wiki

## Common Content Flags

CONT.FOG

CONT.LAVA

CONT.MOVER

CONT.SLIME

CONT.SOLID

CONT.TELEPORTER

CONT.TRIGGER

CONT.WATER

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=CONTENT>"

This page has been accessed 142 times. This page was last modified 16:32, 15 September 2007.

# DEBUG

From Omni-bot Wiki

## Common Debug Flags

DEBUG.AIM

DEBUG.BRAIN

DEBUG.EVENTS

DEBUG.GOALS

DEBUG.LOG

DEBUG.MOVEMENT

DEBUG.PLANNER

DEBUG.SENSORY

DEBUG.SCRIPT

DEBUG.WEAPON

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=DEBUG>"

This page has been accessed 172 times. This page was last modified 16:36, 15 September 2007.

# ENTFLAG

From Omni-bot Wiki

## Contents

- 1 Common Entity Flags
- 2 Doom 3 Entity Flags
- 3 Enemy Territory Entity Flags
- 4 Fortress Forever Entity Flags
- 5 Quake 4 Entity Flags

## Common Entity Flags

ENTFLAG.TEAM1  
ENTFLAG.TEAM2  
ENTFLAG.TEAM3  
ENTFLAG.TEAM4  
ENTFLAG.DISABLED  
ENTFLAG.PRONE  
ENTFLAG.CROUCHED  
ENTFLAG.CARRYABLE  
ENTFLAG.DEAD  
ENTFLAG.INWATER  
ENTFLAG.UNDERWATER  
ENTFLAG.ZOOMING  
ENTFLAG.ON\_LADDER  
ENTFLAG.ON\_GROUND  
ENTFLAG.RELOADING  
ENTFLAG.HUMANCONTROLLED

## Doom 3 Entity Flags

ENTFLAG.IN\_VEHICLE  
ENTFLAG.FLASHLIGHT\_ON

## Enemy Territory Entity Flags

ENTFLAG.DISGUISED  
ENTFLAG.MOUNTED  
ENTFLAG.MNT\_MG42

ENTFLAG.MNT\_TANK  
ENTFLAG.MNT\_AAGUN  
ENTFLAG.CARRYINGGOAL  
ENTFLAG.LIMBO  
ENTFLAG.MOUNTABLE  
ENTFLAG.POISONED

## Fortress Forever Entity Flags

ENTFLAG.NEED\_HEALTH  
ENTFLAG.NEED\_ARMOR  
ENTFLAG.BURNING  
ENTFLAG.TRANQUED  
ENTFLAG.SNIPE\_AIMING  
ENTFLAG.AC\_FIRING  
ENTFLAG.LEGSHOT  
ENTFLAG.CALTROP  
ENTFLAG.RADIOTAGGED  
ENTFLAG.CAN\_SABOTAGE  
ENTFLAG.SABOTAGED  
ENTFLAG.SABOTAGING  
ENTFLAG.BUILDING\_SG  
ENTFLAG.BUILDING\_DISP  
ENTFLAG.BUILDING\_DETP

## Quake 4 Entity Flags

ENTFLAG.IN\_VEHICLE  
ENTFLAG.FLASHLIGHT\_ON  
ENTFLAG.IN\_BUY\_ZONE

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=ENTFLAG>"

This page has been accessed 184 times. This page was last modified 16:37, 15 September 2007.

# EVENT

From Omni-bot Wiki

## Contents

- 1 Event Usage
- 2 Common Events
  - 2.1 ADDWEAPON
  - 2.2 CHANGECLASS
  - 2.3 CHANGETEAM
  - 2.4 DEATH
  - 2.5 HEALED
  - 2.6 FEEL\_PAIN
  - 2.7 GLOBAL\_CHAT\_MSG
  - 2.8 GLOBAL\_VOICE
  - 2.9 HEAR\_SOUND
  - 2.10 KILLED SOMEONE
  - 2.11 PLAYER\_USE
  - 2.12 PRIV\_CHAT\_MSG
  - 2.13 PRIVATE\_VOICE
  - 2.14 REMOVEWEAPON
  - 2.15 RESETWEAPONS
  - 2.16 REVIVED
  - 2.17 TEAM\_CHAT\_MSG
  - 2.18 TEAM\_VOICE
  - 2.19 SPAWNED
  - 2.20 WEAPON\_CHANGE
  - 2.21 WEAPON\_FIRE
- 3 Enemy Territory Events
  - 3.1 MORTAR\_IMPACT
  - 3.2 POSTTRIGGERED\_MINE
  - 3.3 PRETRIGGERED\_MINE
- 4 Fortress Forever Events
  - 4.1 DETPACK\_ALREADYBUILT
  - 4.2 DETPACK\_BUILDING
  - 4.3 DETPACK\_BUILT
  - 4.4 DETPACK\_CANTBUILD
  - 4.5 DETPACK\_DETONATED
  - 4.6 DETPACK\_NOAMMO
  - 4.7 DISGUISED
  - 4.8 DISGUISING
  - 4.9 DISPENSER\_ALREADYBUILT

- 4.10 DISPENSER\_BUILDING
- 4.11 DISPENSER\_BUILT
- 4.12 DISPENSER\_CANTBUILD
- 4.13 DISPENSER\_DAMAGED
- 4.14 DISPENSER\_DESTROYED
- 4.15 DISPENSER\_DETONATED
- 4.16 DISPENSER\_DISMANTLED
- 4.17 DISPENSER\_ENEMYUSED
- 4.18 DISPENSER\_NOAMMO
- 4.19 INVALID\_DISGUISE\_CLASS
- 4.20 INVALID\_DISGUISE\_TEAM
- 4.21 RADAR\_DETECTENEMY
- 4.22 RADIOTAG\_UPDATE
- 4.23 SENTRY\_ALREADYBUILT
- 4.24 SENTRY\_BUILDING
- 4.25 SENTRY\_BUILT
- 4.26 SENTRY\_CANTBUILD
- 4.27 SENTRY\_DAMAGED
- 4.28 SENTRY\_DESTROYED
- 4.29 SENTRY\_DETONATED
- 4.30 SENTRY\_DISMANTLED
- 4.31 SENTRY\_NOAMMO
- 4.32 SENTRY\_SPOTENEMY
- 4.33 SENTRY\_UPGRADED

## Event Usage

Events can be assigned callbacks in scripts, normally in the bot scripts.

For Example:

```
this.Events[EVENT.TEAM_CHAT_MSG] = function(WhoSaidIt, Message)
{
    n = GetEntName(WhoSaidIt);
    this.Say(n, "said: ", Message);
    print(n, "said: ", Message);
};
```

## Common Events



## **ADDWEAPON**

### **Parameters:**

- weaponId

## **CHANGECLASS**

### **Parameters:**

- newclass

## **CHANGETEAM**

### **Parameters:**

- newteam

## **DEATH**

### **Parameters:**

- inflictor
- meansofdeath

## **HEALED**

### **Parameters:**

- whohealedme

## **FEEL\_PAIN**

### **Parameters:**

- inflictor
- previoushealth
- currenthealth

## **GLOBAL\_CHAT\_MSG**

## Parameters:

- whosaidit
- whattheysaid

## GLOBAL\_VOICE

## Parameters:

- source
- macro id

## HEAR\_SOUND

## Parameters:

- source
- origin
- soundId
- soundName

## KILLED SOMEONE

## Parameters:

- victim
- meansofdeath

## PLAYER\_USE

## Parameters:

- whotouchedme

## PRIV\_CHAT\_MSG

## Parameters:

- whosaidit
- whattheysaid

## PRIVATE\_VOICE

### Parameters:

- macro id

## REMOVEWEAPON

### Parameters:

- weaponId

## RESETWEAPONS

### Parameters: none

## REVIVED

### Parameters:

- whorevivedme

## TEAM\_CHAT\_MSG

### Parameters:

- whosaidit
- whattheysaid

## TEAM\_VOICE

### Parameters:

- source
- macro id

## SPAWNED

### Parameters: none

## WEAPON\_CHANGE

### **Parameters:**

- weaponId

## **WEAPON\_FIRE**

### **Parameters:**

- weaponId
- projectile entity

## **Enemy Territory Events**

## **MORTAR\_IMPACT**

### **Parameters:**

- position

## **POSTTRIGGERED\_MINE**

### **Parameters:**

- mine\_entity

## **PRETRIGGERED\_MINE**

### **Parameters:**

- mine\_entity

## **Fortress Forever Events**

## **DETPACK\_ALREADYBUILT**

**Parameters:** none

## **DETPACK\_BUILDING**

## Parameters:

- detpack entity

## DETPACK\_BUILT

## Parameters:

- detpack entity

## DETPACK\_CANTBUILD

Parameters: none

## DETPACK\_DETONATED

Parameters: none

## DETPACK\_NOAMMO

Parameters: none

## DISGUISED

## Parameters:

- asTeam
- asClass

## DISGUIISING

## Parameters:

- asTeam
- asClass

## DISPENSER\_ALREADYBUILT

Parameters: none

## DISPENSER\_BUILDING

### **Parameters:**

- dispenser entity

## **DISPENSER\_BUILT**

### **Parameters:**

- dispenser entity

## **DISPENSER\_CANTBUILD**

**Parameters:** none

## **DISPENSER\_DAMAGED**

### **Parameters:**

- inflictor

## **DISPENSER\_DESTROYED**

**Parameters:** none

## **DISPENSER\_DETONATED**

**Parameters:** none

## **DISPENSER\_DISMANTLED**

**Parameters:** none

## **DISPENSER\_ENEMYUSED**

### **Parameters:**

- user entity

## **DISPENSER\_NOAMMO**

**Parameters:** none

## **INVALID\_DISGUISE\_CLASS**

**Parameters:**

- class

## **INVALID\_DISGUISE\_TEAM**

**Parameters:**

- team

## **RADAR\_DETECTENEMY**

**Parameters:**

- detected

## **RADIOTAG\_UPDATE**

**Parameters:**

- detected

## **SENTRY\_ALREADYBUILT**

**Parameters:** none

## **SENTRY\_BUILDING**

**Parameters:**

- sentry entity

## **SENTRY\_BUILT**

**Parameters:**

- sentry entity

## **SENTRY\_CANTBUILD**

**Parameters:** none

## **SENTRY\_DAMAGED**

**Parameters:**

- inflictor

## **SENTRY\_DESTROYED**

**Parameters:** none

## **SENTRY\_DETONATED**

**Parameters:** none

## **SENTRY\_DISMANTLED**

**Parameters:** none

## **SENTRY\_NOAMMO**

**Parameters:** none

## **SENTRY\_SPOTENEMY**

**Parameters:**

- enemy entity seen

## **SENTRY\_UPGRADED**

**Parameters:**

- level



# ITEM

From Omni-bot Wiki

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=ITEM>"

This page has been accessed 143 times. This page was last modified 16:36, 20 May 2007.

# POWERUP

From Omni-bot Wiki

## Contents

- 1 Common Powerup Flags
- 2 Doom 3 Powerup Flags
- 3 Fortress Forever Powerup Flags
- 4 Quake 4 Powerup Flags

## Common Powerup Flags

POWERUP.INVINCIBLE

## Doom 3 Powerup Flags

POWERUP.ADRENALINE

POWERUP.BERSERK

POWERUP.INVISIBILITY

POWERUP.MEGAHEALTH

## Fortress Forever Powerup Flags

POWERUP.DISGUISE\_BLUE

POWERUP.DISGUISE\_CIVILIAN

POWERUP.DISGUISE\_DEMOMAN

POWERUP.DISGUISE\_ENGINEER

POWERUP.DISGUISE\_GREEN

POWERUP.DISGUISE\_HWGUY

POWERUP.DISGUISE\_MEDIC

POWERUP.DISGUISE\_PYRO

POWERUP.DISGUISE\_RED

POWERUP.DISGUISE\_SCOUT

POWERUP.DISGUISE\_SNIPER

POWERUP.DISGUISE\_SOLDIER

POWERUP.DISGUISE\_SPY

POWERUP.DISGUISE\_YELLOW

## Quake 4 Powerup Flags

POWERUP.AMMOREGEN  
POWERUP.DOUBLER  
POWERUP.GUARD  
POWERUP.HASTE  
POWERUP.INVISIBILITY  
POWERUP.MARINEFLAG  
POWERUP.ONEFLAG  
POWERUP.QUADDAMAGE  
POWERUP.REGENERATION  
POWERUP.Scout  
POWERUP.STROGGFLAG  
POWERUP.TEAM\_AMMOREGEN  
POWERUP.TEAM\_DAMAGEMOD  
POWERUP.TEAM\_HEALTHREGEN

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=POWERUP>"

This page has been accessed 159 times. This page was last modified 18:18, 15 September 2007.

# PROFILE

From Omni-bot Wiki

## Profile Types

PROFILE.NONE

PROFILE.CLASS

PROFILE.CUSTOM

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=PROFILE>"

This page has been accessed 171 times. This page was last modified 18:17, 15 September 2007.

# SKILL

From Omni-bot Wiki

## Enemy Territory Skills

BATTLE\_SENSE

FIRST\_AID

COVERTOPS

ENGINEERING

HEAVY\_WEAPONS

LIGHT\_WEAPONS

SIGNALS

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=SKILL>"

This page has been accessed 197 times. This page was last modified 14:40, 14 February 2008.

# TEAM

From Omni-bot Wiki

## Contents

- 1 Enemy Territory Teams
- 2 Doom 3 Teams
- 3 Fortress Forever Teams
- 4 Quake 4 Teams

## Enemy Territory Teams

TEAM.ALLIES

TEAM.AXIS

## Doom 3 Teams

TEAM.BLUE

TEAM.RED

## Fortress Forever Teams

TEAM.BLUE

TEAM.GREEN

TEAM.RED

TEAM.YELLOW

## Quake 4 Teams

TEAM.MARINE

TEAM.STROGG

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=TEAM>"

This page has been accessed 182 times. This page was last modified 18:16, 15 September 2007.

# TRACE

From Omni-bot Wiki

## Common Trace Flags

TRACE.ALL

TRACE.OPAQUE

TRACE.PLAYER

TRACE.PLAYERCLIP

TRACE.SHOT

TRACE.SOLID

TRACE.SMOKEBOMB

TRACE.WATER

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=TRACE>"

This page has been accessed 170 times. This page was last modified 18:15, 15 September 2007.

# VOICE

From Omni-bot Wiki

## Enemy Territory Voice Macros

### Team

VOICE.AFFIRMATIVE

VOICE.ALL\_CLEAR

VOICE.CLEAR\_MINES

VOICE.CLEAR\_PATH

VOICE.COMMAND\_ACK

VOICE.COMMAND\_DECLINED

VOICE.COMMAND\_COMPLETED

VOICE.CONST\_COMMENCING

VOICE.COVER\_ME

VOICE.DEFEND\_OBJECTIVE

VOICE.DESTROY\_CONST

VOICE.DESTROY\_PRIMARY

VOICE.DESTROY\_SECONDARY

VOICE.DESTROY\_VEHICLE

VOICE.DISARM\_DYNAMITE

VOICE.ENEMY\_DISGUISED

VOICE.ENEMY\_WEAK

VOICE.ESCORT\_VEHICLE

VOICE.FIRE\_IN\_THE\_HOLE

VOICE.FOLLOW\_ME

VOICE.HOLD\_FIRE

VOICE.IMA\_COVERTOPS

VOICE.IMA\_ENGINEER

VOICE.IMA\_FIELDDOPS

VOICE.IMA\_MEDIC

VOICE.IMA\_SOLDIER

VOICE.INCOMING

VOICE.LETS\_GO

VOICE.MINES\_CLEARED

VOICE.MOVE

VOICE.NEED\_AMMO

VOICE.NEED\_BACKUP

VOICE.NEED\_ENGINEER

VOICE.NEED\_MEDIC

VOICE.NEED\_OPS



VOICE.NEGATIVE  
VOICE.ON\_DEFENSE  
VOICE.ON\_OFFENSE  
VOICE.OOPS  
VOICE.PATH\_CLEARED  
VOICE.REINFORCE\_OFF  
VOICE.REINFORCE\_DEF  
VOICE.REPAIR\_VEHICLE  
VOICE.SORRY  
VOICE.TAKING\_FIRE  
VOICE.THANKS  
VOICE.WELCOME  
VOICE.WHERE\_TO

## Global

VOICE.G\_AFFIRMATIVE  
VOICE.G\_BYE  
VOICE.G\_CHEER  
VOICE.G\_ENEMY\_WEAK  
VOICE.G\_GOODGAME  
VOICE.G\_GREATSHOT  
VOICE.G\_HI  
VOICE.G\_HOLD\_FIRE  
VOICE.G\_NEGATIVE  
VOICE.G\_OOPS  
VOICE.G\_SORRY  
VOICE.G\_THANKS  
VOICE.G\_WELCOME

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=VOICE>"

This page has been accessed 245 times. This page was last modified 18:14, 15 September 2007.

# WEAPON

From Omni-bot Wiki

## Contents

- 1 Doom 3 Weapons
- 2 Enemy Territory Weapons
- 3 Fortress Forever Weapons
- 4 Quake 4 Weapons

## Doom 3 Weapons

WEAPON.BFG  
WEAPON.CHAINGUN  
WEAPON.CHAINSAW  
WEAPON.FISTS  
WEAPON.FLASHLIGHT  
WEAPON.GRABBER  
WEAPON.HANDGRENADE  
WEAPON.MACHINEGUN  
WEAPON.PLASMAGUN  
WEAPON.PISTOL  
WEAPON.ROCKETLAUNCHER  
WEAPON.SHOTGUN  
WEAPON.SHOTGUN\_DBL  
WEAPON.SOULCUBE

## Enemy Territory Weapons

WEAPON.ADRENALINE  
WEAPON.ALLY\_GRENADE  
WEAPON.AMMO\_PACK  
WEAPON.AXIS\_GRENADE  
WEAPON.BINOCULARS  
WEAPON.CARBINE  
WEAPON.COLT  
WEAPON.COLT\_AKIMBO  
WEAPON.COLT\_AKIMBO\_SILENCED  
WEAPON.COLT\_SILENCED  
WEAPON.DYNAMITE

WEAPON.FG42  
WEAPON.FG42\_SCOPE  
WEAPON.FLAMETHROWER  
WEAPON.GARAND  
WEAPON.GARAND\_SCOPE  
WEAPON.GPG40  
WEAPON.KAR98  
WEAPON.K43  
WEAPON.K43\_SCOPE  
WEAPON.KNIFE  
WEAPON.LANDMINE  
WEAPON.LUGER  
WEAPON.LUGER\_AKIMBO  
WEAPON.LUGER\_AKIMBO\_SILENCED  
WEAPON.LUGER\_SILENCED  
WEAPON.M7  
WEAPON.MEDKIT  
WEAPON.MORTAR  
WEAPON.MORTAR\_SET  
WEAPON.MOBILE\_MG42  
WEAPON.MOBILE\_MG42\_SET  
WEAPON.MOUNTABLE\_MG42  
WEAPON.MP40  
WEAPON.PANZERFAUST  
WEAPON.PLIER  
WEAPON.SATCHEL  
WEAPON.SATCHEL\_DET  
WEAPON.SMOKE\_GRENADE  
WEAPON.SMOKE\_MARKER  
WEAPON.STEN  
WEAPON.SYRINGE  
WEAPON.THOMPSON

## Fortress Forever Weapons

WEAPON.ASSAULTCANNON  
WEAPON.AUTORIFLE  
WEAPON.CROWBAR  
WEAPON.FLAMETHROWER  
WEAPON.GRENADE  
WEAPON.GRENADELAUNCHER  
WEAPON.KNIFE  
WEAPON.MEDKIT  
WEAPON.NAILGUN

WEAPON.NAPALMCANNON  
WEAPON.PIPELAUNCHER  
WEAPON.SHOTGUN  
WEAPON.SPANNER  
WEAPON.SUPERSHOTGUN  
WEAPON.RAILGUN  
WEAPON.RPG  
WEAPON.SNIPERRIFLE  
WEAPON.SUPERNAILGUN  
WEAPON.TRANQGUN  
WEAPON.UMBRELLA

## Quake 4 Weapons

WEAPON.BLASTER  
WEAPON.DARKMATTERGUN  
WEAPON.GAUNTLET  
WEAPON.GRENADELAUNCHER  
WEAPON.HYPERBLASTER  
WEAPON.LIGHTNINGGUN  
WEAPON.MACHINEGUN  
WEAPON.NAILGUN  
WEAPON.NAPALMGUN  
WEAPON.RAILGUN  
WEAPON.RPG  
WEAPON.SHOTGUN

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=WEAPON>"

This page has been accessed 214 times. This page was last modified 18:15, 15 September 2007.

# Math Library

From Omni-bot Wiki

## Basic Math Functions

---

- $\text{Cos}(x)$  - Cosine.
- $\text{ACos}(x)$  - Arc Cosine.
- $\text{Sin}(x)$  - Sin.
- $\text{ASin}(x)$  - Arc Sin.
- $\text{Tan}(x)$  - Tangent.
- $\text{ATan}(x)$  - Arc Tangent.
- $\text{Ceil}(x)$  - Smallest integer  $\geq x$ .
- $\text{Floor}(x)$  - Largest integer  $\leq x$ .
- $\text{Round}(x)$  - Rounds the number to the nearest integer whole number.
- $\text{Abs}(x)$  - Absolute value of  $x$ .
- $\text{Clamp}(x, \text{min}, \text{max})$  - Clamps  $x$  to a minimum and maximum value.
- $\text{DegToRad}(x)$  - Converts degrees to radians.
- $\text{RadToDeg}(x)$  - Converts radians to degrees.
- $\text{Min}(x, y)$  - Returns the minimum value of the 2 parameters.
- $\text{Max}(x, y)$  - Returns the maximum value of the 2 parameters.
- $\text{RandInt}(x, y)$  - Returns a random integer between  $x$  and  $y$ .
- $\text{RandFloat}(x, y)$  - Returns a random float between  $x$  and  $y$ .
- $\text{Sign}(x)$  - Returns a signed scalar of  $x$ . Negative numbers returns -1, positive numbers return 1.
- $\text{Sqrt}(x)$  - Square root of  $x$ .
- $\text{SymmetricRandom}()$  - Returns random number between -1.0 and 1.0
- $\text{UnitRandom}()$  - Returns random number between 0.0 and 1.0
- $\text{ToFloat}(x)$  - Converts an integer or string to a float if possible.
- $\text{ToInt}(x)$  - Converts a float or string to an integer if possible.
- $\text{ToVector}(x)$  - Converts a string to a Vector3 if possible.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Math\\_Library](http://omni-bot.com/wiki/index.php?title=Math_Library)"

This page has been accessed 163 times. This page was last modified 03:50, 28 May 2007.

# System Library

From Omni-bot Wiki

## Contents

- 1 Global FileSystem Functions
  - 1.1 FileDelete
  - 1.2 FileEnumerate
  - 1.3 FileExists
  - 1.4 Newline
- 2 File
  - 2.1 Close
  - 2.2 EndOfFile
  - 2.3 FileSize
  - 2.4 Flush
  - 2.5 IsOpen
  - 2.6 Open
  - 2.7 ReadFloat
  - 2.8 ReadInt
  - 2.9 ReadLine
  - 2.10 ReadString
  - 2.11 Seek
  - 2.12 Tell
  - 2.13 Write

## Global FileSystem Functions

---

### FileDelete

Deletes a file by name. This function is restricted to files under the user folder.

**Parameters:** (filename)

**Returns:** true if success, false if failed

**Example:**

```
FileDelete("myfile.txt");
```

---

## FileEnumerate

Enumerates over all files in a directory. This function is restricted to files under the user folder. This function will call the provided script function with all files enumerated.

**Parameters:** (folder name, script function)

**Returns:** true if success, false if failed

**Example:**

```
myfunc = function(filename)
{
}
FileEnumerate("myfolder", myfunc);
```

---

## FileExists

Checks if a file exists by name. This function is restricted to files under the user folder.

**Parameters:** (filename)

**Returns:** true if exists, false if not

**Example:**

```
if(FileExists("myfile.txt"))
{
}
```

---

## Newline

Returns a NewLine custom type. Used for writing newlines in text formatted files.

**Parameters:** none

**Returns:** none

**Example:**

```
f = File();  
f.Open("myfile.txt", "text", false);  
if(f.IsOpen())  
{  
    f.Write("Some test data", NewLine());  
}  
f.Close();
```

---

## File

Creates a new File Object.

**Parameters:** none

**Returns:** none

**Example:**

```
f = File();
```

---

## Close

Closes the file object and commits changes to disk.

**Parameters:** none

**Returns:** none

**Example:**

```
f.Close();
```



---

## EndOfFile

Checks if the File Object is at the end of the file. Usefor for read operations.

**Parameters:** none

**Returns:** true if end of file, false if not.

**Example:**

```
eof = f.EndOfFile();
```

---

## FileSize

Gets the file size of the file, in bytes.

**Parameters:** none

**Returns:** size of file in bytes

**Example:**

```
size = f.FileSize();
```

---

## Flush

Flushes the file buffer to disk.

**Parameters:** none

**Returns:** none

**Example:**

```
f.Flush();
```

---

## IsOpen

Checks if the file is currently open. Usually used after a call to Open.

**Parameters:** none

**Returns:** true if file is open, false if not

**Example:**

```
if(f.IsOpen())  
{  
}
```

---

## Open

Creates a new File Object.

**Parameters:** (filename, "text"/"binary", readonly<optional>, append<optional>)

**Returns:** true if success, false if failed.

**Example:**

```
if(f.Open("myfile.txt", "text", false))  
{  
}
```

---

## ReadFloat

Reads a float from the file.

**Parameters:** none

**Returns:** float read, or null if there was an error

**Example:**

```
num = f.ReadFloat();
```

---

## ReadInt

Reads an integer from the file.

**Parameters:** none

**Returns:** integer read, or null if there was an error

**Example:**

```
num = f.ReadInt();
```

---

## ReadLine

Reads a string from a file until a newline or end of file is encountered.

**Parameters:** none

**Returns:** string read, or null if there was an error

**Example:**

```
str = f.ReadLine();
```

---

## ReadString

Reads a string from a file.

**Parameters:** none

**Returns:** string read, or null if there was an error

**Example:**

```
str = f.ReadString();
```

---

## Seek

Seeks the read/write position to a specified offset.

**Parameters:** (byte offset to seek to)

**Returns:** none

**Example:**

```
f.Seek(100); // seek 100 bytes into file
```

---

## Tell

Returns the current offset of the read/write position in the file.

**Parameters:** none

**Returns:** byte position in file

**Example:**

```
t = f.Tell();
```

---

## Write

Writes a value of varying types to the file, in whatever file mode was used to open the file.

## Parameters: (...)

This function can take any number of parameters, of types integer, float, string, or NewLine();

**Returns:** none

## Example:

```
f = File();  
f.Open("myfile.txt", "text", false);  
if(f.IsOpen())  
{  
    f.Write("Some test data", NewLine());  
}  
f.Close();
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=System\\_Library](http://omni-bot.com/wiki/index.php?title=System_Library)"

This page has been accessed 196 times. This page was last modified 04:13, 28 May 2007.

# AABB

From Omni-bot Wiki

---

## Contents

- 1 CenterPoint
- 2 Contains
- 3 Expand
- 4 FindIntersection
- 5 GetAxisLength
- 6 Intersects
- 7 IsZero
- 8 Scale
- 9 Set
- 10 SetCenter

## CenterPoint

Gets the center of the bounding box.

**Parameters:** none

**Returns:** Vector3 - center point of the AABB.

**Example:**

```
b = AABB();  
v = b.CenterPoint();
```

---

## Contains

Checks if the bounding box contains a point.

**Parameters:** (Vector3)

**Returns:** true if Vector3 parameter is contained within AABB, false if not.

### Example:

```
b = AABB();  
v = Vector3();  
if(b.Contains(v))  
{  
    print("v is inside b");  
}
```

---

## Expand

Expands the bounding box to contain a position.

**Parameters:** (Vector3)

**Returns:** none

### Example:

```
b = AABB();  
v1 = Vector3(40,50,60);  
v2 = Vector3(30,10,90);  
b.Expand(v1);  
b.Expand(v2);
```

---

## FindIntersection

Gets the AABB overlap of 2 bounding boxes.

**Parameters:** (AABB)

**Returns:** AABB the bounds that overlaps the bounding box. Null if there is no overlap.

### Example:

```
b = AABB();  
b2 = AABB();  
overlapAABB = b.FindIntersection(b2);
```

```
if(overlapAABB)
{
    print("b overlaps b2");
}
```

---

## GetAxisLength

Gets the length of an axis of the bounding box.

**Parameters:** ("x"),("y"),or ("z")

**Returns:** float - length of requested AABB axis.

**Example:**

```
b = AABB();
xlen = b.GetAxisLength("x");
```

---

## Intersects

Checks if 2 AABBs intersect.

**Parameters:** (AABB)

**Returns:** true if AABB overlaps this AABB, false if not.

**Example:**

```
b = AABB();
b2 = AABB();
if(b.Intersects(b2))
{
    print("b touches b2");
}
```

---

## IsZero



Checks if the AABB is has no volume.

**Parameters:** none

**Returns:** true if the AABB is not defined(0,0,0 for mins and maxs)

**Example:**

```
b = AABB();  
if(b.IsZero())  
{  
}
```

---

## Scale

Scales the AABB by some amount in every direction.

**Parameters:** (float)

**Returns:** none

**Example:**

```
b = AABB();  
b.Scale(10);
```

---

## Set

Initializes the AABB mins and maxs.

**Parameters:** (Vector3, Vector3)

**Returns:** none

**Example:**

```
b = AABB();  
min = Vector3(-10, -10, -10);  
max = Vector3(10, 10, 10);  
b.Set(min, max);
```

---

## SetCenter

Translates the bounding box to some position. Usually used to move a local space bounding box to world space.

**Parameters:** (Vector3)

**Returns:** none

### Example:

```
b = AABB();  
c = Vector3(10, 10, 10);  
b.SetCenter(c);
```

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=AABB>"

This page has been accessed 286 times. This page was last modified 19:32, 26 May 2007.

# Bot

From Omni-bot Wiki

## Contents

- 1 Bot Properties
  - 1.1 AimDamping
  - 1.2 AimPersistance
  - 1.3 AimStiffness
  - 1.4 AimTolerance
  - 1.5 Armor
  - 1.6 FieldOfView
  - 1.7 Health
  - 1.8 MaxArmor
  - 1.9 MaxHealth
  - 1.10 MaxTurnSpeed
  - 1.11 MaxViewDistance
  - 1.12 MemorySpan
  - 1.13 Name
  - 1.14 ReactionTime
  - 1.15 Enemy Territory Addendum
    - 1.15.1 TargetBreakableDist
- 2 Bot Functions
  - 2.1 AddSignalThread
  - 2.2 AddScriptGoal
  - 2.3 RemoveState
  - 2.4 SetStateEnabled
  - 2.5 BlockForWeaponChange
  - 2.6 ChangeClass
  - 2.7 ChangeTeam
  - 2.8 DistanceTo
  - 2.9 DumpBotTable
  - 2.10 Enable
  - 2.11 ExecCommand
  - 2.12 Weapons/Ammo
    - 2.12.1 CanSnipe
    - 2.12.2 EnableShooting
    - 2.12.3 GetWeapon
    - 2.12.4 GetBestWeapon
    - 2.12.5 GetRandomWeapon
    - 2.12.6 GetCurrentWeapon
    - 2.12.7 GetCurrentAmmo

- 2.12.8 GetMostDesiredAmmo
- 2.12.9 FireWeapon
- 2.12.10 HasAmmo
- 2.12.11 HasWeapon
- 2.12.12 IsWeaponCharged
- 2.12.13 SelectBestWeapon
- 2.13 GetAllType
- 2.14 GetClass
- 2.15 GetEyePosition
- 2.16 GetFacing
- 2.17 GetGameEntity
- 2.18 GetGameId
- 2.19 GetHealthPercent
- 2.20 GetHighLevelGoalName
- 2.21 GetLastTarget
- 2.22 GetNearest
- 2.23 GetNearestAlly
- 2.24 GetNearestEnemy
- 2.25 GetPosition
- 2.26 GetTarget
- 2.27 GetTargetInfo
- 2.28 GetTeam
- 2.29 GetVelocity
- 2.30 HasTarget
- 2.31 HasEntityFlag
- 2.32 HasLineOfSightTo
- 2.33 HasPowerUp
- 2.34 HoldButton
- 2.35 IgnoreTarget
- 2.36 IgnoreTargetForTime
- 2.37 InFieldOfView
- 2.38 IsAllied
- 2.39 IsStuck
- 2.40 MoveTowards
- 2.41 PressButton
- 2.42 ReleaseButton
- 2.43 ReloadProfile
- 2.44 RemoveSignalThread
- 2.45 ResetStuckTime
- 2.46 ResetSubGoals
- 2.47 Say
- 2.48 SayTeam
- 2.49 SayVoice

- 2.50 SelectWeapon
- 2.51 SetDebugFlag
- 2.52 SetDesiredFacing
- 2.53 SetGoalProperty
- 2.54 SetWatchEntity
- 2.55 Signal
- 2.56 ToLocalSpace
- 2.57 ToWorldSpace
- 2.58 Enemy Territory Addendum
  - 2.58.1 CanSnipe
  - 2.58.2 ChangePrimaryWeapon
  - 2.58.3 ChangeSecondaryWeapon
  - 2.58.4 ChangeSpawnPoint
  - 2.58.5 GetCursorHint
  - 2.58.6 GetReinforceTime
  - 2.58.7 GetSkills
  - 2.58.8 GetStat

## Bot Properties

---

Unless otherwise noted, all properties can be get and set.

### AimDamping

Effects the acceleration and turn rate.

### AimPersistence

The time(in seconds) the bot will continue to aim at the last position of a target before aborting.

### AimStiffness

Effects the acceleration of turning.

### AimTolerance

The tolerance(in degrees) the bot tries to aim at stuff with.

## **Armor**

Current Armor

## **FieldOfView**

The angle(in degrees) the bot is capable of 'seeing'.

## **Health**

Current health.

## **MaxArmor**

Max Armor

## **MaxHealth**

Max Health

## **MaxTurnSpeed**

The maximum speed(in degrees/second) the bot can rotate his aim at.

## **MaxViewDistance**

The max distance(in game units) the bot is capable of seeing.

## **MemorySpan**

The time(in seconds) taken for the memory about an entity to become stale.

## **Name**

Name of the bot.

## **ReactionTime**

The time(in seconds) it takes the bot to begin responding to a newly sensed threat.

---

# Enemy Territory Addendum

---

## TargetBreakableDist

The maximum distance(in game units), the bot will attempt to target breakable entities.

---

## Bot Functions

---

### AddSignalThread

Adds a thread id to the bots list of threads to forward signals to. The thread will also be destroyed when the bot is destroyed(kicked).

**Parameters:** (threadId, true/false autodelete<optional, default true>)

**Returns:** none

#### Example:

```
t = thread(somefunction);
b.AddSignalThread(t);
// OR
b.AddSignalThread(t, false); // don't auto delete the thread.
```

**See Also:** RemoveSignalThread, Signal

---

### AddScriptGoal

Adds a goal to the bots behavior tree by name. Useful when a script goal has AutoAdd set to false, so you can more explicitly control when a bot gets the goal.

**Parameters:** (name of script goal to add to bot behavior tree)

**Returns:** true on success, false on failure

#### Example:

```
b.AddScriptGoal( "TestBot" );
```

---

## RemoveState

Removes a state from a bots behavior tree by name.

**Parameters:** (name of script goal to remove from bot behavior tree)

**Returns:** none

**Example:**

```
b.RemoveState( "TestBot" );
```

---

## SetStateEnabled

Enables/Disables a state by name.

**Parameters:** (name of state, bool enable)

**Returns:** none

**Example:**

```
b.SetStateEnabled( "Attack", false ); // disable attack state
```

---

## BlockForWeaponChange

Blocks the script thread and waits for a change to a specific weapon.

**Parameters:** (weaponId)

**Returns:** none

**Example:**



```
b.BlockForWeaponChange(WEAPON.MEDKIT);
```

---

## ChangeClass

Changes the bot to a specified class.

**Parameters:** (classId)

**Returns:** none

**Example:**

```
b.ChangeClass(CLASS.SOLDIER);
```

---

## ChangeTeam

Changes the bot to a specified team.

**Parameters:** (teamId)

**Returns:** none

**Example:**

```
b.ChangeTeam(Team.AXIS);
```

---

## DistanceTo

Overloaded utility function for simplified distance checks.

**Parameters:** (Vector3, useEyePosition<default false>)

**Parameters:** (GameEntity/GameId, useEyePosition<default false>)

**Returns:** float - distance from bot to Vector3, GameEntity, or Gameld

**Example:**

```
v = Vector3(10,20,30);
target = b.GetTarget();
// distance from bot to target entity
d1 = b.DistanceTo(target);
// distance from bot to world position
d2 = b.DistanceTo(v);
// distance from bot eye to world position
d3 = b.DistanceTo(v, true);
```

---

## DumpBotTable

Dumps a file with the bots table to a file in the user/ folder.

**Parameters:** (filename)

**Returns:** none

**Example:**

```
b.DumpBotTable("foo.txt");
```

---

## Enable

Disables all thinking for the bot.

**Parameters:** (true/false)

**Returns:** none

**Example:**

```
b.Enable(false);
```

---

## ExecCommand

Executes a string command on the bot as if the bot executed a console command.

**Parameters:** (command)

**Returns:** none

**Example:**

```
b.ExecCommand( "kill" );
```

---

## Weapons/Ammo

### CanSnipe

Checks if the bot can snipe based on the internally implemented snipe checks. Vary by game. Typically involves checking if the bot has a snipe enabled weapon and has ammo.

**Parameters:** none

**Returns:** true if the bot can snipe, else false

**Example:**

```
if(b.CanSnipe()) {}
```

---

### EnableShooting

Disables all shooting for the bot.

**Parameters:** (true/false)

**Returns:** none

### Example:

```
b.EnableShooting(false);
```

---

## GetWeapon

Gets a reference to a specific weapon from the bot.

**Parameters:** (weaponId)

**Returns:** Weapon reference, or NULL if the bot doesn't have the weapon.

### Example:

```
wpn = b.GetWeapon(WEAPON.SHOTGUN);  
wpn.PrimaryFire.AimOffset = Vector3(0,0,10);
```

---

## GetBestWeapon

Gets a weapon id for the best weapon versus current target or a specific entity.

**Parameters:** none - uses current target

**Parameters:** (entity) evaluate weapon versus specific target entity

**Returns:** weapon id of weapon that is chosen.

### Example:

```
best = b.GetBestWeapon();  
// OR  
best = b.GetBestWeapon(someEnt);
```

---

## GetRandomWeapon

Gets a random weapon id from the list of weapons the bot currently has.

**Parameters:** none

**Returns:** random weapon id

**Example:**

```
randwpn = b.GetRandomWeapon( );
```

---

## GetCurrentWeapon

Gets the current weapon id for the bot.

**Parameters:** none

**Returns:** weapon Id for bot

**Example:**

```
mywpn = b.GetCurrentWeapon( );  
if (mywpn == WEAPON.SHOTGUN)  
{  
}
```

---

## GetCurrentAmmo

Gets the ammo information for a weapon. **Parameters:** none

**Parameters:** (firemode)

**Parameters:** (firemode, weaponId)

**Returns:** Ammo info table

**Example:**

```
// Get primary ammo for current weapon
a1 = b.GetCurrentAmmo();
// Get secondary ammo for current weapon
a2 = b.GetCurrentAmmo(1);
// Get primary ammo for another weapon.
a3 = b.GetCurrentAmmo(0, WEAPON.SHOTGUN);

print("Current Ammo:", a1.CurrentAmmo);
print("Max Ammo:", a1.MaxAmmo);
print("Current Clips:", a1.CurrentClips);
print("Max Clips:", a1.MaxClips);
```

---

## GetMostDesiredAmmo

Gets information about the currently most needed ammo.

**Parameters:** (table)

**Returns:** none

**Example:**

```
mostNeededAmmo = table();
b.GetMostDesiredAmmo(mostNeededAmmo);
// mostNeededAmmo.Desire - How badly this ammo is needed, rough 0-1 scale.
// mostNeededAmmo.AmmoType - The ammo Id.
```

---

## FireWeapon

Fires the current weapon.

**Parameters:** none

**Returns:** none

**Example:**

```
b.FireWeapon();
```

---

## HasAmmo

Checks if the bot has a certain ammo type.

**Parameters:** () - If no parameters passed, checks current weapon ammo.

**Parameters:** (ammold)

**Returns:** true/false if the bot has ammo.

**Example:**

```
if (b.HasAmmo ( ) )  
{  
}  
if (b.HasAmmo ( AMMO . SHELLS ) )  
{  
}
```

---

## HasWeapon

Checks if the bot has a specific weapon.

**Parameters:** (weaponId)

**Returns:** true if the bot has it, false if not.

**Example:**

```
if (b.HasWeapon ( WEAPON . SHOTGUN ) )  
{  
}
```

---

## IsWeaponCharged

Checks if a weapon is *charged*. A charged weapon is one that is capable of being used that might occasionally be unavailable for use due to lack of character 'charge', whether that be a gameplay charge bar, stamina, skill, etc.

**Parameters:** (weaponId, firemode<optional>)

**Returns:** true if weapon is charged, false if not.

**Example:**

```
if (b.IsWeaponCharged(WEAPON.MORTAR, 0))  
{  
}
```

---

## SelectBestWeapon

Used to calculate the best weapon for the bots current target, or a target passed as a parameter.

**Parameters:** () - If no parameter, uses the current target.

**Parameters:** (entity/gameld)

**Returns:** weaponId of best weapon.

**Example:**

```
weaponId = b.SelectBestWeapon(someentity);
```

---

## GetAllType

Fills a table with all known entities that match a desired criteria.

**Parameters:** (category, class, table)

**Returns:** none

**Example:**



```
// Get all known soldier players.
players = table();
b.GetAllType(CAT.PLAYER, CLASS.SOLDIER, players);
foreach ( i and ent in players )
{
    print(ent);
}
```

---

## GetClass

Gets the current class id for the bot.

**Parameters:** none

**Returns:** class Id for bot

### Example:

```
myclass = b.GetClass();
if(myClass == CLASS.SOLDIER)
{
}
```

---

## GetEyePosition

Gets the world position of the bots 'eye'

**Parameters:** none

**Returns:** Vector3 eye position

### Example:

```
eyepos = b.GetEyePosition();
```

---

## GetFacing

Gets the facing vector for the bots aim.

**Parameters:** none

**Returns:** Vector3 facing vector

**Example:**

```
facing = b.GetFacing();  
pointinfront = b.GetEyePosition() + facing * 100;
```

---

## GetGameEntity

Gets the game entity for the bot.

**Parameters:** none

**Returns:** entity

**Example:**

```
ent = b.GetGameEntity();
```

---

## GetGameId

Gets the game id for the bot.

**Parameters:** none

**Returns:** entity

### Example:

```
gameid = b.GetGameId();
```

---

## GetHealthPercent

Gets the bot's current health percentage.

**Parameters:** none

**Returns:** 0.0 - 1.0 health scalar

### Example:

```
if(bot.GetHealthPercent() < 0.5)
{
    // < 50% health
}
```

---

## GetHighLevelGoalName

Gets the bot's current high level goal name.

**Parameters:** none

**Returns:** goal name, or NULL if the bot doesn't have a high level goal.

### Example:

```
g = bot.GetHighLevelGoalName;
```

---

## GetLastTarget

Gets the target the bot had before the current target.

**Parameters:** none

**Returns:** entity OR null

**Example:**

```
last = b.GetLastTarget();
```

---

## GetNearest

Gets the nearest target matching the desired criteria, friend or foe.

**Parameters:** (category)

**Parameters:** (category, classId)

**Returns:** entity OR null

**Example:**

```
// get nearest player, defaults to any class within that category  
p = b.GetNearest(CAT.PLAYER);  
// get nearest soldier player  
s = b.GetNearest(CAT.PLAYER, CLASS.SOLDIER);
```

---

## GetNearestAlly

Gets the nearest target matching the desired criteria, only allies

**Parameters:** (category)

**Parameters:** (category, classId)

**Returns:** entity OR null

### Example:

```
// get nearest player, defaults to any class within that category
p = b.GetNearestAlly(CAT.PLAYER);
// get nearest soldier player
s = b.GetNearestAlly(CAT.PLAYER, CLASS.SOLDIER);
```

---

## GetNearestEnemy

Gets the nearest target matching the desired criteria, only enemies

**Parameters:** (category)

**Parameters:** (category, classId)

**Returns:** entity OR null

### Example:

```
// get nearest player, defaults to any class within that category
p = b.GetNearestEnemy(CAT.PLAYER);
// get nearest soldier player
s = b.GetNearestEnemy(CAT.PLAYER, CLASS.SOLDIER);
```

---

## GetPosition

Gets the world position of the bot.

**Parameters:** none

**Returns:** Vector3 world position

### Example:

```
mypos = b.GetPosition();
```

---

## GetTarget

Gets the current target.

**Parameters:** none

**Returns:** entity OR null

**Example:**

```
target = b.GetTarget();
```

---

## GetTargetInfo

Returns the TargetInfo for an entity.

**Parameters:** (entity/gameld)

**Returns:** TargetInfo for entity.

**Example:**

```
ti = b.GetTargetInfo(b.GetTarget());
```

**See Also:** TargetInfo

---

## GetTeam

Gets the current team the bot is on.

**Parameters:** none

**Returns:** team #

**Example:**

```
ti = b.GetTargetInfo(b.GetTarget());
```

---

## GetVelocity

Gets the world velocity of the bot.

**Parameters:** none

**Returns:** Vector3 world velocity

**Example:**

```
myvel = b.GetVelocity();
```

---

## HasTarget

Checks if the bot has a current target

**Parameters:** none

**Returns:** true if bot has a target

**Example:**

```
// checking 1 flag
if(b.HasTarget())
{
}
```

---

## HasEntityFlag

Checks if the bot has a given entity flag. This function can take any number of flags to check, and will return true if the bot has all the flags, or false if it is missing any.

**Parameters:** (entityflag, ...)

**Returns:** true if bot has all flags, false if not

**Example:**

```
// checking 1 flag
if(b.HasEntityFlag(ENTFLAG.RELOADING))
{
}

// checking multiple flags.
if(b.HasEntityFlag(ENTFLAG.INWATER, ENTFLAG.UNDERWATER))
{
}
```

---

## HasLineOfSightTo

This functions checks whether the bot has line of sight to a 3d position. This function does not account for field of view, simply does a raycast for obstructions between the bots eye position and the provided position. To account for field of view, use <InFieldOfView>. If an entity or gameld is provided as the and parameter, the function will return true if the raycast hits nothing on its way to the position OR if it hits the entity that is passed.

**Parameters:** (Vector3 position)

**Parameters:** (Vector3 position, entity)

**Returns:** true if the bot has line of sight to the position or entity passed. false if the bots view is obstructed.

**Example:**

```
v = Vector3(10,10,10); // some position
if(b.HasLineOfSightTo(v))
{
}
```

---

## HasPowerUp

Checks if the bot has a given powerup. This function can take any number of flags to check,



and will return true if the bot has all the flags, or false if it is missing any.

**Parameters:** (powerup, ...)

**Returns:** true if bot has all powerups, false if not

**Example:**

```
// checking 1 flag
if(b.HasPowerUp(POWERUP.INVINCIBLE))
{
}

// checking multiple flags.
if(b.HasPowerUp(POWERUP.INVINCIBLE, POWERUP.QUADDAMAGE))
{
}
```

---

## HoldButton

Makes the bot 'press' a button, and hold it for an amount of time. This function can take any number of buttons as parameters, and will apply the effect to all of them. The time value is **ALWAYS** the last parameter.

**Parameters:** (buttonId, ..., time in seconds)

**Returns:** none

**Example:**

```
// hold crouch for 5 seconds
b.HoldButton(BTN.CROUCH, 5);

// hold multiple buttons for 5 seconds
b.HoldButton(BTN.CROUCH, BTN.SPRINT, 5);
```

**See Also:** PressButton, ReleaseButton

---

## IgnoreTarget

This function causes the bot to ignore a specific entity for targeting for some duration of time.

**Parameters:** (entity/gameld, seconds to ignore)

**Returns:** none

**Example:**

```
// ignore the entity for some map goal, so we wont target and shoot at it.
mg = GetGoal("some map goal");
b.IgnoreTarget(mg.GetEntity(), 99999999);
```

---

## IgnoreTargetForTime

**See:** IgnoreTarget

---

## InFieldOfView

Checks whether a position is within the bots current FieldOfView. This function can take an optional field of view(in degrees) to check. If not provided, it will use the bots current FieldOfView.

**Parameters:** (Vector3 position)

**Parameters:** (Vector3 position, fov angles)

**Returns:** Vector3 - center point of the AABB.

**Example:**

```
v = Vector3(10,10,10);
// check if in default field of view
if(b.InFieldOfView(v))
{
}
// check if in custom field of view
if(b.InFieldOfView(v, 180))
{
}
```

```
}
```

---

## IsAllied

Checks if the bot is allied with a given entity.

**Parameters:** (entity/gameId)

**Returns:** true if allied, false if not.

**Example:**

```
if(b.IsAllied(someentity))  
{  
}
```

---

## IsStuck

Checks if the bot considers himself to be stuck. Stuckness is typically defined as an insignificant amount of movement over a short period of time. The time can be optionally passed to the function for flexibility.

**Parameters:** () - If no params passed, the time used is 0.5 seconds.

**Parameters:** (time in seconds)

**Returns:** true if bot is *stuck*, false if not.

**Example:**

```
if(b.IsStuck())  
{  
}  
// or check stuckness for 1.5 seconds.  
if(b.IsStuck(1.5))  
{
```

```
}
```

---

## MoveTowards

Causes the bot to blindly move towards a target. This function does no path finding. It's primary use is short range movement when you are reasonably sure the bot can run right at the target.

**Parameters:** (Vector3 position) - if distance tolerance not specified, default of 32 units is used.

**Parameters:** (Vector3 position, distance tolerance)

**Returns:** true if the bot is within the distance tolerance, false if not.

### Example:

```
// get within 64 units of this position
v = Vector3(10,10,10);
while(b.MoveTowards(v, 64) == false)
{
    yield();
}
```

---

## PressButton

Makes the bot 'press' a button. This function can take any number of buttons as parameters, and will apply the effect to all of them. Buttons that are *pressed* using this function are released automatically for the next bot update.

**Parameters:** (buttonId, ...)

**Returns:** none

### Example:

```
// press crouch
```

```
b.PressButton(BTN.CROUCH);  
// press multiple buttons  
b.PressButton(BTN.CROUCH, BTN.SPRINT);
```

**See Also:** HoldButton, ReleaseButton

---

## ReleaseButton

Makes the bot 'release' a button. This function can take any number of buttons as parameters, and will apply the effect to all of them.

**Parameters:** (buttonId, ...)

**Returns:** none

**Example:**

```
// release crouch  
b.ReleaseButton(BTN.CROUCH);  
// release multiple buttons  
b.ReleaseButton(BTN.CROUCH, BTN.SPRINT);
```

**See Also:** HoldButton, PressButton

---

## ReloadProfile

Reloads the bots profile. The profile is any script associated with the bots name.

**Parameters:** none

**Returns:** none

**Example:**

```
b.ReloadProfile();
```

---

## RemoveSignalThread

Removes a thread id from the bots signal list, so that signal events for the bots won't be sent to the thread.

**Parameters:** (threadId)

**Returns:** none

**Example:**

```
// remove the current thread from the bots signal list
b.RemoveSignalThread(threadId());
```

**See Also:** AddSignalThread, Signal

---

## ResetStuckTime

Reset the stuck timer.

**Parameters:** none

**Returns:** none

**Example:**

```
b.ResetStuckTime();
```

---

## ResetSubGoals

Resets all the current goals the bot is using. This will cause an immediate re-evaluation of available goals. Normally used to force a bot to re-evaluate its goals after you've made changes to conditions that would effect choosing a goal.

**Parameters:** none

**Returns:** none

**Example:**

```
b.ResetSubGoals();
```

---

## Say

Chat function used for bots to *say* messages through the normal chat channels of the game. For flexibility, this function can take any number and type of parameters, which it will convert to a single long string and use as the chat message.

**Parameters:** (string, ...)

**Returns:** none

**Example:**

```
b.Say("My name is ", b.Name);
```

**See Also:** SayTeam, SayVoice

---

## SayTeam

Chat function used for bots to *say* messages through the team chat channel of the game. For flexibility, this function can take any number and type of parameters, which it will convert to a single long string and use as the chat message.

**Parameters:** (string, ...)

**Returns:** none

**Example:**

```
b.SayTeam("My name is ", b.Name);
```

**See Also:** Say, SayVoice

---

## SayVoice

Broadcasts a voice macro to the game. Used with global VOICE table values.

**Parameters:** (macroId)

**Returns:** none

**Example:**

```
b.SayVoice(VOICE.NEED_MEDIC);
```

**See Also:** Say, SayTeam

---

## SelectWeapon

Tells the bot to choose a specific weapon. Typically you should set the bot to script controlled weapons, with SetScriptControlledWeapons, or the default behavior of the bot can override the weapon selection from this function.

**Parameters:** (weaponId)

**Returns:** none

**Example:**

```
b.SelectWeapon(WEAPON.MEDKIT);
```

---

## SetDebugFlag

Enables a debug flag for the bot. See the global DEBUG table for available options. Enabling debug flags typically cause various debug visualizations to render in order to help identify whats going on with various systems of the bot.

**Parameters:** (flag, true/false to enable)

**Returns:** none

**Example:**

```
b.SetDebugFlag(DEBUG.AIM);
```



---

## SetDesiredFacing

Sets a desired world facing for the bot. A desired facing is a direction the bot will attempt to look towards in the absence of a target to aim at.

**Parameters:** (Vector3 facing)

**Returns:** none

**Example:**

```
// face to the world right
b.SetDesiredFacing(Vector3(1,0,0));
```

---

## SetGoalProperty

Sets a named property of a goal. This function is currently the only way to set properties of built in goals. The properties and expected values vary by the goal. This function will be deprecated for 0.7.

**Parameters:** (string goalname, string propertyname, value)

**Returns:** true if successful, false if not.

**Example:**

```
b.SetGoalProperty("ATTACK", "MinCampTime", 10);
b.SetGoalProperty("ATTACK", "MaxCampTime", 20);
```

---

## SetWatchEntity

Sets an entity the bot should watch. Bot should rotate to watch the entity at all times.

**Parameters:** (entity/gameId)

**Returns:** none

**Example:**

```
b.SetWatchEntity(someentity);
```

**See Also:** GetWatchEntity, ClearWatchEntity

---

## Signal

Sends a signal to the bots signal threads. This function is different from the global signal function built into Game Monkey script, in that it only signals the threads belonging to this specific bot.

**Parameters:** (anything, ...)

**Returns:** none

**Example:**

```
b.Signal("blah", 10);
```

**See Also:** RemoveSignalThread, AddSignalThread

---

## ToLocalSpace

Converts a vector into local bot space. This results in a position relative to the bots current position and rotation. This function is useful for simplifying some types of logic.

- x axis is side to side.
- y axis is forward and back.
- z axis is up and down.

**Parameters:** (Vector3)

**Returns:** Vector3 local space position.

**Example:**

```
v = Vector3(4356,342,276); // some world position.
v2 = b.ToLocalSpace(v);
if(v2.y < 0)
{
    // the position is behind the bot.
}
if(v2.x < 0)
{
    // the position is to the right of the bot
}
```

---

## ToWorldSpace

Converts a vector from local bot space to world space.

- x axis is side to side.
- y axis is forward and back.
- z axis is up and down.

**Parameters:** (Vector3)

**Returns:** Vector3 world space position.

**Example:**

```
// slightly forward and to the right.
v = Vector3(10,10,0);
v2 = b.ToWorldSpace(v);
```

---

## Enemy Territory Addendum

### CanSnipe

Checks if the bot is capable of sniping. Currently this function works as follows. If bot is CLASS.COVERTOPS and the bot has ammo for the FG42\_SCOPE, K43\_SCOPE, or

GARAND\_SCOPE, this returns true.

**Parameters:** (true/false)

**Returns:** true if bot can snipe, false is not.

**Example:**

```
if (b.CanSnipe())  
{  
}
```

---

## ChangePrimaryWeapon

Selects a new primary weapon for the bot. Should take effect next spawn.

**Parameters:** (weaponId)

**Returns:** true if successful, false if not

**Example:**

```
b.ChangePrimaryWeapon(WEAPON.PANZERFAUST);
```

**See Also:** ChangeSecondaryWeapon

---

## ChangeSecondaryWeapon

Selects a new secondary weapon for the bot. Should take effect next spawn.

**Parameters:** (weaponId)

**Returns:** true if successful, false if not

**Example:**

```
b.ChangeSecondaryWeapon(WEAPON.LUGER);
```

**See Also:** [ChangePrimaryWeapon](#)

---

## ChangeSpawnPoint

Changes the desired spawn point for the bot to respawn at.

**Parameters:** (spawnpoint #)

**Returns:** none

**Example:**

```
b.ChangeSpawnPoint(2);
```

---

## GetCursorHint

When a player is near enough something to get a hint icon, such as for dynamiting, this function can access what hint is showing. Some of these hints may or may not ever show up, they were taken as-is from the ET SDK.

- NONE - 0
- PLAYER - 1
- ACTIVATE - 2
- DOOR - 3
- DOOR\_ROTATING - 4
- DOOR\_LOCKED - 5
- DOOR\_ROTATING\_LOCKED - 6
- MG42 - 7
- BREAKABLE - 8
- BREAKABLE\_DYNAMITE - 9
- CHAIR - 10
- ALARM - 11
- HEALTH - 12
- TREASURE - 13
- KNIFE - 14
- LADDER - 15
- BUTTON - 16
- WATER - 17
- CAUTION - 18
- DANGER - 19

- SECRET - 20
- QUESTION - 21
- EXCLAMATION - 22
- CLIPBOARD - 23
- WEAPON - 24
- AMMO - 25
- ARMOR - 26
- POWERUP - 27
- HOLDABLE - 28
- INVENTORY - 29
- SCENARIC - 30
- EXIT - 31
- NOEXIT - 32
- PLYR\_FRIEND - 33
- PLYR\_NEUTRAL - 34
- PLYR\_ENEMY - 35
- PLYR\_UNKNOWN - 36
- BUILD - 37
- DISARM - 38
- REVIVE - 39
- DYNAMITE - 40
- CONSTRUCTIBLE - 41
- UNIFORM - 42
- LANDMINE - 43
- TANK - 44
- SATCHELCHARGE - 45
- LOCKPICK - 46

**Parameters:** (table)

**Returns:** none

**Example:**

```
hint = table();
b.GetCursorHint(hint);

print(hint.type); // one of the values above
print(hint.value); // usually a health associated with certain types
```

---

**GetReinforceTime**

Gets the current time left(in seconds) before reinforcements spawn for the bots team.

**Parameters:** none

**Returns:** seconds left till reinforcements spawn

**Example:**

```
if(b.GetReinforceTime() < 2)
{
    // 2 seconds to reinforcements
}
```

---

## GetSkills

Gets the current skill values and puts them into the table parameter.

**Parameters:** (table)

**Returns:** none

**Example:**

```
skills = table();
b.GetSkills(skills);
print("Battle Sense:", skills[SKILL.BATTLE_SENSE]);
print("Engineering:", skills[SKILL.ENGINEERING]);
print("First Aid:", skills[SKILL.FIRST_AID]);
print("Signals:", skills[SKILL.SIGNALS]);
print("Light Weapons:", skills[SKILL.LIGHT_WEAPONS]);
print("Heavy Weapons:", skills[SKILL.HEAVY_WEAPONS]);
print("Coverops:", skills[SKILL.COVERTOPS]);
```

**See Also:** SKILL

---

## GetStat

Gets a current stat from the bot by name.

**Parameters:** (string) the stat name

**Returns:** the value of the stat

valid stat names are kills, deaths, and xp for ET

**Example:**

```
xp = b.GetStat("xp");  
print("Bot xp value is",xp);
```

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Bot>"

This page has been accessed 1,045 times. This page was last modified 06:22, 25 August 2008.



# Blackboard

From Omni-bot Wiki

The blackboard is basically a global database that can be used to store generic data records. Items on the blackboard are made up of several common properties, and any number of additional user properties that can be passed as part of the table.

These properties are **required** for every record posted.

- Owner - The owner id
- Target - The target id
- Duration - How long does it take for the record to expire
- DeleteOnExpire - Whether the record is automatically deleted when it expires

What each property means is normally dependent on the type of blackboard record. The Owner is typically the game id of the bot, the target is normally an id that can be mapped to another object, such as a MapGoal serial number. The other properties control the records lifetime.

## Contents

- 1 Blackboard Functions
  - 1.1 MakeKey
  - 1.2 PostRecord
  - 1.3 GetRecords
  - 1.4 GetNumRecords
  - 1.5 RecordExistsOwner
  - 1.6 RecordExistsTarget
  - 1.7 RemoveByPoster
  - 1.8 RemoveByTarget
  - 1.9 PrintBlackboard

## Blackboard Functions

### MakeKey

Makes a numeric blackboard item key. Normally you will inject this value into the global BB table with the rest of the blackboard keys.

**Parameters:** none

**Returns:** numeric key.

**Example:**

```
BB.MYNEWKEY = Blackboard.MakeKey();
```

---

## PostRecord

Posts a record to the blackboard, keyed to a certain type, and storing arbitrary values.

**Parameters:** (blackboard key, table)

**Returns:** none

**Example:**

```
mytable =  
{  
    Owner = b.GetGameId(),  
    Target = 100,  
    Duration = 10,  
    DeleteOnExpire = true,  
  
    someinfo = 10,  
    someinfo2 = 20,  
};  
Blackboard.PostRecord(BB.MYNEWKEY, mytable);
```

---

## GetRecords

Retrieves all records matching a type from global Blackboard

**Parameters:** (blackboard key)

**Returns:** table of all blackboard records, null if non exist.

**Example:**

```
records = Blackboard.GetRecords(BB.MYNEWKEY);  
if(records)  
{  
    foreach ( i and rcd in records )  
    {  
        print(rcd.someinfo);  
        print(rcd.someinfo2);  
    }  
}
```

---

## GetNumRecords

Retrieves the number of records of a given type.

**Parameters:** (blackboard key)

**Returns:** number of records that exist of this type.

**Example:**

```
numrecords = Blackboard.GetNumRecords(BB.MYNEWKEY);
```

---

## RecordExistsOwner

Checks if a record exists for a certain key that matches a given owner id.

**Parameters:** (blackboard key, owner id #)

**Returns:** number of records that exist of this type.

**Example:**

```
if(Blackboard.RecordExistsOwner(BB.MYNEWKEY, b.GetGameId()))  
{  
}
```

---

## RecordExistsTarget

Checks if a record exists for a certain key that matches a given target id.

**Parameters:** (blackboard key, target id #)

**Returns:** number of records that exist of this type.

**Example:**

```
if (Blackboard.RecordExistsTarget(BB.MYNEWKEY, mapgoal.GetSerialNum()))
{
}
```

---

## RemoveByPoster

Removes all records of a given type by the given owner id.

**Parameters:** (owner id #) - If key type is left off, removes ALL records

**Parameters:** (owner id #, blackboard key type to remove)

**Returns:** number of records removed

**Example:**

```
// remove just my record type from this owner
numremoved1 = Blackboard.RemoveByPoster(b.GetGameId(), BB.MYNEWKEY);
// remove all records of all types for this owner
numremoved2 = Blackboard.RemoveByPoster(b.GetGameId());
```

---

## RemoveByTarget

Removes all records of a given type by the given target id.

**Parameters:** (target id #) - If key type is left off, removes ALL records

**Parameters:** (target id #, blackboard key type to remove)

**Returns:** number of records removed

**Example:**

```
// remove just my record type from this owner
numremoved1 = Blackboard.RemoveByTarget(b.GetGameId(), BB.MYNEWKEY);
// remove all records of all types for this owner
numremoved2 = Blackboard.RemoveByTarget(b.GetGameId());
```

---

## PrintBlackboard

Prints the global blackboard to the in game console.

**Parameters:** () - if key left off, prints records of ALL types.

**Parameters:** (blackboard key) - only print records of this type.

**Returns:** none

**Example:**

```
Blackboard.PrintBlackboard();
Blackboard.PrintBlackboard(BB.MYNEWKEY);
```

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Blackboard>"

This page has been accessed 137 times. This page was last modified 19:56, 26 May 2007.

# MapGoal

From Omni-bot Wiki

## Contents

- 1 Map Goal Functions
  - 1.1 AddRoute
  - 1.2 AddUsePoint
  - 1.3 GetBounds
  - 1.4 GetEntity
  - 1.5 GetFacing
  - 1.6 GetGoalState
  - 1.7 GetLocalBounds
  - 1.8 GetMatrix
  - 1.9 GetName
  - 1.10 GetNumUsePoint
  - 1.11 GetOwner
  - 1.12 GetPosition
  - 1.13 GetRadius
  - 1.14 GetTagName
  - 1.15 GetTypeName
  - 1.16 GetUsePoint
  - 1.17 IsAvailable
  - 1.18 SetAvailable
  - 1.19 SetEnableDraw
  - 1.20 SetBounds
  - 1.21 SetFacing
  - 1.22 SetMatrix
  - 1.23 SetPosition
  - 1.24 SetRadius
  - 1.25 SetRemoveFlag
  - 1.26 MaxUsers\_InProgress
  - 1.27 MaxUsers\_InUse

## Map Goal Functions

### AddRoute

**See:** Omni-bot Routing

## AddUsePoint

Adds a use point to the goal. Not yet used.

**Parameters:** (Vector3)

**Returns:** none

**Example:**

```
mg.AddUsePoint(Vector3(10,10,10));
```

---

## GetBounds

Gets the AABB bounds of the map goal.

**Parameters:** none

**Returns:** AABB

**Example:**

```
bounds = mg.GetBounds();
```

---

## GetEntity

Gets the entity of the map goal.

**Parameters:** none

**Returns:** GameEntity

**Example:**

```
ent = mg.GetEntity();
```

---

## GetFacing

Gets the facing of the map goal.

**Parameters:** none

**Returns:** Vector3 facing

**Example:**

```
face = mg.GetFacing();
```

---

## GetGoalState

Gets the goal state of the map goal.

**Parameters:** none

**Returns:** goal state id

**Example:**

```
state = mg.GetGoalState();
```

---

## GetLocalBounds

Gets the local space bounds of the map goal.

**Parameters:** none

**Returns:** AABB bounds

**Example:**

```
localbounds = mg.GetLocalBounds();
```



---

## GetMatrix

Gets the Matrix3 transform of the map goal.

**Parameters:** none

**Returns:** Matrix3 transform

**Example:**

```
m = mg.GetMatrix();
```

---

## GetName

Gets the name of the map goal.

**Parameters:** none

**Returns:** name of mapgoal

**Example:**

```
name = mg.GetName();
```

---

## GetNumUsePoint

Gets the number of use points the goal has.

**Parameters:** none

**Returns:** # use points

**Example:**

```
n = mg.GetNumUsePoint();
```

---

## GetOwner

Gets the owner of the map goal. Typically someone that is carrying it.

**Parameters:** none

**Returns:** Owner GamelId, or null if no owner

**Example:**

```
owner = mg.GetOwner();
```

---

## GetPosition

Gets the position of the map goal.

**Parameters:** none

**Returns:** Vector3 position

**Example:**

```
pos = mg.GetPosition();
```

---

## GetRadius

Gets the radius of the map goal.

**Parameters:** none

**Returns:** float radius

**Example:**

```
pos = mg.GetRadius();
```

---

## GetTagName

Gets the tag name of the map goal. This is usually a mash of the entity name and goal type.

**Parameters:** none

**Returns:** tag name

**Example:**

```
tag = mg.GetTagName();
```

---

## GetTypeNames

Gets the type name of the map goal.

**Parameters:** none

**Returns:** type name

**Example:**

```
type = mg.GetTypeNames();
```

---

## GetUsePoint

Gets a use point, by index. If no index provided, returns a random one.

**Parameters:** (index<optional>)

**Returns:** Vector3 use point

**Example:**

```
pt = mg.GetUsePoint( );  
// OR  
pt = mg.GetUsePoint( 2 );
```

---

## IsAvailable

Checks if the map goal is currently available to a certain team.

**Parameters:** (team id)

**Returns:** true if goal is available for team, false if not

**Example:**

```
if ( mg.IsAvailable( TEAM.RED ) )  
{  
}
```

---

## SetAvailable

Sets the map goal available for a certain team.

**Parameters:** (team id, true/false)

**Returns:** none

**Example:**

```
mg.SetAvailable( TEAM.RED, false );
```

---

## SetEnableDraw

Enables debug rendering for this map goal.

**Parameters:** (true/false)

**Returns:** none

**Example:**

```
mg.SetEnableDraw(true);
```

---

## SetBounds

Sets the bounds for a map goal. This is the local space bounds. It will automatically be transformed to the world position of the map goal.

**Parameters:** (AABB)

**Returns:** none

**Example:**

```
AABB aabb;  
// initialize it somehow  
mg.SetBounds(aabb);
```

---

## SetFacing

Sets the facing for a map goal.

**Parameters:** (Vector3)

**Returns:** none

**Example:**

```
v = Vector3(1,0,0);  
// initialize it somehow  
mg.SetFacing(v);
```

---

## SetMatrix

Sets the matrix for a map goal.

**Parameters:** (Matrix3)

**Returns:** none

**Example:**

```
m = Matrix3();  
// initialize it somehow  
mg.SetMatrix(m);
```

---

## SetPosition

Sets the position for a map goal.

**Parameters:** (Vector3)

**Returns:** none

**Example:**

```
mg.SetPosition(Vector3(10,10,10));
```

---

## SetRadius

Sets the radius for a map goal.

**Parameters:** (radius)

**Returns:** none

**Example:**

```
mg.SetRadius(65);
```

---

## SetRemoveFlag

Marks the goal for removal.

**Parameters:** (true/false)

**Returns:** none

**Example:**

```
mg.SetRemoveFlag( true );
```

---

## MaxUsers\_InProgress

## MaxUsers\_InUse

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=MapGoal>"

This page has been accessed 185 times. This page was last modified 04:40, 28 May 2007.

# Matrix3

From Omni-bot Wiki

## Contents

- 1 Matrix3 Functions
  - 1.1 Inverse
  - 1.2 InverseTransformVector
  - 1.3 TransformVector

## Matrix3 Functions

### Inverse

### InverseTransformVector

### TransformVector

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Matrix3>"

This page has been accessed 128 times. This page was last modified 22:50, 19 May 2007.



# TargetInfo

From Omni-bot Wiki

## Contents

- 1 TargetInfo Properties
- 2 Distance
- 3 Position
- 4 Facing
- 5 Velocity
- 6 Class
- 7 TargetInfo Functions
  - 7.1 IsA

## TargetInfo Properties

All properties are **read only**.

### Distance

The bots distance to this target.

### Position

The targets world position Vector3.

### Facing

The targets facing Vector3.

### Velocity

The targets velocity Vector3.

### Class

The class id for the target.

## TargetInfo Functions

### IsA

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=TargetInfo>"

This page has been accessed 166 times. This page was last modified 23:04, 25 May 2007.

# Timer

From Omni-bot Wiki

## Timer Functions

**GetElapsedTime**

**Reset**

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Timer>"

This page has been accessed 121 times. This page was last modified 22:52, 19 May 2007.

# TriggerInfo

From Omni-bot Wiki

## Contents

- 1 TriggerInfo Properties
- 2 Name
- 3 Action
- 4 Activator
- 5 Entity

## TriggerInfo Properties

All properties are **read only**.

### Name

The name of the trigger. Also known as tagname.

### Action

The action string for the trigger. Normally indicates a verb or *action* the trigger resulted from.

### Activator

The entity activator of the trigger, whenever relevant.

### Entity

The entity representation.

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=TriggerInfo>"

This page has been accessed 143 times. This page was last modified 05:18, 20 May 2007.

# Vector3

From Omni-bot Wiki

## Contents

- 1 Vector3 Functions
  - 1.1 Cross
  - 1.2 Dot
  - 1.3 IsZero
  - 1.4 Length
  - 1.5 LengthSq
  - 1.6 MidPoint
  - 1.7 Normalize
  - 1.8 ProjectOntoVector
  - 1.9 Random
  - 1.10 Reflect
  - 1.11 Truncate
  - 1.12 UnitCross

## Vector3 Functions

Vector3 is a basic math type that is made up of 3 real numbers typically used to represent positions, velocities, and directions. As of 0.65, Vector3 objects in scripting have gotten a significant optimization to eliminate their previous need to allocate memory that would then need garbage collected. As a result, scripts that use them heavily, especially in mathematical operations, should not trigger garbage collection cycles, and should also run a bit faster.

**Cross**

**Dot**

**IsZero**

**Length**

**LengthSq**

**MidPoint**

**Normalize**

**ProjectOntoVector**

**Random**

**Reflect**

**Truncate**

**UnitCross**

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Vector3>"

This page has been accessed 122 times. This page was last modified 22:52, 19 May 2007.

# Weapon

From Omni-bot Wiki

## Contents

- 1 Weapon
  - 1.1 Properties
    - 1.1.1 Name
    - 1.1.2 WeaponId
    - 1.1.3 MinUseTime
    - 1.1.4 PrimaryFire
    - 1.1.5 SecondaryFire
- 2 Fire Mode
  - 2.1 Properties
    - 2.1.1 RequiresAmmo
    - 2.1.2 WaterProof
    - 2.1.3 SplashDamage
    - 2.1.4 HasClip
    - 2.1.5 HasZoom
    - 2.1.6 Stealth
    - 2.1.7 InheritsVelocity
    - 2.1.8 NeedsInRange
    - 2.1.9 ManageHeat
    - 2.1.10 IgnoreReload
    - 2.1.11 RequiresTargetOutside
    - 2.1.12 RequiresShooterOutside
    - 2.1.13 WeaponType
    - 2.1.14 MaxEquipMoveMode
    - 2.1.15 Offhand
    - 2.1.16 ManualDetonation
    - 2.1.17 MustBeOnGround
    - 2.1.18 FireOnRelease
    - 2.1.19 UseMortarTrajectory
    - 2.1.20 SniperWeapon
    - 2.1.21 MaxAimError
    - 2.1.22 AimOffset
    - 2.1.23 PitchOffset
    - 2.1.24 ShootButton
    - 2.1.25 ZoomButton
    - 2.1.26 LowAmmoThreshold
    - 2.1.27 LowAmmoPriority
    - 2.1.28 FuseTime
    - 2.1.29 AmmoType
    - 2.1.30 ProjectileSpeed
    - 2.1.31 MinRange
    - 2.1.32 MaxRange
    - 2.1.33 MinChargeTime
    - 2.1.34 MaxChargeTime
    - 2.1.35 DelayAfterFiring
    - 2.1.36 ProjectileGravity

- 2.1.37 DefaultDesirability
- 2.1.38 MinAimAdjustmentTime
- 2.1.39 MaxAimAdjustmentTime
- 2.1.40 CalculateDefaultDesirability
- 2.1.41 CalculateDesirability
- 2.1.42 CalculateAimPoint
- 2.1.43 PreShoot
- 2.2 Functions
  - 2.2.1 SetBurstRange
  - 2.2.2 SetDesirabilityRange
  - 2.2.3 SetTargetBias

# Weapon

## Properties

### Name

Name of the weapon.

### WeaponId

The weapon id of the weapon. Should correspond to an entry in the global WEAPON table.

### MinUseTime

The minimum time the bot should maintain the weapon once selected.

### PrimaryFire

Returns a reference to the primary Fire Mode

### SecondaryFire

Returns a reference to the secondary Fire Mode

## Fire Mode

## Properties

### RequiresAmmo

This weapon requires ammo to operate. The type of ammo is determined by the Properties|AmmoType.

### WaterProof

The weapon can be fired underwater. If false, the weapon will not be considered when the user is



underwater.

## **SplashDamage**

The weapon produces splash damage, and should be disfavored when there are allies in the target area.

## **HasClip**

The weapon has a clip. Helps determine how ammo checking is performed.

## **HasZoom**

The weapon has a zoom mode. Not yet used.

## **Stealth**

The weapon should be favored when stealth is desired. Not yet used.

## **InheritsVelocity**

The weapon projectile inherits the velocity of the player when fired. This is used for a bit more accuracy in the projectile leading calculations.

## **NeedsInRange**

The bot must be within the MinRange and MaxRange to use, even if that means the bot should chase the target. Typically used for melee weapons.

## **ManageHeat**

The weapon can overheat, so the bot should oscillate the firing to attempt to keep from overheating.

## **IgnoreReload**

Ignore this weapon if it needs to reload. Normally bot would switch to weapons to reload them if they have no target. This option disables that.

## **RequiresTargetOutside**

This weapon requires that the target is outdoors.

## **RequiresShooterOutside**

This weapon requires that the shooter is outdoors.

## **WeaponType**

The type of weapon this fire mode is.

- melee - Melee attacks are close range and treated differently.
- instant - Instant hit means no leading or projectile velocity is necessary.
- projectile - Bot will lead targets based on projectile velocity.
- grenade - Bot will take fuse time into account.

## **MaxEquipMoveMode**

The fastest the bot should move when this weapon is equipped.

- run
- walk
- still

## **Offhand**

Weapon can be used simultaneously with another weapon, such as offhand grenades.

## **ManualDetonation**

The weapon projectile must be manually detonated. Not used yet.

## **MustBeOnGround**

The bot must be standing on the ground to use this weapon(not airborne).

## **FireOnRelease**

The weapon is fired when the fire button is released, as opposed to firing when the fire button is pressed.

## **UseMortarTrajectory**

The weapon should use the mortar trajectory when calculating trajectory.

## **SniperWeapon**

The weapon is a sniper weapon, and should be considered for use for goals that require a sniper weapon.

## **MaxAimError**

The maximum aim error to use for the fire mode. Aim Error is a  $\text{Vector2}(x,y)$ , where x is the horizontal aim error and y is the vertical aim error.

## **AimOffset**

The offset where to aim at players. AimOffset is a  $\text{Vector3}(x,y,z)$  that represents a world offset to aim the weapon.

## **PitchOffset**

Normally a projectile comes out along the aim vector of the bot, but some weapons can come out at an angle. PitchOffset is the pitch offset(in degrees) the projectile shoots at.

## **ShootButton**

The button id of the button used to fire the weapon. Defaults to BTN.ATTACK1

## **ZoomButton**

The button id of the button used to zoom the weapon. Defaults to BTN.AIM

## **LowAmmoThreshold**

Used for ammo desirability calculations. This is the threshold the weapon will be considered low on ammo.

## **LowAmmoPriority**

When the ammo is below the threshold above, this is the priority used for the desirability of the ammo.

## **FuseTime**

The countdown timer the projectile takes before it explodes/activates.

## **AmmoType**

The ammo id the weapon uses. Depending on the game, this may be a duplicate of the weapon id or it may be an actual ammo id, from the global AMMO table.

## **ProjectileSpeed**

The speed the projectile flies at. Only relevant to projectile fire modes.

## **MinRange**

The minimum range the bot should consider using this fire mode.

## **MaxRange**

The maximum range the bot should consider using this fire mode.

## **MinChargeTime**

The minimum time the bot should charge this weapon before shooting. Normally used with FireOnRelease property. Used as a minimum range for a random charge time, along with MaxChargeTime.

## **MaxChargeTime**

The maximum time the bot should charge this weapon before shooting. Normally used with FireOnRelease property. Used as a maximum range for a random charge time, along with MinChargeTime.

## **DelayAfterFiring**

How long the bot should wait to choose this weapon again after firing.

## **ProjectileGravity**

The gravity multiplier used for the projectile. Used to calculate trajectory of the projectile. Projectiles that aren't effected by gravity will have a ProjectileGravity of 0. A projectile that has half gravity is a 0.5, while full gravity is 1.0

## **DefaultDesirability**

Default desirability is the desirability used for choosing a weapon when the bot has no target. The highest default desirability will be equipped when the bot has no target, after all weapons that need reloading is considered.

## **MinAimAdjustmentTime**

Minimum time between new calculations of aim error.

## **MaxAimAdjustmentTime**

Maximum time between new calculations of aim error.

## **CalculateDefaultDesirability**

Script function callback to calculate the default desirability. Should return a 0-1 desirability value.

## **CalculateDesirability**

Script function callback to calculate the desirability of the weapon. Should return a 0-1 desirability value. This callback will be passed the TargetInfo of the current target.

## **CalculateAimPoint**

Script function callback to calculate an aim point for the weapon. Should return a Vector3 world aim position. This callback will be passed the TargetInfo of the current target.

## **PreShoot**

Script function callback to that is called just before the weapon is fired. Not used yet.

## **Functions**

### **SetBurstRange**

This function can be called multiple times in order to set several ranges for burst firing. Whenever the target is within one of the ranges, the bot will use controlled burst fire as set up by this function.

**Parameters:** (minrange, maxrange, numrounds, mindelay, maxdelay)

**Returns:** none

**Example:**

```
// from 200-500 units away, set up a 3 shot burst rate, with a 1-2 second delay between each burst
w.PrimaryFire.SetBurstRange(200, 500, 3, 1, 2);
```

## SetDesirabilityRange

This function can be called multiple times to assign a desirability to use the weapon when the target is within a given range of the bot.

**Parameters:** (minrange, maxrange, desirability)

**Returns:** none

**Example:**

```
// between 200 and 500 range, we want the desirability to be 0.5
w.PrimaryFire.SetDesirabilityRange(200, 500, 0.5);
```

## SetTargetBias

Sets a bias multiplier for a target type. This bias will be multiplied with the desirability when calculating the desirability of a weapon versus a target.

**Parameters:** (targettype, bias)

**Returns:** none

**Example:**

```
// make this weapon 1.5 times more likely to be used against heavy vehicles
w.PrimaryFire.SetTargetBias(CLASS.VEHICLE_HVY, 1.5);
```

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Weapon>"

This page has been accessed 200 times. This page was last modified 06:30, 25 August 2008.

# Template:Map Script Utilities

From Omni-bot Wiki

## Contents

- 1 Utilities
  - 1.1 Util.AddInvVehicle
  - 1.2 Util.AliveCount
  - 1.3 Util.DisableGroup
  - 1.4 Util.DistanceView
  - 1.5 Util.EnableGroup
  - 1.6 Util.GetGroup
  - 1.7 Util.OnTriggerPosition
  - 1.8 Util.RemoveGoal
  - 1.9 Util.SetGoalOffset
  - 1.10 Util.SetGoalPosition
  - 1.11 Util.SetGroup
  - 1.12 Util.ShowGroup
  - 1.13 Util.SetMaxUsersInProgress
  - 1.14 Util.SetPositionGoal
  - 1.15 Util.ShowGoalInfo
  - 1.16 Util.ShowGoalName
  - 1.17 Util.ShowGoalOffset
  - 1.18 Util.AddUsePoint
  - 1.19 Util.AddUseWp

## Utilities

Utility functions are located in the ~/omni-bot/global\_scripts/utilities.gm file.

### Util.AddInvVehicle

syntax: `Util.AddInvVehicle( goalname )`  
usage: adds a **vehicle** to the Map.InvVehicle **table**  
used **for script goals in** cases where the **vehicle** is invulnerable, but shows as "**dead**"

### Util.AliveCount

syntax: `Util.AliveCount( team, class )`  
usage: returns the number of bots alive on a **team** with a given **class**

### Util.DisableGroup

syntax: `Util.DisableGroup(groupname, team);`  
example: `Util.DisableGroup( "somegroupname", TEAM.SOMETEAM );`

usage: used to disable a set of goals in the given group for a given team

## Util.DistanceView

```
syntax: Util.DistanceView(<optional off|0>);  
usage: /bot dist  
       /bot dist off  
       /bot dist 0
```

note: used for determining bot.MaxViewDistance settings. aim at a position as far as you can see and type /bot dist

## Util.EnableGroup

```
syntax: Util.EnableGroup(groupname, team);  
example: Util.EnableGroup( "somegroupname", TEAM.SOMETEAM );
```

usage: used to enable a set of goals in the given group for a given team

## Util.GetGroup

```
syntax: Util.GetGroup(groupname);  
example: Util.GetGroup("somegroupname");
```

usage: this function will return a table of goals belonging to the given group name

## Util.OnTriggerPosition

```
syntax: Util.OnTriggerPosition( goalname, wpname, tolerance, wpfunction )  
example: Util.OnTriggerPosition( Map.Mover_train1, "depotyard", 200.0, Map.tug_depotyard );
```

note: used for setting up positional triggers for movers.

## Util.RemoveGoal

```
syntax: Util.RemoveGoal( goalname );  
example: Util.RemoveGoal( "MOVER_truck" );
```

note: this command will remove the goal from the map goal table

## Util.SetGoalOffset

```
syntax: Util.SetGoalOffset( x, y, z, GoalName );  
example: Util.SetGoalOffset( 0, 0, 30, "BUILD_construct" )
```

usage: the x y and z parameters are added to the origin of the goal to move the location of where the bots will look for the goal.

## Util.SetGoalPosition

```
syntax: Util.SetGoalPosition( x, y, z, GoalName );  
example: Util.SetGoalPosition( 4534, 2168, -199, "BUILD_construct" )
```

usage: used to give the goal a new origin.

note: using /devmap to load the map and issuing the /viewpos command in console will give your origin in the map

## Util.SetGroup

syntax: `Util.SetGroup(goalname, groupname);`

example: `Util.SetGroup("somegoalname", "somegroupname");`

usage: this function will add a given goal to a given groupname

## Util.ShowGroup

syntax: `Util.ShowGroup(groupname);`

example: `Util.ShowGroup("somegroupname");`

usage: this function will list all goals in the given group in the console.

## Util.SetMaxUsersInProgress

syntax: `Util.SetMaxUsersInProgress( Users, GoalNames );`

example: `Util.SetMaxUsersInProgress( 15, "CHECKPOINT.*" );`

usage: used to set the maximum number of bots going for a particular goal(s).

## Util.SetPositionGoal

syntax: `Util.SetPositionGoal( goalname1, goalname2 );`

example: `Util.SetPositionGoal( Map.Build_tank_construct, Map.Mover_tank );`

usage: sets the origin of one goal to match the origin of another. useful for centering construct goals on movers

## Util.ShowGoalInfo

syntax: `Util.ShowGoalInfo(goalname);`

command: `/bot sgi 'goalname'`

usage: used to print a goals current information in console including the goals health, status of the DEAD flag,  
and the goals position

## Util.ShowGoalName

syntax: `Util.ShowGoalName(radius, showOffset);`

command: `/bot sgn <radius> <optional true>`

usage: issue the command /bot sgn to find the goalname of any goals within 100 units. Optionally expand the radius and  
find your current offset from the goal with /bot sgn 500 true

## Util.ShowGoalOffset



syntax: `Util.ShowGoalOffset(goalname);`  
command: `/bot sgo 'somegoalname'`

usage: used to determine a players current offset from a goal. the values given in console can be used for goals that  
require positional offsets

## Util.AddUsePoint

syntax: `Util.AddUsePoint(goalname, position);`  
example: `Util.AddUsePoint( "somegoalname", Vector3(123, 456, 789) );`

usage: used to add a Use Point for a goal at a given position.

## Util.AddUseWp

syntax: `Util.AddUseWp(goalname, waypointname);`  
example: `Util.AddUseWp( "somegoalname", "somewaypointname" );`

usage: used to add a Use Point for a goal at a given waypoints position.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Template:Map\\_Script\\_Uilities](http://omni-bot.com/wiki/index.php?title=Template:Map_Script_Uilities)"

This page has been accessed 67 times. This page was last modified 06:00, 25 August 2008.

# Category:Constants

From Omni-bot Wiki

## Pages in category "Constants"

There are 19 pages in this category.

<b>A</b>	<b>C cont.</b>	<b>P cont.</b>
<div><ul style="list-style-type: none"><li>• AMMO</li></ul></div>	<div><ul style="list-style-type: none"><li>• CLASS</li><li>• CONTENT</li></ul></div>	<div><ul style="list-style-type: none"><li>• PROFILE</li></ul></div>
<b>B</b>	<b>D</b>	<b>S</b>
<div><ul style="list-style-type: none"><li>• BB</li><li>• BONE</li><li>• BTN</li><li>• BUY</li></ul></div>	<div><ul style="list-style-type: none"><li>• DEBUG</li></ul></div>	<div><ul style="list-style-type: none"><li>• SKILL</li></ul></div>
<b>C</b>	<b>E</b>	<b>T</b>
<div><ul style="list-style-type: none"><li>• CAT</li></ul></div>	<div><ul style="list-style-type: none"><li>• ENTFLAG</li><li>• EVENT</li></ul></div>	<div><ul style="list-style-type: none"><li>• TEAM</li><li>• TRACE</li></ul></div>
	<b>I</b>	<b>V</b>
	<div><ul style="list-style-type: none"><li>• ITEM</li></ul></div>	<div><ul style="list-style-type: none"><li>• VOICE</li></ul></div>
	<b>P</b>	<b>W</b>
	<div><ul style="list-style-type: none"><li>• POWERUP</li></ul></div>	<div><ul style="list-style-type: none"><li>• WEAPON</li></ul></div>

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Category:Constants>"

This page has been accessed 200 times. This page was last modified 03:55, 28 April 2008.

# Category:ET Constants

From Omni-bot Wiki

## Pages in category "ET Constants"

There are 7 pages in this category.

### B

- Buttons Enemy Territory

### C cont.

- Classes Enemy Territory

### V

- Voice Macros Enemy Territory

### C

- Categories Enemy Territory

### E

- Entity Flags Enemy Territory
- Events Enemy Territory

### W

- Weapons Enemy Territory

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Category:ET\\_Constants](http://omni-bot.com/wiki/index.php?title=Category:ET_Constants)"

This page has been accessed 192 times. This page was last modified 20:49, 15 September 2007.

# Category:ET ScriptGoals

From Omni-bot Wiki

## Pages in category "ET ScriptGoals"

There are 4 pages in this category.

### E

- Escort vehicle

### M

- Mount vehicle

### R

- Ride vehicle

### U

- Useswitch

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Category:ET\\_ScriptGoals](http://omni-bot.com/wiki/index.php?title=Category:ET_ScriptGoals)"

This page has been accessed 172 times. This page was last modified 20:50, 15 September 2007.

# ET:goal airstrike

From Omni-bot Wiki

ET Script Goals	<b>Goal Airstrike</b>
-----------------	-----------------------

This goal provides functionality for Fieldops bots to throw an airstrike from a waypoint. It relies on a populated table of waypoint names for the bots team. It requires some script and waypoint configuration.

## Setup

After selecting and naming waypoints for use with the goal, the teams AS table needs to be populated for the bots to use them. The tables must be in the Map table of the mapscript.gm:

```
global Map =
{
    //tables to hold current AS goals
    AxisASTable = {},
    AlliesASTable = {},
}
```

In this case, the tables have been initialized but are empty. The tables can be populated in triggers to ensure that they are active at the correct time:

```
global Map =
{
    //tables to hold current AS goals
    AxisASTable = {},
    AlliesASTable = {},

    sometrigger = function( trigger )
    {
        //activate some airstrike waypoints for the axis team
        Map.AxisASTable = { "as_waypoint1", "as_waypoint2", },
    }

    anothertrigger = function( trigger )
    {
```

```

        //deactivate some airstrike waypoints for the axis team
        Map.AxisASTable = {},
    }
};

```

In cases where you may want different airstrike goals for different phases, it may be a good idea to set up a few tables to be copied to the main AS table(s):

```

global Map =
{
    //tables to hold current AS goals
    AxisASTable = {},
    AlliesASTable = {},

    //different airstrike tables to be copied
    axis_aswall = { "as_waypoint1", "as_waypoint2", },
    axis_asgate = { "as_waypoint3", "as_waypoint4", },

    sometrigger = function( trigger )
    {
        //activate the wall airstrike waypoints for the axis team
        Map.AxisASTable = Map.axis_aswall;
    }

    walltrigger = function( trigger )
    {
        //activate the gate airstrike waypoints for the axis team
        Map.AxisASTable = Map.axis_asgate;
    }

    anothertrigger = function( trigger )
    {
        //deactivate some airstrike waypoints for the axis team
        Map.AxisASTable = {},
    }
};

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_airstrike](http://omni-bot.com/wiki/index.php?title=ET:goal_airstrike)"

This page has been accessed 42 times. This page was last modified 02:29, 25 August 2008.

# ET:goal askforammo

From Omni-bot Wiki

ET Script Goals	<b>Goal Ask For Ammo</b>
-----------------	--------------------------

This script goal provides functionality for bots to look for ammo packs when needed or call for a Fieldop if no packs are found. If no Fieldops are alive on the team, the bot(s) will not call for ammo. No additional scripting setup is required

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_askforammo](http://omni-bot.com/wiki/index.php?title=ET:goal_askforammo)"

This page has been accessed 42 times. This page was last modified 02:37, 25 August 2008.

# ET:goal askforhealth

From Omni-bot Wiki

ET Script Goals	<b>Goal Ask For Health</b>
-----------------	----------------------------

This script goal provides functionality for bots to look for medic packs when needed or call for a Medic if no packs are found. If no Medics are alive on the team, the bot(s) will not call for ammo. No additional scripting setup is required

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_askforhealth](http://omni-bot.com/wiki/index.php?title=ET:goal_askforhealth)"

This page has been accessed 41 times. This page was last modified 02:40, 25 August 2008.



# ET:goal combatmovement

From Omni-bot Wiki

ET Script Goals	<b>Goal Combat Movement</b>
-----------------	-----------------------------

This script goal provides for some strafing / crouching in combat. There are some configuration options in the script for different levels of difficulty. The level can be set at this line at the top of the script:

```
this.Difficulty = 4; // 0 - off, 1 - easy, 2 - medium, 3 - hard, 4 - random
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_combatmovement](http://omni-bot.com/wiki/index.php?title=ET:goal_combatmovement)"

This page has been accessed 45 times. This page was last modified 02:44, 25 August 2008.

# ET:goal deliversupplies

From Omni-bot Wiki

ET Script Goals	<b>Goal Deliver Supplies</b>
-----------------	------------------------------

This script goal provides functionality for Medics and Fieldops to respond to respective team voice requests "NeedAmmo" and "NeedMedic". No additional scripting or configuration is required for the goal to run.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_deliversupplies](http://omni-bot.com/wiki/index.php?title=ET:goal_deliversupplies)"

This page has been accessed 36 times. This page was last modified 02:47, 25 August 2008.

# ET:goal dispenseammo

From Omni-bot Wiki

ET Script Goals	<b>Goal Dispense Ammo</b>
-----------------	---------------------------

This script goal provides functionality for Fieldops to pass out ammo when they spawn. The time the bots will wait before dispensing the ammo is defaulted to 5, but can be adjusted in the mapscript.gm file. The parameter must be in the map table:

```
global Map =
{
    DispenseAmmoTime = 10, //wait 10 seconds after spawn to dispense ammo
};
```

The goal can also be completely disabled in the mapscript.gm:

```
global Map =
{
    DontDispenseAmmo = true, //disable fieldops passing out ammo in spawn
};
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_dispenseammo](http://omni-bot.com/wiki/index.php?title=ET:goal_dispenseammo)"

This page has been accessed 33 times. This page was last modified 02:53, 25 August 2008.

# ET:goal dispensehealth

From Omni-bot Wiki

ET Script Goals	<b>Goal Dispense Health</b>
-----------------	-----------------------------

This script goal provides functionality for Medics to pass out health when they spawn. The time the bots will wait before dispensing the health is defaulted to 5, but can be adjusted in the mapscript.gm file. The parameter must be in the map table:

```
global Map =
{
    DispenseHealthTime = 10, //wait 10 seconds after spawn to dispense ammo
};
```

The goal can also be completely disabled in the mapscript.gm:

```
global Map =
{
    DontDispenseHealth = true, //disable fieldops passing out ammo in spawn
};
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_dispensehealth](http://omni-bot.com/wiki/index.php?title=ET:goal_dispensehealth)"

This page has been accessed 33 times. This page was last modified 02:54, 25 August 2008.

# ET:goal escortvehicle

From Omni-bot Wiki

ET Script Goals	Goal Escort Vehicle
-----------------	---------------------

This script goal provides functionality for bots to escort vehicles. It requires some map script setup. The goal requires a Map.EscortVehicle table to be defined in the mapscript.gm:

```
global Map =
{
    EscortVehicle =
    {
        Truck =
        {
            Enabled = true, // enable the goal
            Priority = 0.65, // activate the goal with a 0.65 priority
            LimitClass = Util.AllClasses, // all classes can use this goal for this vehicle
            LimitDistance = 256, //if the mover has moved over 256 units since getting the
goal, abort
            EscortVehicleGoalName = "MOVER_truck", //the exact goal name of this vehicle goal
            LimitBots = 2, // only 2 bots can use this goal
            Offset = // position(s) around the vehicle to escort from
            {
                Vector3(0,-40,0),
            },
        },
    },
};
```

The script goal will default most of the variables above to certain values. Technically all that is required is that the vehicle goal name be given:

```
global Map =
{
    EscortVehicle =
    {
        Truck =
        {
            EscortVehicleGoalName = "MOVER_truck", //the exact goal name of this vehicle goal
        },
    },
};
```

If the mover goal is not showing up in the goal list it is because it is using a non standard name. Most mover goals are removed from the goal list because typically maps have several movers that aren't needed by omnibots (moving fans, etc). To find the goal name, add the ShowMovers flag to the Map table and list all mover goals with /bot show\_goals MOVER.\* in console:

```
global Map =
```

```
{
    ShowMovers = true, // keep all movers in the goal table
};
```

Once you have found the goal name for the mover, be sure to set the ShowMovers flag to false and set the mapsript up so the vehicle you want is retained in the goal table:

```
global Map =
{
    ShowMovers = false, // don't keep all movers in the goal table

    Movers =
    {
        //retain this mover for the escort goal
        "MOVER_somemover",
    },
};
```

In some cases with invulnerable vehicles, the bots will not escort because they believe that the vehicle is "Dead". By default, every attempt possible is made to determine if the vehicle is invulnerable and can be escorted, but some map makers set the entities up in an unusual manner. If you find that pathing and availability is correct for an escortable vehicle that is invulnerable, this is most likely the reason. This can be fixed by manually adding the vehicle to the invulnerable table with a function call in OnMapLoad:

```
global OnMapLoad = function()
{
    Util.AddInvVehicle("MOVER_somemover");
};
```

Triggers can be used to set availability for the mover:

```
global Map =
{
    EscortVehicle =
    {
        Truck =
        {
            Enabled = true, // enable the goal
            Priority = 0.65, // activate the goal with a 0.65 priority
            LimitClass = Util.AllClasses, // all classes can use this goal for this vehicle
            LimitDistance = 256, //if the mover has moved over 256 units since getting the
goal, abort
            EscortVehicleGoalName = "MOVER_truck", //the exact goal name of this vehicle goal
            LimitBots = 2, // only 2 bots can use this goal
            Offset = // position(s) around the vehicle to escort from
            {
                Vector3(0,-40,0),
            },
        },
    },
};
```

```
},

sometrigger = function( trigger )
{
    //disable the escort goal
    Map.EscortVehicle.Truck.Enabled = false;
},

anothertrigger = function( trigger )
{
    //enable the escort goal
    Map.EscortVehicle.Truck.Enabled = true;
},

};
```

Tip: finding offsets for the vehicle is easily done by standing in a spot you would like them to escort from and issuing the Show Goal Name command:

```
/bot sgn 300 true
```

This will find all goals within 300 units of your current position and print the offsets in the console.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_escortvehicle](http://omni-bot.com/wiki/index.php?title=ET:goal_escortvehicle)"

This page has been accessed 46 times. This page was last modified 03:41, 25 August 2008.

# ET:goal grenadetarget

From Omni-bot Wiki

## ET Script Goals    Goal Grenade Target

This script goal provides functionality for the bots to use grenades to destroy goals that can be grenaded; like barriers or mg42's. It requires a Map.Target table to be defined in the mapscript.gm:

```
global Map =
{
    Target =
    {
        Barricade1 =
        {
            Priority = 0.67,
            Enabled = true,
            GrenadeTargetGoalName = "EXPLODE_barrier1",
            LimitBots = 1,
            LimitTeam = (1<<TEAM.ALLIES),
            LimitClass = (1<<CLASS.MEDIC) | (1<<CLASS.LIEUTENANT),
            Offset =
            {
                Vector3(0, -400, 0),
            },
        },
    },
};
```

The goal will automatically be skipped once the target is destroyed. Availability can also be set in triggers using the Enabled flag:

```
global Map =
{
    sometriggers = function( trigger )
    {
        //disable the grenade target goal for this barrier
        Map.Target.Barricade1.Enabled = false;
    },
};
```



```
anothertrigger = function( trigger )  
{  
    //enable the grenade target goal for this barrier  
    Map.Target.Barrier1.Enabled = true;  
},  
};
```

The bots will go to the offset position defined in the table and throw a grenade at it. The best way to find the offset is to stand where you want the bots to throw it from and type /bot sgn 300 true in the console. The console will print all goal names found within 300 units of your current position and list the offsets in the console.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_grenadetarget](http://omni-bot.com/wiki/index.php?title=ET:goal_grenadetarget)"

This page has been accessed 37 times. This page was last modified 03:58, 25 August 2008.

# ET:goal mountvehicle

From Omni-bot Wiki

ET Script Goals	Goal Mount Vehicle
-----------------	--------------------

This script goal provides functionality for the bots to mount vehicles with mountable mg42's. By default the goal should run automatically if there are mountable vehicles in the map, but some configuration options can be given in the Map.MountVehicle table:

```
global Map =
{
    MountVehicle =
    {
        Enabled = true, // enable the goal
        Priority = 0.65, // activate the goal with a 0.65 priority
        LimitClass = Util.AllClasses, // all classes can use this goal
        LimitTeams = Util.AllTeams, // all teams can use this goal
        LimitBots = 2, // only 2 bots can use this goal
    },
};
```

If the mover goal is not showing up in the goal list it is because it is using a non standard name. Most mover goals are removed from the goal list because typically maps have several movers that aren't needed by omnibots (moving fans, etc). To find the goal name, add the ShowMovers flag to the Map table and list all mover goals with /bot show\_goals MOVER.\* in console:

```
global Map =
{
    ShowMovers = true, // keep all movers in the goal table
};
```

Once you have found the goal name for the mover, be sure to set the ShowMovers flag to false and set the mapscript up so the vehicle you want is retained in the goal table:

```
global Map =
{
    ShowMovers = false, // don't keep all movers in the goal table

    Movers =
    {
        //retain this mover for the mount goal
        "MOVER_somemover",
    },
};
```

In some cases with invulnerable vehicles, the bots will not mount because they believe that the vehicle is "Dead". By default, every attempt possible is made to determine if the vehicle is invulnerable and can be mounted, but some map makers set the entities up in an unusual manner. If you find that pathing and

availability is correct for a mountable vehicle that is invulnerable, this is most likely the reason. This can be fixed by manually adding the vehicle to the invulnerable table with a function call in OnMapLoad:

```
global OnMapLoad = function()  
{  
    Util.AddInvVehicle("MOVER_somemover");  
};
```

Triggers can be used to set availability for the mover:

```
global Map =  
{  
    MountVehicle =  
    {  
        Enabled = true, // enable the goal  
        Priority = 0.65, // activate the goal with a 0.65 priority  
        LimitClass = Util.AllClasses, // all classes can use this goal  
        LimitTeams = Util.AllTeams, // all teams can use this goal  
        LimitDistance = 256, //if the mover has moved over 256 units since getting the goal,  
abort  
        LimitBots = 2, // only 2 bots can use this goal  
    },  
  
    sometrigger = function( trigger )  
    {  
        //disable the mount goal  
        Map.MountVehicle.Enabled = false;  
    },  
  
    anothertrigger = function( trigger )  
    {  
        //enable the mount goal  
        Map.MountVehicle.Enabled = true;  
    },  
};
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_mountvehicle](http://omni-bot.com/wiki/index.php?title=ET:goal_mountvehicle)"

This page has been accessed 39 times. This page was last modified 04:11, 25 August 2008.

# ET:goal ridevehicle

From Omni-bot Wiki

## ET Script Goals    Goal Ride Vehicle

This script goal provides functionality for the bots to ride vehicles like the tugs in railgun. It requires a Map.RideVehicle table to be set up in the mapscript.gm:

```
global Map =
{
    RideVehicle =
    {
        Tug1 =
        {
            Enabled = true,
            Priority = 0.65,
            LimitTeam = (1<<TEAM.AXIS),
            LimitClass = (1<<CLASS.SOLDIER) | (1<<CLASS.COVERTOPS),
            RideVehicleGoalName = "MOVER_train1",
            SeatRadius = 90,
            Seat =
            {
                {
                    BoardPosition = Vector3(-128,64,0),
                    RidePosition = Vector3(-25,64,26),
                    FacePosition = Vector3(-128,64,55),
                },
                {
                    BoardPosition = Vector3(128,64,0),
                    RidePosition = Vector3(25,64,26),
                    FacePosition = Vector3(128,64,55),
                },
            },
        },
    },
};
```

If the mover goal is not showing up in the goal list it is because it is using a non standard name. Most mover goals are removed from the goal list because typically maps have several movers that aren't needed by omnibots (moving fans, etc). To find the goal name, add the ShowMovers flag to the Map table and list all mover goals with /bot show\_goals MOVER.\* in

console:

```
global Map =
{
    ShowMovers = true, // keep all movers in the goal table
};
```

Once you have found the goal name for the mover, be sure to set the ShowMovers flag to false and set the mapscript up so the vehicle you want is retained in the goal table:

```
global Map =
{
    ShowMovers = false, // don't keep all movers in the goal table

    Movers =
    {
        //retain this mover for the escort goal
        "MOVER_somemover",
    },
};
```

Triggers can be used to set availability for the mover:

```
global Map =
{

    sometrigger = function( trigger )
    {
        //disable the ride vehicle goal
        Map.RideVehicle.Tug1.Enabled = false;
    },

    anothertrigger = function( trigger )
    {
        //enable the ride vehicle goal, but limit it to allies
        Map.RideVehicle.Tug1.Enabled = true;
        Map.RideVehicle.Tug1.LimitTeam = (1<<TEAM.ALLIES);
    },
};
```

The seat radius is an important variable to customize. It's function is to provide a radius in

which the bot considers itself to be able to ride the mover. If the bot for any reason moves outside of the seat radius, the goal will finish so the bot can re-board the vehicle.

To find the BoardPosition, RidePosition, and FacePosition values, stand in each respective position and issue the /bot sgn 300 true command in the console. This will find all goals within 300 units of your current position and list the offsets in console. Some further tweaking may be required in the case of FacePosition.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_ridevehicle](http://omni-bot.com/wiki/index.php?title=ET:goal_ridevehicle)"

This page has been accessed 38 times. This page was last modified 04:31, 25 August 2008.

# ET:goal supplyself

From Omni-bot Wiki

ET Script Goals	<b>Goal Supply Self</b>
-----------------	-------------------------

This script goal provides functionality for Fieldops and Medics to supply themselves with packs if they need it. It will run by default and does not require any additional configuration.

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_supplyself](http://omni-bot.com/wiki/index.php?title=ET:goal_supplyself)"

This page has been accessed 35 times. This page was last modified 04:33, 25 August 2008.

# ET:goal useswitch

From Omni-bot Wiki

ET Script Goals	Goal Use Switch
-----------------	-----------------

This script goal provides functionality for bots to use buttons and switches. It requires a Map.Switches table to be defined in mapscript.gm:

```
global Map =
{
    Switches =
    {
        Gate1 =
        {
            Enabled = true,
            Priority = 0.65,
            WaypointName = "gate1",
            LimitTeam = (1<<TEAM.ALLIES),
            LimitBots = 2,
            LimitClass = (1<<CLASS.FIELDOPS) | (1<<CLASS.SOLDIER),
            LimitDistance = 4000, //optional, only bots within 4000 units will use the switch
            //optional exit conditions
            ExitConditions =
            {
                immediate = function()
                {
                    //immediately disable it
                    Map.Switches.Gate1.LimitTeam = 0;
                    return true;
                },
            },
        },
    },
};
```

The waypoint closest to the switch should be used for this goal. Use /bot waypoint\_setname <yourname> to set the name. The facing of the waypoint is also critical for the goal to execute correctly. Stand at the waypoint, face the switch and issue the /bot waypoint\_setfacing command.

The usage of triggers to set availability for switches is a key component to the execution of this goal since once a door or gate is opening, you typically will want to disable it immediatley. Use /bot debugtriggers and look in console once you use the switch to find the correct triggers to use:

```
global Map =
{
    gate_trigger = function( trigger )
    {
        // convert the Action string into a vector3 value
```



```
vel = ToVector(trigger.Action);
```

```
    //in the debug triggers output, the action strings third value changes, so use vel.z  
to evaluate
```

```
    // if the first value changes, use vel.x and if the second value changes, use vel.y
```

```
    if ( vel.z < 0 )
```

```
    {
```

```
        //disable the gate switch for both teams
```

```
        Map.Switches.Gate1.LimitTeam = 0;
```

```
    }
```

```
    else if ( vel.z > 0 )
```

```
    {
```

```
        //enable the gate switch for the allies
```

```
        Map.Switches.Gate1.LimitTeam = (1<<TEAM.ALLIES);
```

```
    }
```

```
    },
```

```
};
```

```
global OnMapLoad =
```

```
{
```

```
    OnTrigger( "base_lever1_goto", Map.gate_lever );
```

```
};
```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET:goal\\_useswitch](http://omni-bot.com/wiki/index.php?title=ET:goal_useswitch)"

This page has been accessed 38 times. This page was last modified 05:09, 25 August 2008.

# Map template

From Omni-bot Wiki

## Map-Template

### TBD

- Syntax check
- Adding priorities to routes to template

### Template

```
// Omni-Bot Version 0.XX
// Don't forget to disable Map.scriptDebug on release

// Our Map
global Map =
{
    name = GetMapName(),
    scriptDebug = true,

    routingEnabled = false,

    // TBD
    currentPhase = "",
    currentGoal = "",
    currentGoalGroup = "",

    minDefCampTime = 10,
    maxDefCampTime = 60,
    minAtCampTime = 10,
    maxAtCampTime = 60,

    // lower value if map contains rain, fog etc.
    maxViewDistance = 1700,
    // destroy breakables (player size is about 70)
    targetBreakableDist = 100,

    mineDefBias = 1.0,
    snipeDefBias = 1.0,
```

```

attackDefBias = 1.0,
defendDefBias = 1.0,
healthDefBias = 1.0,
ammoDefBias = 1.0,
constrDefBias = 1.0,
dynaDefBias = 1.0,
mountMgBias = 1.0,

//////////
// MAP ROUTES
//////////
//maproutes =
//{
//  GOALNAME =
//  {
//    ROUTENAME = {},
//  },
//  GOALNAME2 =
//  {
//    ROUTENAME = {},
//  },
//},
//////////
// MAP FUNCTIONS
//////////

// reset ATTACK & DEFENCE (ETUtil-function ?)
// param _phaseX (see code convention)
// param _isActive true/false - sets goal to given boolean
resetAD = function(_phaseX, _isActive)
{
  // print("InResetAD");
  resetD = "DEFEND_" + _phaseX + ".*" ;
  resetA = "ATTACK_" + _phaseX + ".*" ;
  SetAvailableMapGoals( TEAM.AXIS, _isActive, resetD );
  SetAvailableMapGoals( TEAM.AXIS, _isActive, resetA );

  SetAvailableMapGoals( TEAM.ALLIES, _isActive, resetD );
  SetAvailableMapGoals( TEAM.ALLIES, _isActive, resetA );
},
setMapDefaultBias = function()
{
  SetBiasGoals( Map.attackDefBias, "ATTACK.*" );
  SetBiasGoals( Map.defendDefBias, "DEFEND.*" );

```

```

SetBiasGoals( Map.mineDefBias, "PLANT_MINE_GOAL_.*" );
SetBiasGoals( Map.snipeDefBias, "MAP_SNIPER.*" );
SetBiasGoals( Map.ammoDefBias, "AMMO.*" );
SetBiasGoals( Map.healthDefBias, "HEALTH.*" );
SetBiasGoals( Map.constrDefBias, "MAP_CONSTRUCTION.*" );
SetBiasGoals( Map.dynaDefBias, "MAP_DYNAMITE.*" );
SetBiasGoals( Map.mountMgBias, "MAP_MOUNTABLE_MG42.*" );
},
setMapTrigger = function()
{
    // TBD (copy and paste from /bot makemapgm)
    // f.e. OnTrigger( "MISSING_STRING", callTriggerFunction );
    Map.debug("Trigger set.");
},
// just two avoid 2 lines in code, better readable
debug = function(_out)
{
    if (Map.scriptDebug)
    {
        print(_out);
    }
}
};

//////////
// GLOBAL MAP GOALS
//////////
// TBD (copy & paste from /bot makemapgm)
// END GLOBAL MAP GOALS

//////////
// TRIGGER FUNCTIONS
//////////
// TBD (copy & paste from /bot makemapgm)
// END TRIGGER FUNCTIONS

global OnBotJoin = function( _bot )
{
    _bot.MaxViewDistance = Map.maxViewDistance;
    _bot.TargetBreakableDist = Map.targetBreakableDist;

    _bot.SetGoalProperty("DEFEND", "MinCampTime", Map.minDefCampTime);
    _bot.SetGoalProperty("DEFEND", "MaxCampTime", Map.maxDefCampTime);

```

```

_bot.SetGoalProperty( "ATTACK", "MinCampTime", Map.minAtCampTime);
_bot.SetGoalProperty( "ATTACK", "MaxCampTime", Map.maxATCampTime);

Map.debug( "OnBotJoin ", _bot.Name );
};

global OnMapLoad = function()
{
    EnableScriptDebug(Map.scriptDebug);

    if ( TestMapOn ) { ETUtil.AutoTestMap(); }

    // trigger
    Map.setMapTrigger();

    // setting bias defaults
    Map.setDefaultBias();

    if (Map.routingEnabled)
    {
        Util.Routes( Map.maproutes );
        Map.debug("Routes added to map.");
    }

    // Set first goals/routings here ... or reset goal bias
    // reset goals
    // Map.resetAD( "P1", false );
    Map.debug( "First goals set.");

    Map.debug( "OnMapLoad ", Map.name);
};

global OnBotLeave = function( _bot )
{
    // Do stuff
    if (_bot) { Map.debug( "OnBotLeave ", _bot.Name );}
};

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=Map\\_template](http://omni-bot.com/wiki/index.php?title=Map_template)"

This page has been accessed 151 times. This page was last modified 14:32, 13 October

# ET Waypointing Tutorial/Archives/066

From Omni-bot Wiki

This is an introduction to making waypoints for Omni-bot. Most examples used here come from Detoeni's Alpine Assault map for the game RTCW: Enemy Territory. However, it should be relatively easy to transfer the information given here to other maps and games.

If you want to print this tutorial, there is a printer-friendly PDF Version available (1.3 MB).

Before starting to make your own waypoints, you should familiarize yourself with the topic by studying some of the waypoints for the 'official' maps that come bundled with Omni-bot. In order to do this, load the map in question, bring down the in-game console and type `/bot waypoint_view on`. You will see waypoints and connections between them, much like the ones you are about to create.

Also, an important prerequisite for making good waypoints is that you have played with bots before for some time, because you should have a general idea of what bots can and can't do before starting to waypoint.

## Contents

- 1 Understanding the "story" of the Map
- 2 Waypointing: The First Steps
- 3 Streamlining Bot Navigation
- 4 Using Waypoint Flags
  - 4.1 Navigational Flags
  - 4.2 Tactical Flags
- 5 Map Scripts

## Understanding the "story" of the Map

Before you start waypointing a map, it's important that you have a good understanding of the map's game flow and know your way around the map. You may still discover some things when in waypoint mode, but generally, you should know what's going on in the map before you start. In our Alpine Assault example, a rough outline of the map's "storyline" seen from Allied perspective would be:

Starting from your first spawn point, capture and defend the second spawn in the garage:



Gain access to the mine complex, where the Axis is defending the safe key in the Armory:



Grab the key from the desk in the Armory, and try to reach the safe with the documents ...



... which will of course be guarded by the enemy. Steal the documents from the safe ...





... and escape to the safety of your first spawn point.



*All stills were taken on location with real bots. No bots were harmed in the making of these screenshots.*

With such a basic outline in mind of what a typical game flow will be like, you'll be able to build efficient paths for the bots, and above all, make a good map script.

## Waypointing: The First Steps



**1. Start a listen server.** Start a server with a client on the same machine (a.k.a. "listen server", as opposed to a dedicated server). Normally, double-clicking the shortcut that Omni-bot generates on your desktop during the installation process will do just that.  
If you don't know how to start, visit the Omni-bot F.A.Q.

**2. Load the map you want to waypoint**, either by selecting "Host game" from the game menu and selecting the map from the list, or by typing

`/map <mapname>`

into the game console. Instead of `/map`, you could consider using `/devmap`, which enables some "cheats" (mainly intended for use by map developers), such as walking through walls, warping to a given position, etc.

**3. Disable the time limit of the map** or set it to something very high if possible. In ET, use the command `/ref timelimit 0` to have unlimited time on a map. If a map restarts while you are editing waypoints, your unsaved changes will be lost.

**4. Join a team.** On most maps, joining the attacking team as an engineer is the best thing to do, because then you can dynamite obstacles out of your own way, build bridges etc. If you used `/devmap` to start the map, try issuing the `/noclip` command in the game console and experiment a little with what you can do. Use `/noclip off` to turn it off. It's better not to add waypoints in noclip mode, but it might help to reach some spot very quickly that would be otherwise very difficult and cumbersome to reach.

**5. Enable waypoint mode and start adding waypoints.** Waypoint mode is turned on by using the `/bot waypoint_view` on command. Add waypoints by issuing `/bot waypoint_add` in the console, while running around the map as a normal player would. A single waypoint will look like this:

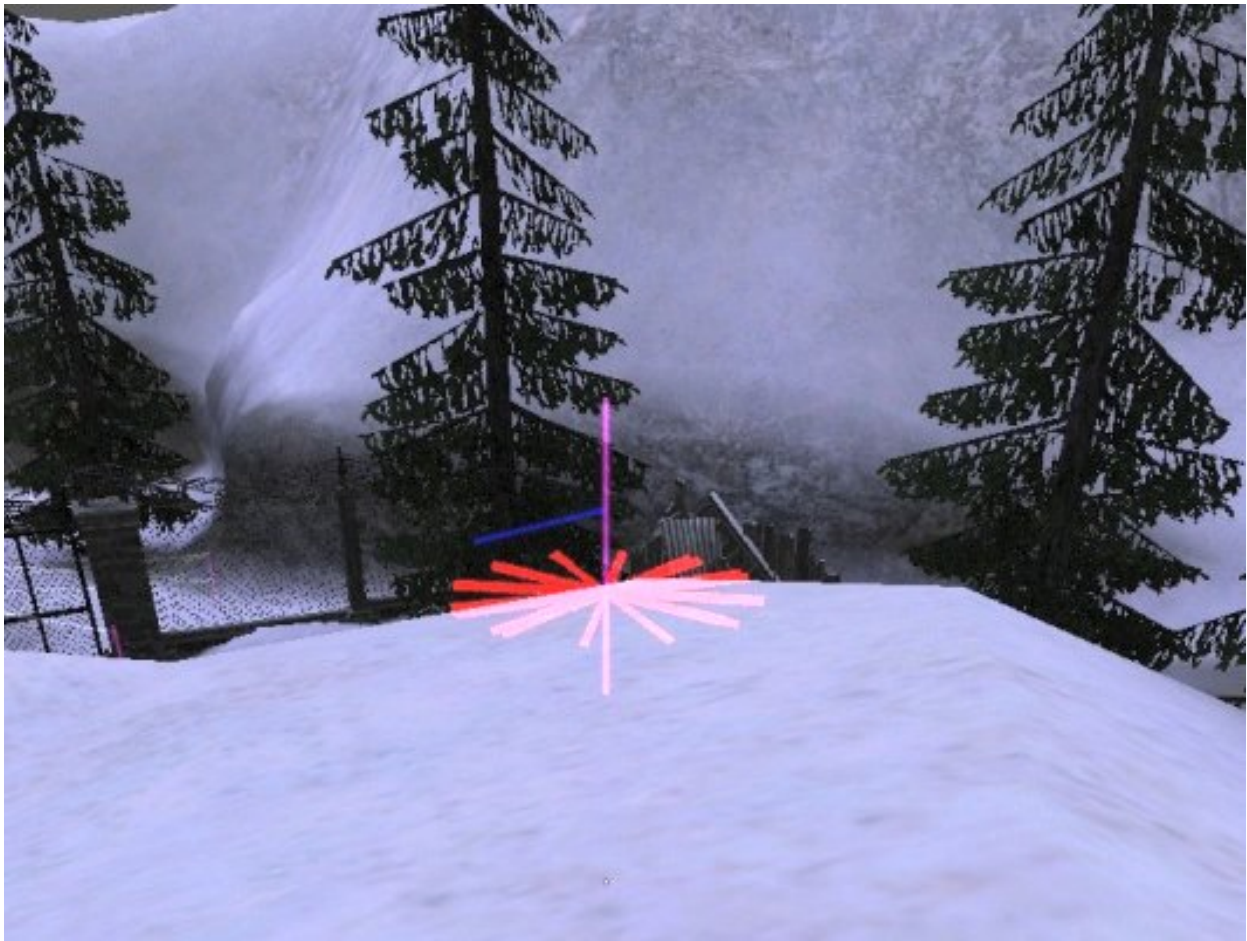


**6. Basic waypoint properties.** When you hover your crosshairs over the waypoint, some information about it will be echoed to the screen:

- the waypoint's number,
- the waypoint's radius,
- the waypoint's unique ID (this will always stay the same, even if you delete some waypoints later on, while the number may change).

While the number and UID are relatively unimportant at the moment, the radius is worth mentioning. The radius is effectively the tolerance for moving toward a waypoint. If you need the bot to get closer, use a smaller radius, if the waypoint represents a large room, use a large radius. You can change a waypoint's radius with the command `/bot waypoint_setradius <value>`. However, leaving it at the default radius of 35 game units will be okay in most cases. We will get back to radius later on.

**7. Advanced waypoint properties.** Additionally to a radius, UID and number, all waypoints added with Omni-bot 0.6 and higher will have a *facing*, which is basically the direction in which you looked when you added the waypoint. Use the command `/bot waypoint_viewfacing on` to see the facing, indicated by a blue line:



A waypoint's facing is a vector associated with the waypoint that can be used to tell the bot in which direction to aim in some situations, e.g. when sniping. For merely navigational purposes, however, the facing is irrelevant in most cases. We will cover facing later on.

Finally, a waypoint can have one or more *flags* that can serve different purposes, for example tell a bot to jump, prone, crouch or walk. We will go into the details of flags later in chapter 5, just keep in mind that there are such things for now.

**8. Connect the waypoints.** By themselves, waypoints are pretty useless. They have to be *connected* to each other in order to enable the bots to move. The waypoints together with their connections can be considered a (directed) graph that gives a rough representation of the map's geometry. It's up to you if you first add a large number of waypoints and connect them later on, optionally using the `/bot waypoint_autobuild` command, or if you want to connect every newly added waypoint to one or more other waypoints immediately after adding it.

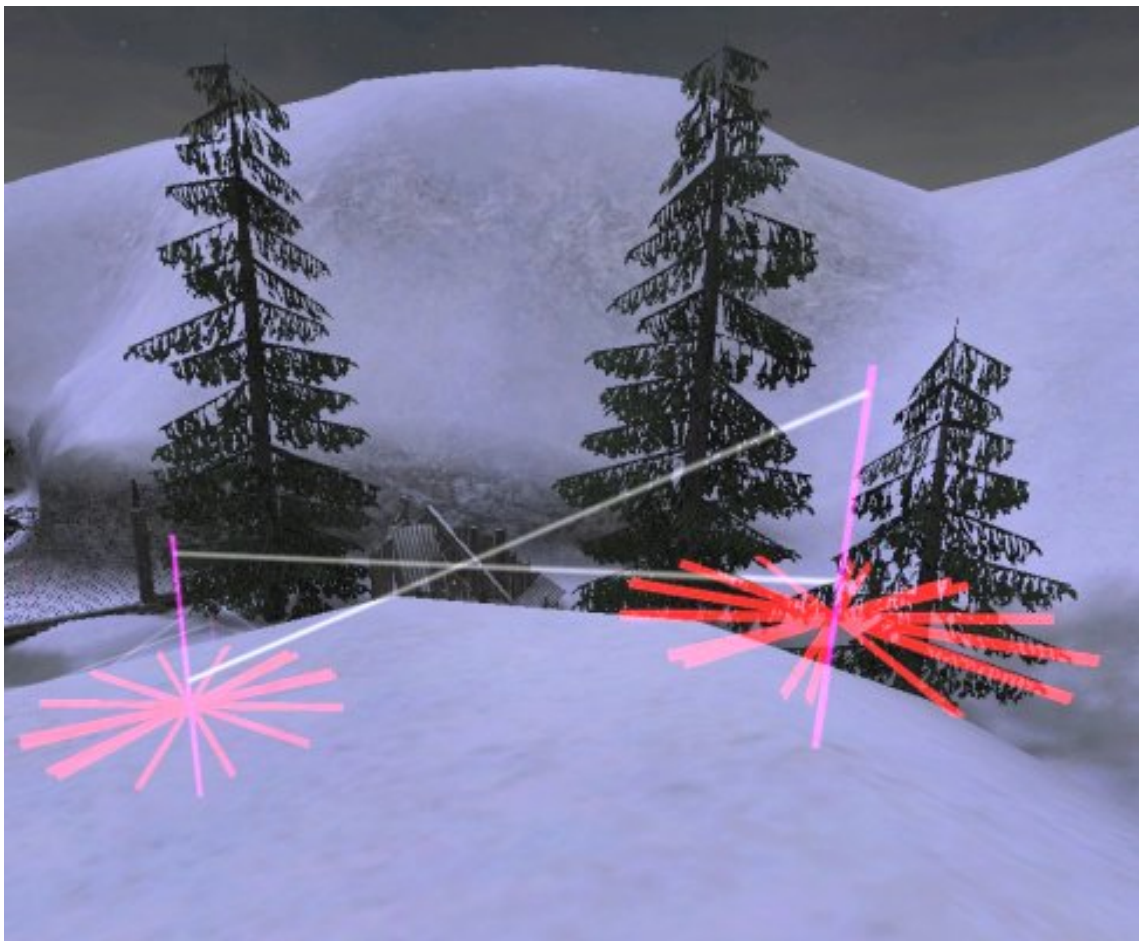
A one-way connection between two waypoints is added by issuing the `/bot waypoint_connect` command in the console twice, while standing over both waypoints. The connection will look like this:



This screenshot shows two waypoints connected by a one-way path. We know the connection here is from the right waypoint to the left because of the angle of the link. Link direction is shown by a downward sloping link.

In the following screenshot we see what a two-way connection looks like, which is added by using the `/bot waypoint_biconnect` command (or by using `/bot waypoint_connect` twice on each waypoint). Since there is a connection going both ways, the resulting link appears as an 'X' between the waypoints:





It is strongly recommended to use bidirectional connections in most cases; one-way connections should be constrained to jumping down and other special cases. The distance between the two waypoints shown above is a good one to use in many cases. As a general rule, distances much greater than, say, 200 game units should be avoided in most cases, because it allows for much smoother bot navigation if the paths between waypoints are relatively short.

**9. Save your waypoints** using `/bot waypoint_save`. This saves the waypoints for the currently loaded map to a file called `<mapname>.way` in the `nav` directory under the game currently running.

*Warning:* This will overwrite an existing waypoint file of the same name without asking. Rename or backup your original waypoint files when you want to experiment.

Generally, it is a good habit to save often during the process of waypointing a map. But since there is no undo command at the moment, do not save changes that are extremely experimental without taking a moment to think before. The only way to undo mistakes is to load the previously saved version.

Now you should be able to add one or more bots. If you have added waypoints close to the spawn areas, you will see them move around and do more or less useful things, and most likely get stuck occasionally. This is why we have to fine-tune our rough-and-ready waypoints a bit.

---

**TIP:** Bind your most frequently used commands to keys. In ET for example, a bot command is bound to a key using the following command in the game console: `/bind <key> "bot <command>"`. For example: `/bind f5 "bot waypoint_setradius 25"`, `/bind f6 "bot waypoint_addflag blockable;bot waypoint_addflag wall"`

You can easily save lots of these bind commands to a plain text file, say `waypointing.cfg`, and later execute this file by issuing the following in the console: `/exec waypointing.cfg` (Save the file in your `etmain` folder.)

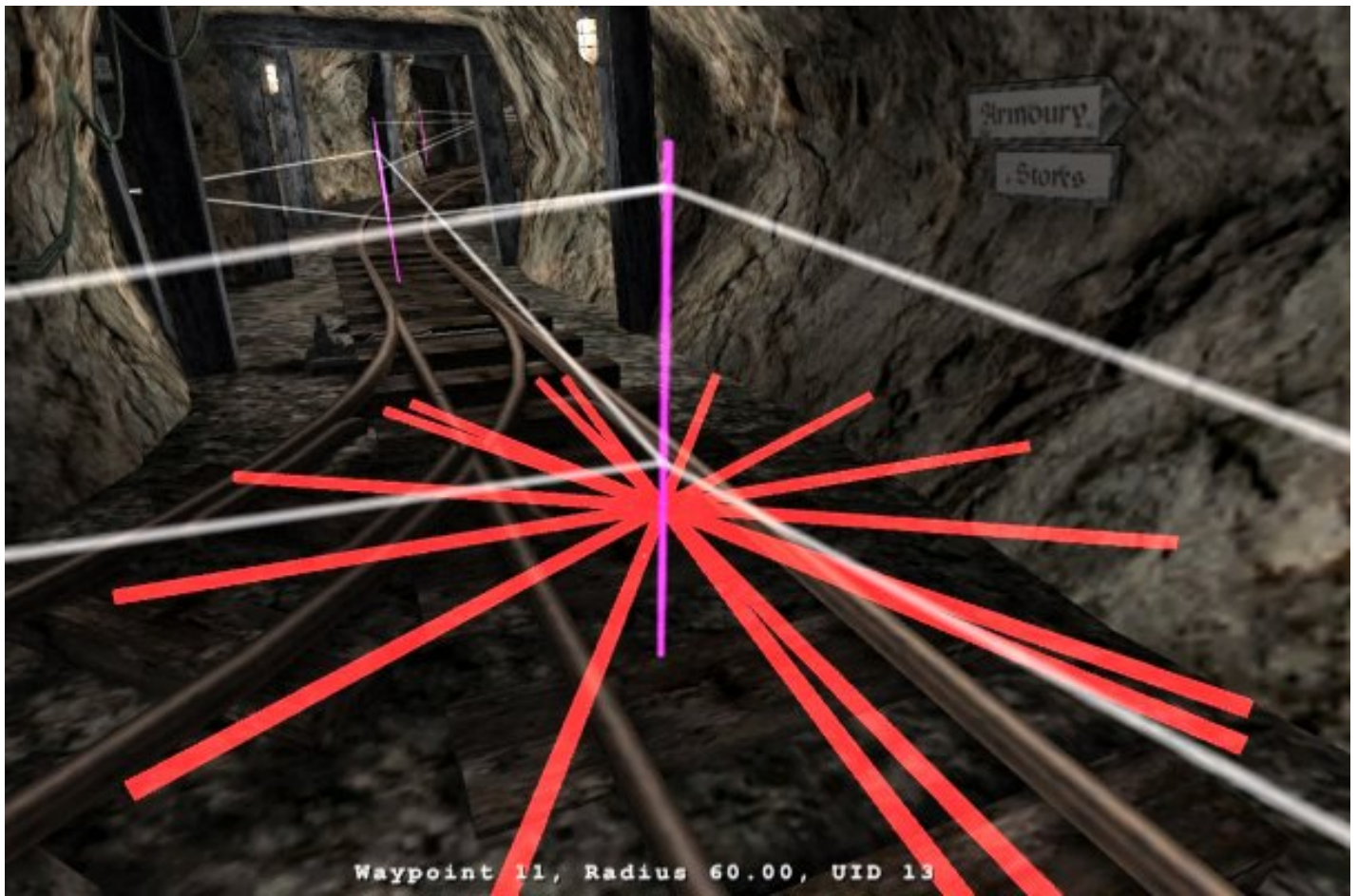
---

## Streamlining Bot Navigation

**1. Radius in more detail.** As mentioned before, a waypoint's radius is the tolerance for moving towards a waypoint. If you want the bots to get very close to a certain spot, use a smaller radius, if it doesn't matter much if a bot reaches this specific waypoint exactly, use a large radius. You can change a waypoint's radius with the command `/bot waypoint_setradius <value>`. Don't use radius values of less than 10 or so.

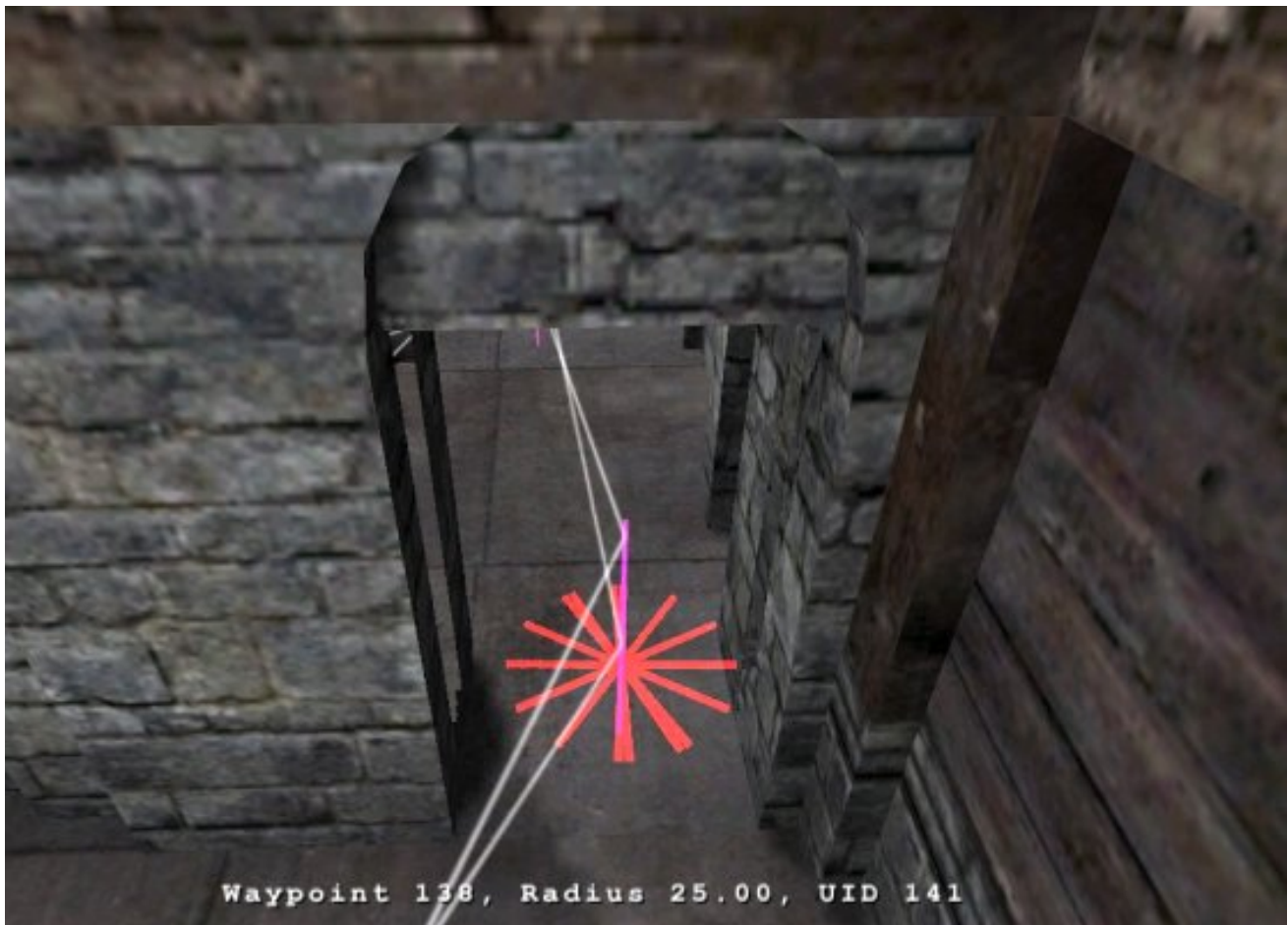
While the default radius of 35 game units will do in most cases, some situations require some radius tweaking. In areas where there is plenty of space, a larger radius will make it easier for the bots to avoid running into each other when they meet on a waypoint, so a radius of 60, 80, or even 100 will be a better choice:





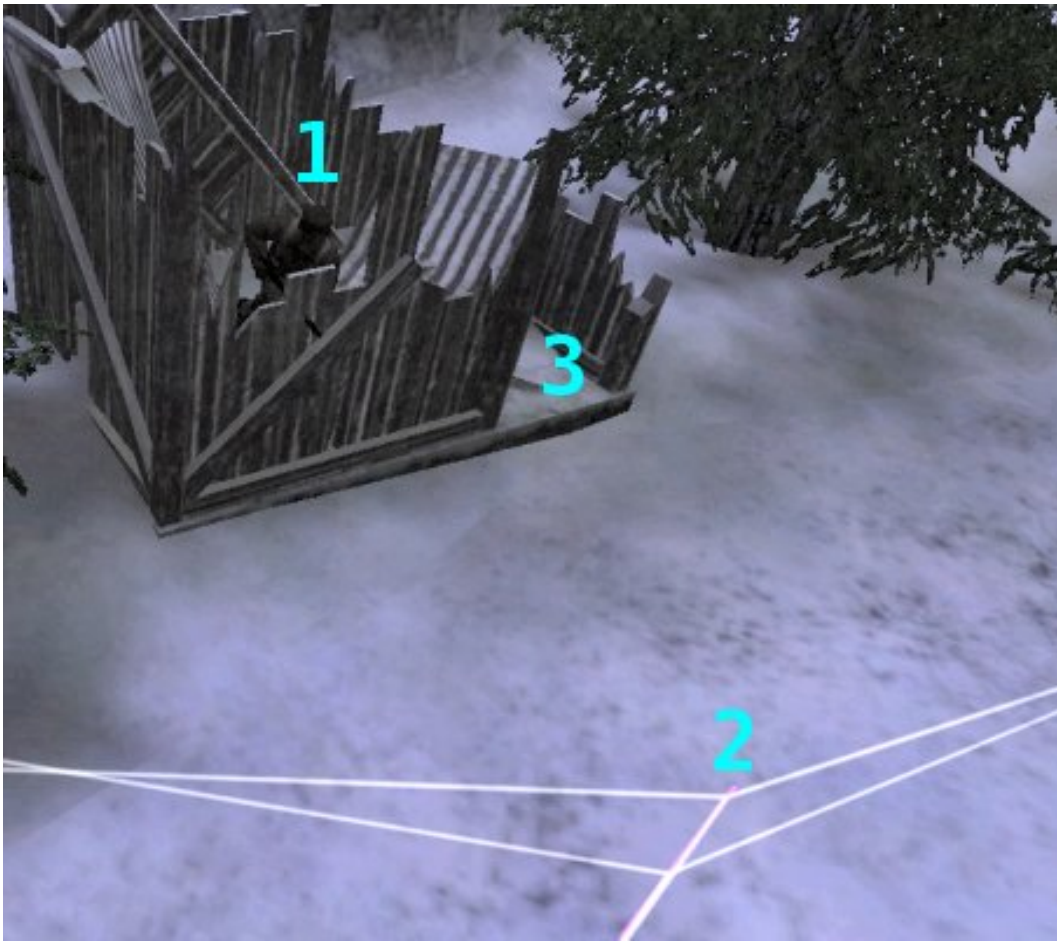
In tight areas, in particular at doors and ladders, use a smaller radius, but normally not much smaller than 20. Make sure that the red "beams" that visualize the radius do not hit walls or other objects:





**2. Avoiding stuckage.** When you try to add bots to a map you are waypointing, they will almost inevitably get stuck at first. A typical example of what has come to be called 'stuckage' can be seen in the following screenshot:





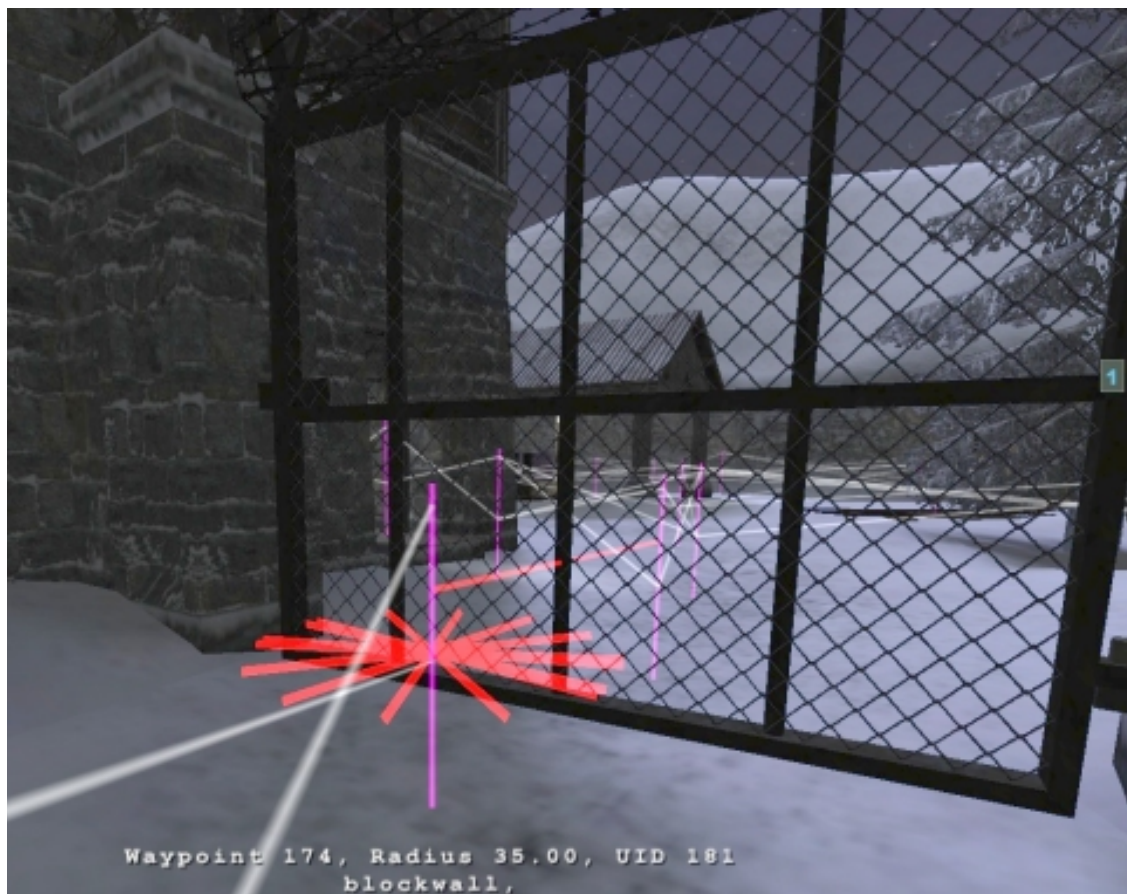
The bot (1) is trying to move directly towards the closest waypoint (2), unaware of the path 1 - 3 - 2 a human player would naturally choose. Even though there is no path the bot could possibly have followed to reach its present location (1), this type of problem is very common, because bots often get pushed off their intended paths by other players or due to the impact of weapons, especially grenades and such.

The next image shows a solution:



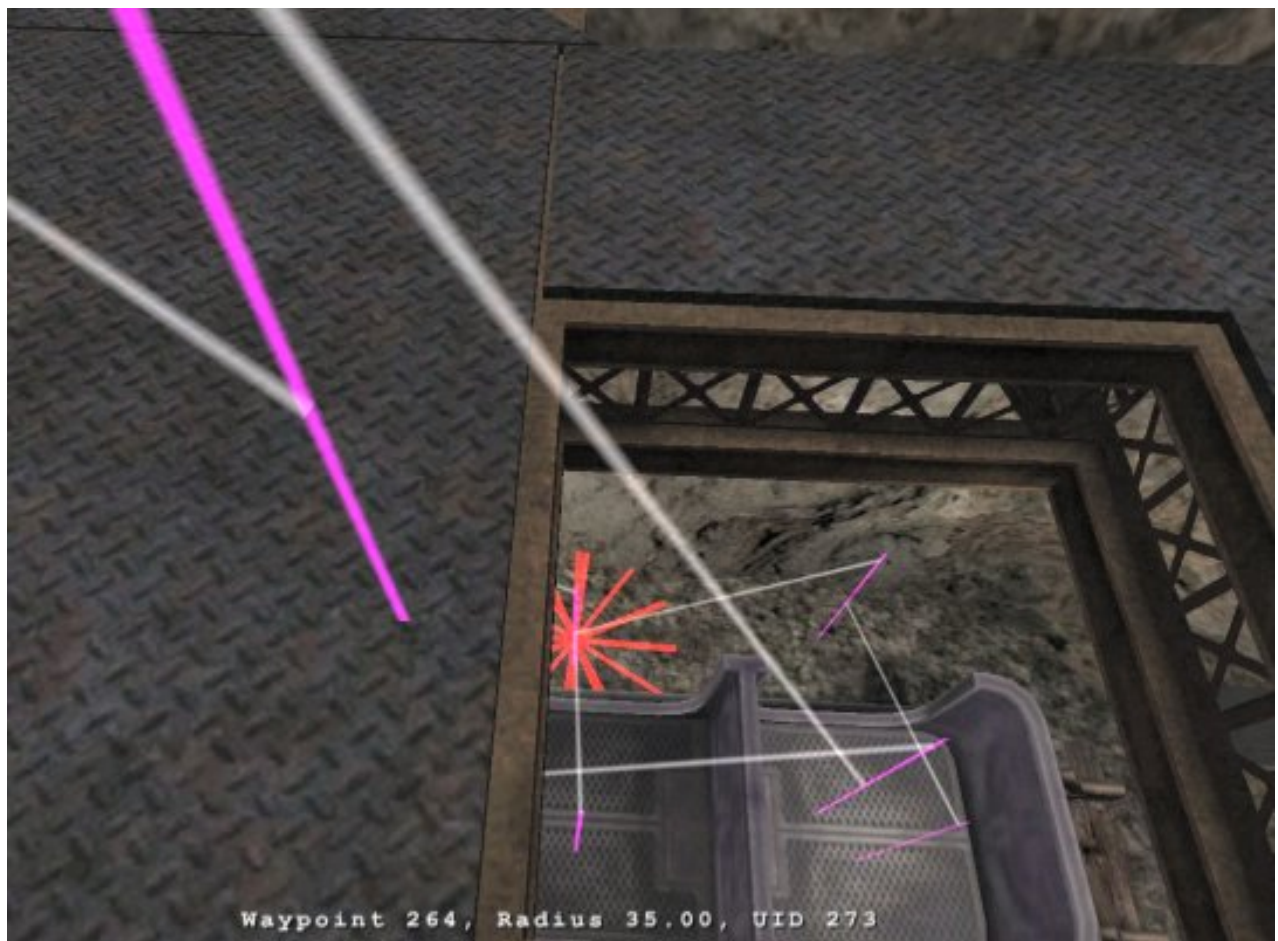
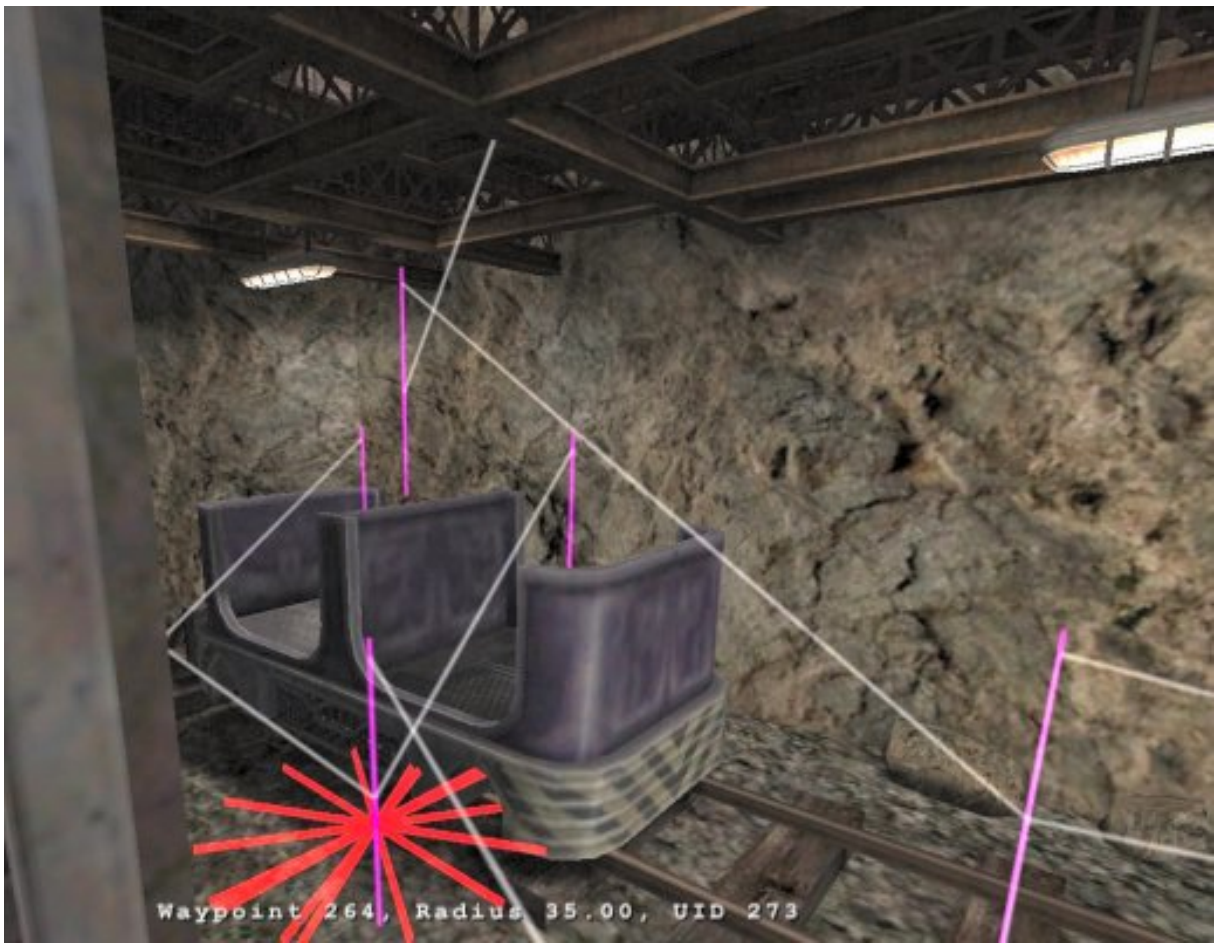
In the above example, adding a waypoint at position 3 is sufficient to "free" the bot that got stuck at 1, because 3 rather than 2 will be the closest waypoint once it is added.

**3. Make sure the bots can reach the objectives.** A rough sketch of the bot logic is as follows: First the bot picks a goal, which can be just going somewhere, or something to achieve, like constructing or destroying some object. Then the bot computes a path to this goal along the waypoints given by the waypointer. Therefore, it is important that there are waypoints close to the objectives you want the bots to accomplish. The following screenshot shows a dynamite/satchel target with waypoints nearby:



**4. Make moderate use of one-way paths.** Most of the time, bidirectional connections between waypoints should be preferred. Just think of medics to see why: dead or injured players can end up virtually anywhere on a map, thus the more spots on the map bots can reach, the better. In some situations, however, one-way connections are in order, in particular for jump-down paths. The following screenshots show an example of legitimate use of one-way connections from two different perspectives:





By the way, when you want to connect waypoints in a situation similar to the one shown above, the `/bot waypoint_connectx` command might be helpful. It is almost the same as `waypoint_connect`, but applies to the waypoint you are aiming at. Thus you don't have to actually jump down in order to connect the upper waypoint to the lower one.

## Using Waypoint Flags

Waypoint flags can serve a large number of purposes. A flag is added to a waypoint by standing over the waypoint and writing `/bot waypoint_addflag <flagname>` into the console. In order to remove a flag, use the same command. In order to remove all flags of a given kind, use `/waypoint_clearallflags <flagname>`. `<flagname>` can be a list of flags, both when adding and deleting flags. When you hover your crosshairs over a waypoint, the flags it carries will be echoed to the screen along with other information about it.

The following is not intended as a complete guide to all the waypoint flags, we will cover only the most basic flags and their usage here. See the Wiki flag list.

### Navigational Flags

**Movement flags.** The names of the crouch, prone, sprint, climb and sneak flags speak pretty much for themselves: They affect the way a bot will move towards a waypoint with these flags. Used in conjunction with a snipe flag on the same waypoint, however, crouch and prone will be interpreted as a stance to be in, i.e. the bot that snipes there will crouch or prone all the time.

The difference between the jump, jumpgap and jumplow flags is as follows:

- jumpgap will make the bot check the ground ahead for gaps to jump over,
- jumplow will make the bot check for obstacles such as small walls, crates, boxes, and jump upon or over them,
- jump will make the bots jump without any of those checks.

jump is useful if the detection methods mentioned above fail. The other jump flags should normally be preferred.

**The door flag.** Doors are typically waypointed depending on how they operate. In many games, doors are automatic and simply open when the player approaches them. It isn't necessary to do anything special in the case of those doors. In some games, doors have to be "used", such as in Enemy Territory. In these cases, the door waypoint flag is useful. The door flag is used by adding the flag to the two waypoints on either side of a door. This tells the bot to hit the USE key while moving across the link between the two waypoints. In the following picture we see both waypoints flagged with door, and in this case they are team specific:



**Team flags.** The purpose of team flags is to make a waypoint off limits for any team whose team flag it hasn't. The teamonly flag is added automatically once you add any of the team1-team4 flags to a waypoint. For example, a waypoint can have a team1 and a team3 flag, which will make it unavailable for Team 2 and Team 4. In Enemy Territory, it obviously doesn't make sense to add both team flags (there are only two), because having both is as good as having none at all, it will just confuse any waypointer who might try to update your waypoints in the future. Also in Enemy Territory it's more convenient to use the alias axis for team1, allies for team2. The teamonly flag will be added automatically and should not be used directly.

Keep in mind that team flags will mark the waypoint as a no-go area for the other teams, so use with care. Team flags are most useful in conjunction with door flags ("team doors"), and perhaps the snipe, defend, cappoint and attack flag. They can also be used at spawn point entries (thinking of huts and the like), in order to keep bots out of the other team's spawn areas. (This is considered good style by many players.)

**The cappoint flag** is used only on the "capture and deliver" style of maps. On those maps, however, it is crucial. The bot will auto-detect the items to be stolen (captured), but not the point on the map where these items should be delivered. Use /bot waypoint\_addflag cappoint to mark a place where the bots should deliver the objectives. It will create a goal for bots carrying objectives that should by default override most other goals.

**Blockable paths.** The blockable flag deserves special mentioning. Blockable paths are a feature of Omni-bot that allow paths to update their own blocked status, saving some scripting effort. Similar to door, the



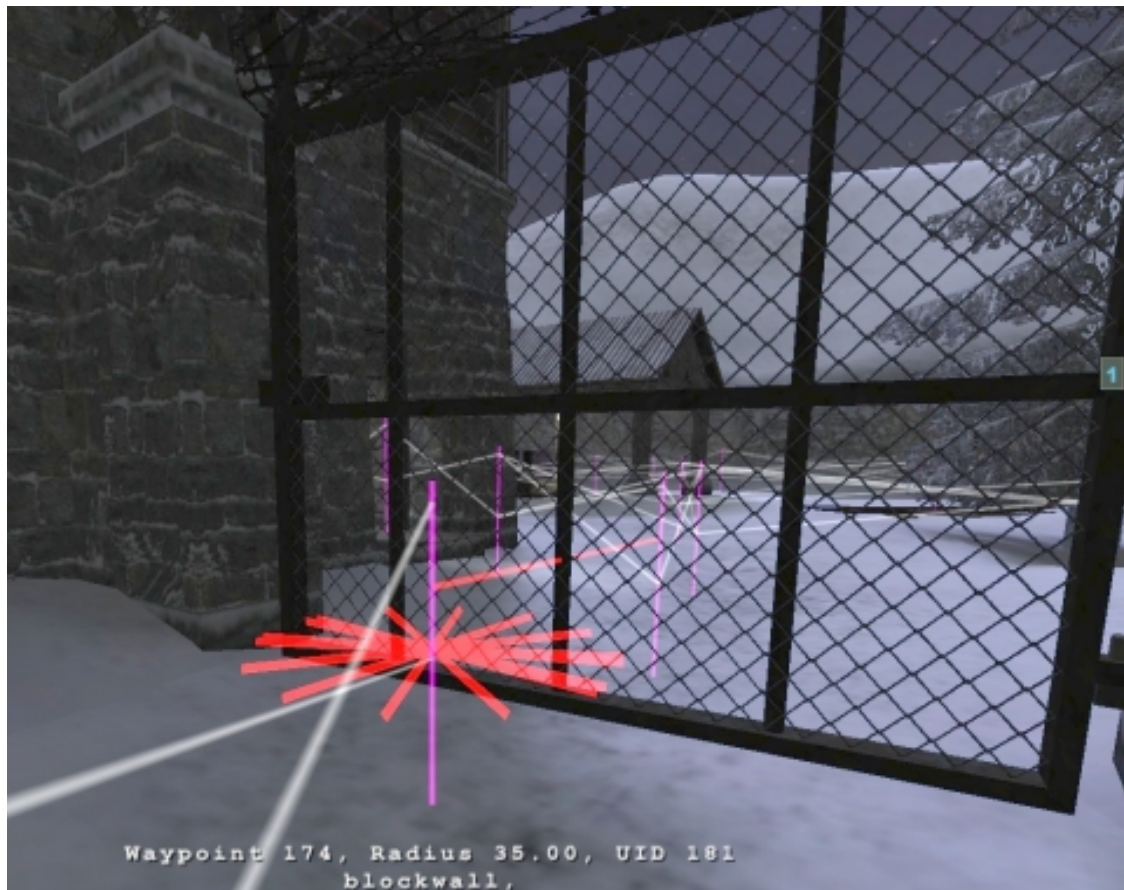
blockable flag needs to be placed on both sides of the object between them (a wall or a bridge, typically). The blockable flag *must* be used in conjunction with another flag that determines the type of blockable path that is desired. The type of flag used in addition to the blockable flag determines how the blocked status is determined.

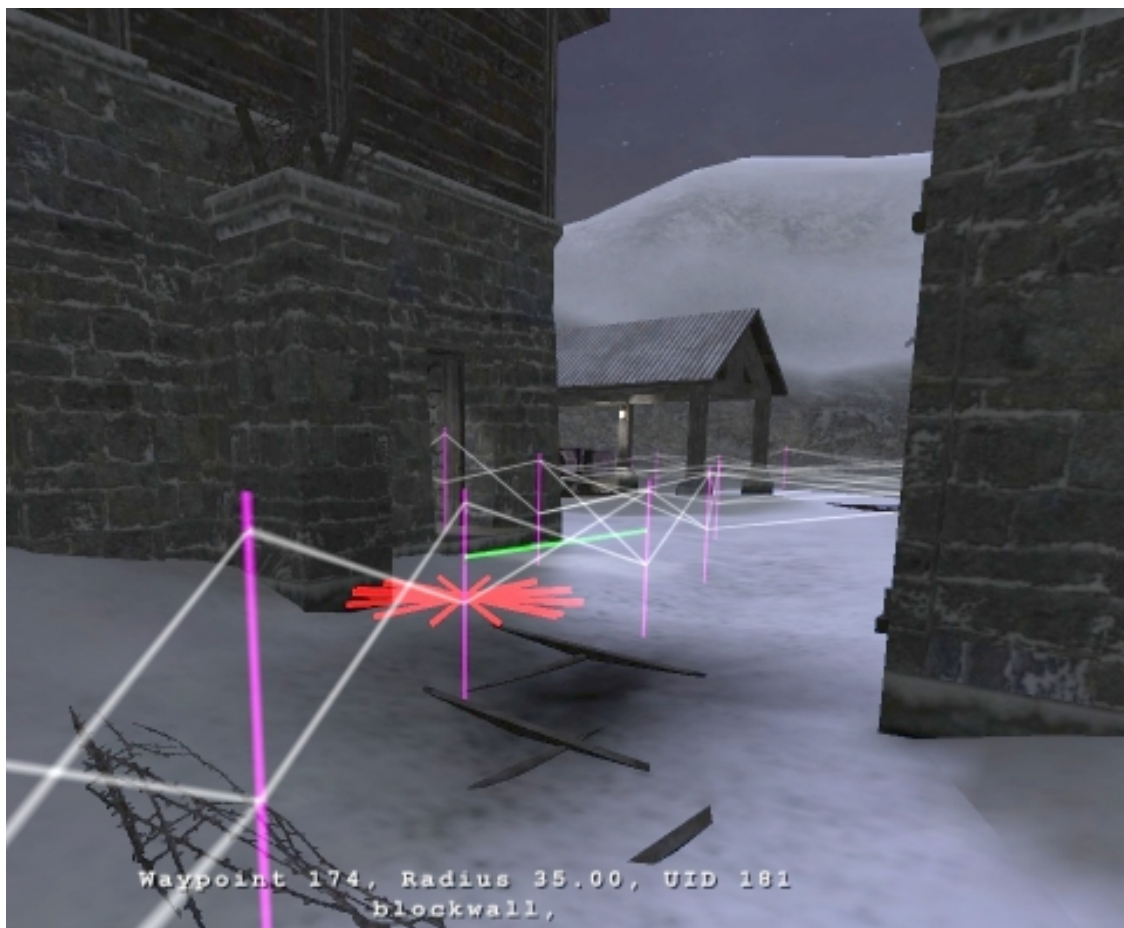
There are two types of blockable paths:

- wall - If a line of sight check passes between the 2 waypoints, the path is opened, otherwise it is closed.
- bridge - If an object isn't detected half-way between the 2 waypoints (such as a bridge, a barrier or the like), the path is blocked, otherwise it is open.

When in waypointing mode, a red line marks a path that is detected as blocked, while a green line indicates a free path.

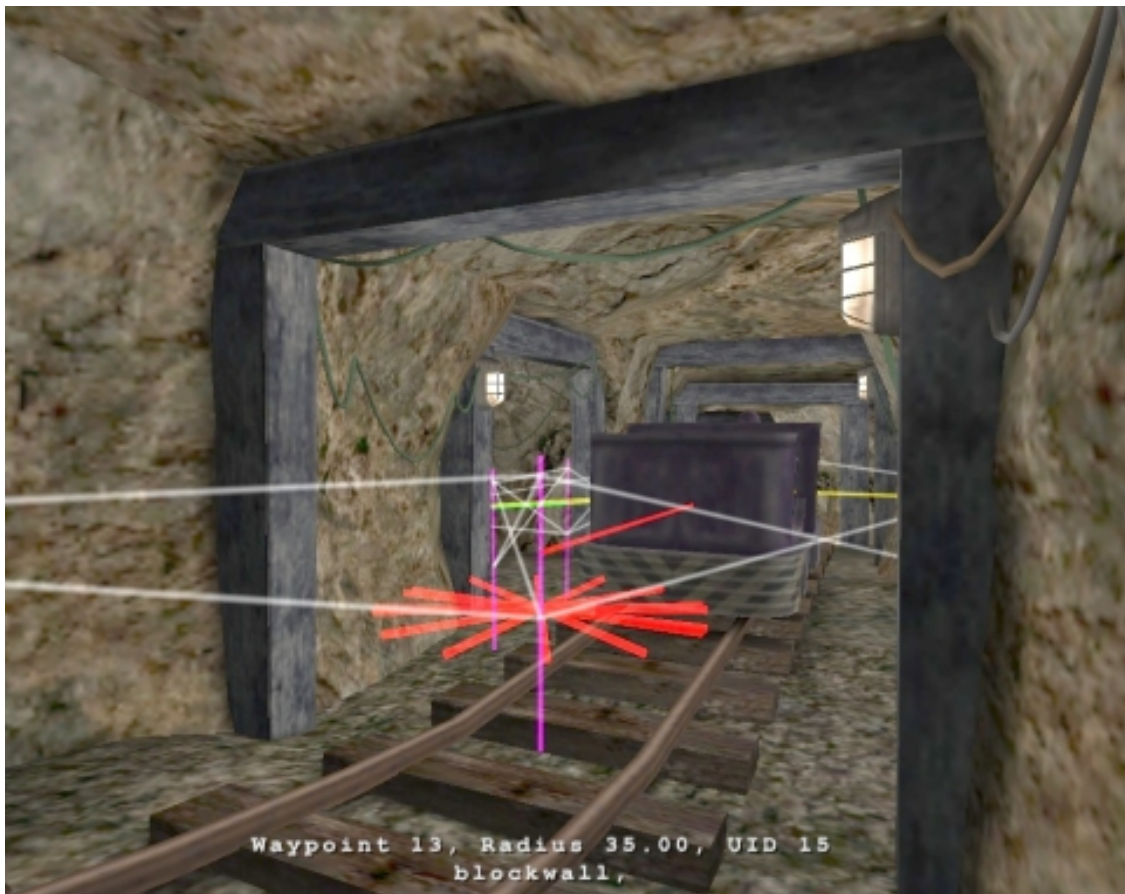
An example of blockable/wall:





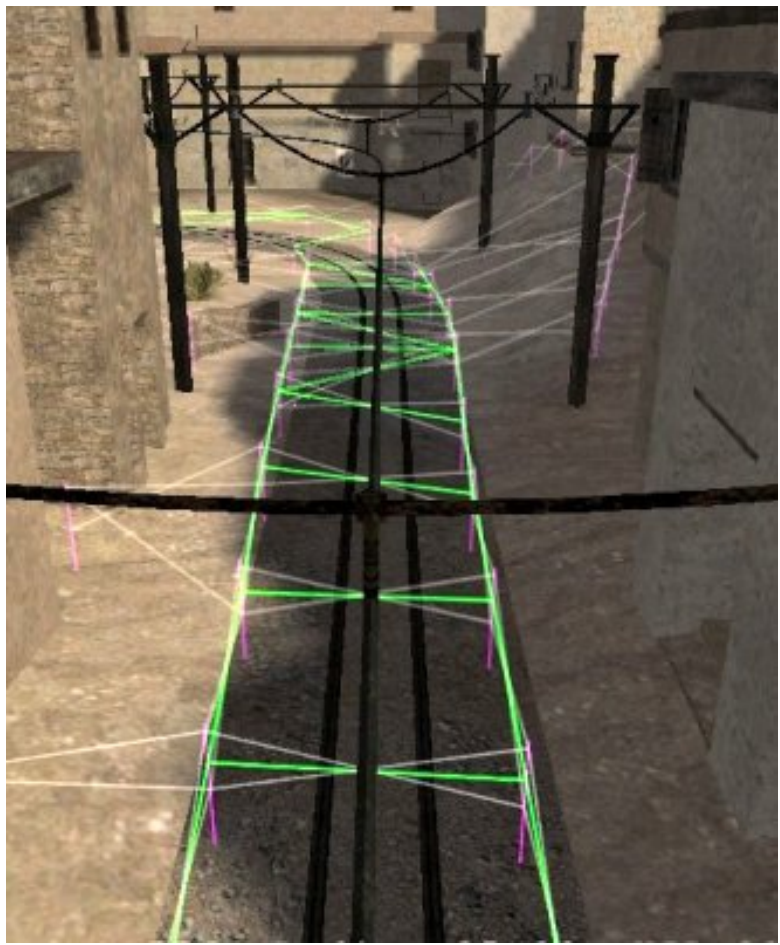
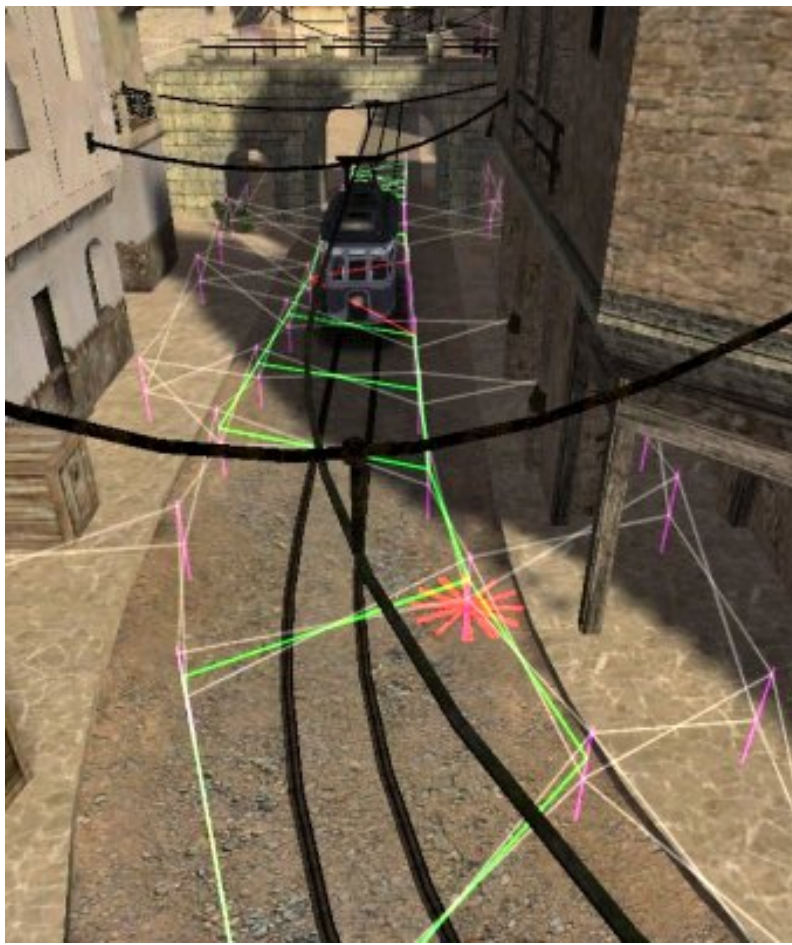
Keep in mind that vehicles can block paths, too. As far as Omni-bot navigation is concerned, vehicles are "wall-like" objects:



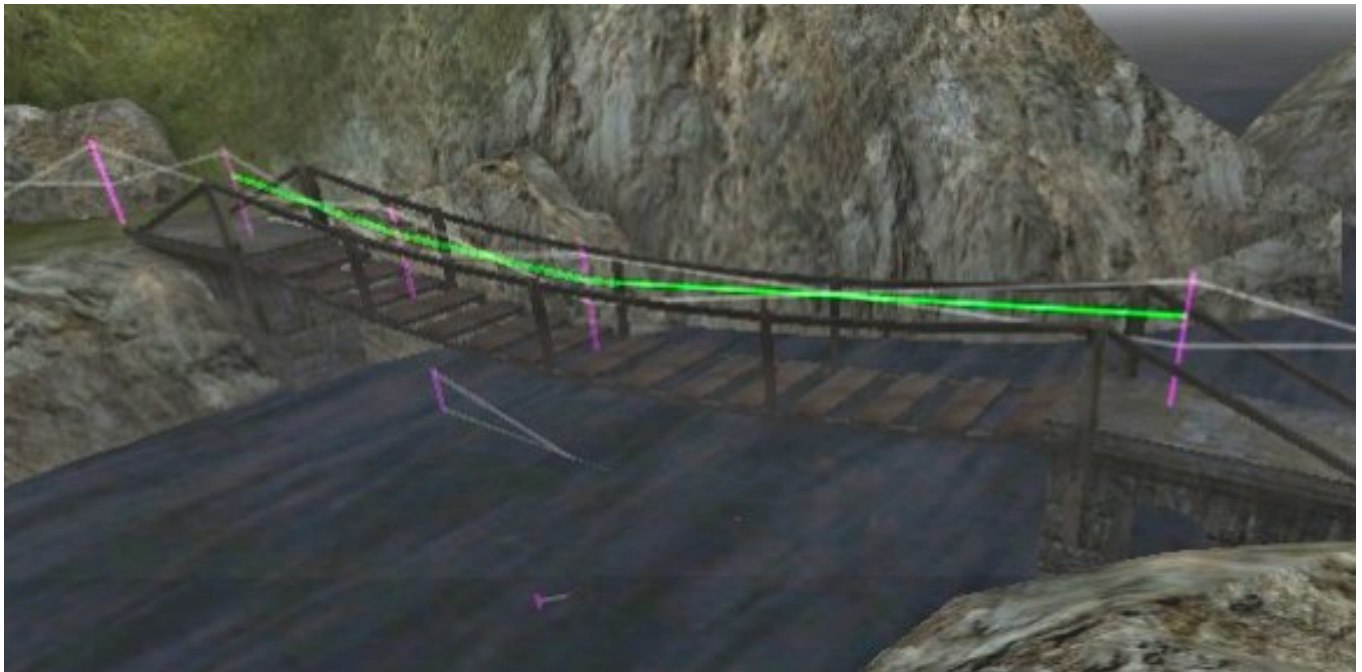


Note how the waypoints in the image above indicate an alternative path around the vehicle.

Normally, all the waypoints along the path of a vehicle should have the blockable and wall flag set, because otherwise the bots will run into the vehicle when it is destroyed or standing still for some other reason. The tug in Alpine Assault is a special case, because it can only stop at predefined places and is indestructible. There is a page on the Omni-bot Wiki dedicated to vehicle pathing. Here's how a vehicle route will typically look:



An example of blockable/bridge (taken from the Bergen map):



The check for bridge is basically a probe to see if there is a surface detected under the link between the waypoints. When there is no bridge built, the probe doesn't detect the surface, and results in a blocked



path. When the bridge is built, the surface is detected, and the link is opened.

Due to this method of detecting the blocked status of a path across a bridge, rope bridges and similar constructions with a slightly curved shape may require three or more waypoints, all of which should have blockable and bridge flags. Otherwise, the detection of a surface beneath the path might fail, as can be seen in this screenshot:



See the Waypointing page on the Omni-bot Wiki to learn more about blockable paths.

## Tactical Flags

The three waypoint flags that influence the bots' tactical behavior are attack, defend, and snipe. They create goals for the bots, much like the constructible and destructible objects that are auto-detected when a map is loaded.

**Sniper spots.** Probably one of the most popular and most widely used waypoint flags is the snipe flag. As with all flags, a snipe flag is added to a waypoint by standing over the waypoint in question and issuing the command `/bot waypoint_addflag snipe` in the game console. This flag makes bots that have the appropriate weapons camp on this waypoint for some time with their sniper rifle in scoped mode. Do not use it to make bots switch to scoped mode somewhere in an open area, they will camp there and might make fools of themselves, standing still in the middle of a market place with tons of enemies around them. See the Wiki page about sniper spots to learn more.

It is important to understand that sniper waypoints must have a valid facing. Without the facing, you could as well not add the sniper flag at all. A waypoint's facing is basically a vector associated with the waypoint that tells the bot in which direction to aim when sniping. Since version 0.6, Omni-bot adds the waypointer's direction of view automatically to a waypoint when it is placed. However, it's a good habit to adjust the facing the very moment you are adding the snipe flag, using the `waypoint_setfacing` command. You can easily bind this functionality to a key like this:

```
/bind f5 "bot waypoint_addflag snipe;bot waypoint_setfacing"
```

There are several reasons for adjusting the facing at the time the snipe flag is added: The waypoint may have no facing at all, because (some of) the waypoints were made with an older version of Omni-bot, or it may have some random facing that is totally inappropriate.

It is a good idea to actually join as a sniper when you plan to add sniper spots to a map. Then before issuing the setfacing command, try to take the position the sniping bot will have (e.g. go prone etc.), scope your rifle and aim where you want the bots to aim. When you use the `/bot waypoint_setfacing` command, your current aim direction will be saved as the waypoint's facing.

Also, sniper spots should in many cases have a smaller radius than other waypoints (say 20-30), because the exact position matters a great deal: A few inches to the left, and the sniper will expose himself to the enemy's sight, a few inches to the right, and the sniper might aim in a direction where he will hardly ever get a target, or aim at a rock that is two yards in front of him, blocking his sight.

Two typical sniper spots on the Alpine Assault map:

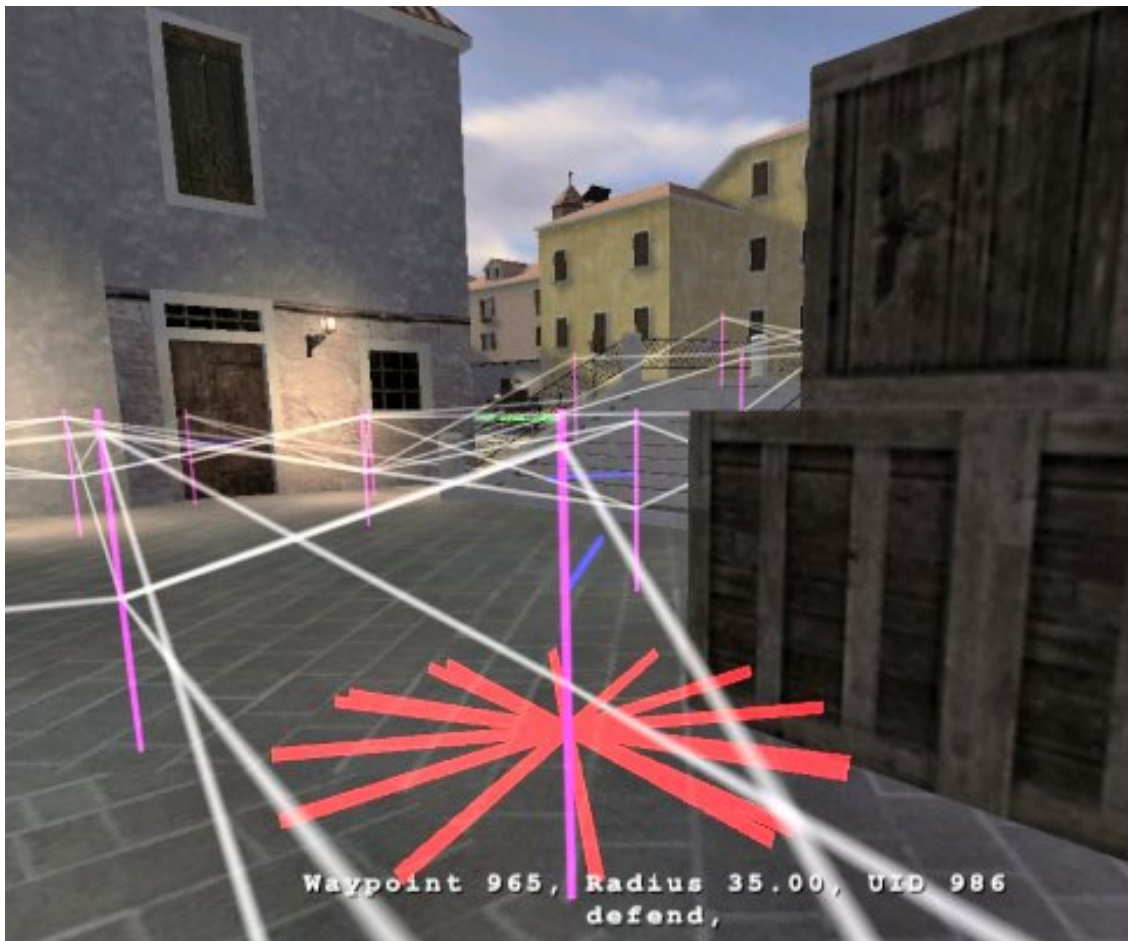




**Attack and defend flags** are basically only goto goals, though by default, the bots will camp on defend spots for some time, which is not the case for attack spots. (This can be overridden via script.) Note that the attack goal will take precedence over the defend goal, so attack and defend goals shouldn't be active for the same team at the same time. attack and defend flags should be placed on waypoints near important objects, ideally in places where the environment provides some cover for the bots.

The following screenshot (taken from the Venice map) shows a well chosen defend spot on the square in front of the library. The facing is set in such a way as to make the bot primarily aim towards the bridge, from where an attack is likely to be expected; the crates on the right provide a good cover:





**Make your attack, defend, and snipe spots easily accessible from scripts.** Attack, defend, and sniper spots will frequently be obsoleted by the game flow. Using the well-known Venice map (ET) as an example again: It makes obviously no sense for bots to camp on the market place (where the game starts) once the Allies have taken over their second spawn. This is where the map script comes in. The script can serve other purposes as well, but disabling and enabling certain goals for the bots based on what has happened on a map is one of a script's primary tasks.

In order to make the goals easier to access from a script, name them using the `/bot waypoint_setname` command. Ideally, make up some naming scheme before-hand. If you put the team name in the waypoint name, this makes things even easier: For example, waypoint names like "phase 1 axis 1", "phase 1 axis 2" and so on will create goals for the bots that will look like "MAP\_SNIPER\_SPOT\_phase\_1\_axis\_1" or "DEFEND\_phase\_2\_axis\_3". The next chapter will provide a (very) brief introduction to map scripting, and make it clear why this is helpful.

## Map Scripts

**Introduction.** This isn't the place to provide a full-fledged map scripting tutorial; see the appropriate Wiki page for all you need to know about that. All we're up to here is an introduction to the most basic concepts from a waypointer's point of view, mostly based on our Alpine Assault example.

A map script is a plain text file (written in a scripting language called GameMonkey, if you're interested), which should have the same name as the waypoint file with a `.gm` extension, and should be in the same folder (your nav folder). You can use the bot command `makemapgm` to generate a skeleton map script for

you in the folder /omni-bot/et/user/. For our present purposes however, it's easier to start from scratch.

As we said in the last chapter, the map script's chief task from our current point of view is disabling and enabling certain goals for the bots, based on events that happen on a map. Let's see what that means.

**The OnMapLoad function.** The most important place to begin with is the OnMapLoad function. It is called automatically after the waypoint file is loaded and the bot goals are initialized. This function is the place where we set up the initial state of the goals in the map, as well as trigger callback functions that should be called when events happen in the game.

Pretending that we have set all our waypoint names according to a common scheme ("phase *n* axis *i*"/"phase *n* allies *i*"), our OnMapLoad function could look like this:

```
global OnMapLoad = function()
{
    //Disable all snipe, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "MAP_SNIPER_SPOT_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_SNIPER_SPOT_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "MAP_SNIPER_SPOT_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_SNIPER_SPOT_phase_1_allies.*");

    //Set up triggers for two important events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
};
```

The triggers used at the end of the function can be found by watching the messages you see on your screen when playing (read more about triggers on the Wiki). Most maps have far more triggers than two, of course, but let's focus on these two for the moment.

**The Map table.** The Map.FunctionName syntax in the last two lines of the OnMapLoad function tells us that the callback functions set up there can be found in the Map table. Thus, in addition to our OnMapLoad function, we need at least a Map table in our script. (For those who have a basic knowledge of programming, a GameMonkey table is pretty much the same thing as an array in most other languages. For those who haven't, just accept the syntax for the moment, you'll gain a better understanding of what it's all about along the way.) So, assuming we have two triggers, we should add this to our GM script:

```
global Map =
{
    KeyStolen = function()
```

```

{
},

KeyReturned = function()
{

},

};

```

Let's assume we have only two major events in our map. When the attacking team (ALLIES) has stolen the safe key, the defending team (AXIS) should focus on defending the safe that the attacking team will try to open with the key. If the key is returned, they should focus on the initial defend spots again:

```

global Map =
{
  KeyStolen = function()
  {
    //Disable phase 1 goals:
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, false, "MAP_SNIPER_SPOT_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_SNIPER_SPOT_phase_1_allies.*");

    //Enable phase 2 goals:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_2_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_2_allies.*");
  },

  KeyReturned = function()
  {
    //Disable phase 2 goals when the key is returned:
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_2_axis.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_2_allies.*");

    // ... and reenables phase 1 goals:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "MAP_SNIPER_SPOT_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_SNIPER_SPOT_phase_1_allies.*");
  },
};

```

**Disabling and enabling cappoint goals.** If these two events were all that happened on the map, we'd be pretty much done now. But there is one more thing we are going to cover here: the use of cappoints (capture points). One interesting feature of the Alpine Assault map is the existence of two items that must be captured, and two different capture points, one for each item. The second item (a briefcase with documents) is stored in a safe, and becomes available only after the safe has been opened with a key (the first item). Thus the safe is the capture point for the first item, while the second item must be



brought to the halftrack (a vehicle at the Allies' first spawn point).

If you type `/bot show_goals "MAP_.*"` into the game console, you'll see some output similar to the following:

```
— Goal List —
1: MAP_CAPPOINT_halftrack -> 1100 serial 2 bias 1.00
2: MAP_FLAG_key -> 0100 serial 3 bias 1.00
3: MAP_CAPPOINT_safe -> 1100 serial 1 bias 1.00
4: MAP_FLAG_case -> 0100 serial 4 bias 1.00
— End Goal List —
```

The ones and zeros after the `"->"` are bits that indicate the availability for each team, in numerical order. That means for example that the goal `MAP_CAPPOINT_halftrack` is available for team 1 and 2, while `MAP_FLAG_key` is only available for Team 2 (in Enemy Territory, we can ignore whatever is said about the other teams). So we see that both cappoint goals are available at the moment, which is not what we want, because the bot that captures the key from the desk has no clue where to run with it.

That means we should disable the halftrack cappoint at the beginning; and we will need one more event that enables it and disables the safe as a cappoint. So let's update our `OnMapLoad` function:

```
global OnMapLoad = function()
{
    //Disable all sniper, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "MAP_SNIPER_SPOT_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_SNIPER_SPOT_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "MAP_SNIPER_SPOT_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_SNIPER_SPOT_phase_1_allies.*");

    //Disable unusable capture objectives and capture spots:
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_FLAG_case");

    //Set up triggers for three events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
    OnTrigger("Allied team has opened the Safe!", Map.SafeOpened);
};
```

We add one more function to the Map table, so the second cappoint becomes available when the safe is open:

```

SafeOpened = function()
{
    //Enable new CAPPOINT and FLAG goals ...
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_FLAG_case");

    // ... and disable the obsolete ones:
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_CAPPOINT_safe");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_FLAG_key");
},

```

Putting all this together, we finally end up with the following (simplified) map script:

```

//=====
// Map script for Alpine Assault by d00d
// Works with Omni-bot 0.65
// Last update: Thu, 2 August 2007 15:07:38 GMT
//=====

global Map =
{
    KeyStolen = function()
    {
        //Disable phase 1 goals:
        SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_1_axis.*");
        SetAvailableMapGoals(Team.AXIS, false, "MAP_SNIPER_SPOT_phase_1_axis.*");
        SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_1_allies.*");
        SetAvailableMapGoals(Team.ALLIES, false, "MAP_SNIPER_SPOT_phase_1_allies.*");

        //Enable phase 2 goals:
        SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_2_axis.*");
        SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_2_allies.*");
    },

    KeyReturned = function()
    {
        //Disable phase 2 goals when the key is returned:
        SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase_2_axis.*");
        SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase_2_allies.*");

        // ... and reenale phase 1 goals:
        SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
        SetAvailableMapGoals(Team.AXIS, true, "MAP_SNIPER_SPOT_phase_1_axis.*");
        SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
        SetAvailableMapGoals(Team.ALLIES, true, "MAP_SNIPER_SPOT_phase_1_allies.*");
    },

    SafeOpened = function()

```

```

{
    //Enable new CAPPOINT and FLAG goals ...
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_FLAG_case");

    // ... and disable the obsolete ones:
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_CAPPOINT_safe");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_FLAG_key");
},
};

global OnMapLoad = function()
{
    //Disable all sniper, defend and attack goals for both teams:
    SetAvailableMapGoals(Team.AXIS, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.AXIS, false, "MAP_SNIPER_SPOT_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "ATTACK_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "DEFEND_phase.*");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_SNIPER_SPOT_phase.*");

    //Selectively enable phase 1 goals on a per team basis:
    SetAvailableMapGoals(Team.AXIS, true, "DEFEND_phase_1_axis.*");
    SetAvailableMapGoals(Team.AXIS, true, "MAP_SNIPER_SPOT_phase_1_axis.*");
    SetAvailableMapGoals(Team.ALLIES, true, "ATTACK_phase_1_allies.*");
    SetAvailableMapGoals(Team.ALLIES, true, "MAP_SNIPER_SPOT_phase_1_allies.*");

    //Disable unusable capture objectives and capture spots:
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_CAPPOINT_halftrack");
    SetAvailableMapGoals(Team.ALLIES, false, "MAP_FLAG_case");

    //Set up triggers for three events:
    OnTrigger("Allied team has stolen the Safe Keys!", Map.KeyStolen);
    OnTrigger("Axis team has returned the Safe Keys!", Map.KeyReturned);
    OnTrigger("Allied team has opened the Safe!", Map.SafeOpened);
};

```

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ET\\_Waypointing\\_Tutorial/Archives/066](http://omni-bot.com/wiki/index.php?title=ET_Waypointing_Tutorial/Archives/066)"

This page has been accessed 77 times. This page was last modified 05:15, 1 May 2008.

# ETPub: Kicking all bots

(Redirected from User:Papagali)

## Kicking Bots in ETPub

ETPub has a separate cvar setting for bot presence on the server: `g_bot_minPlayers`. If this cvar is not set to 0 in your `etpub.cfg`, kicking bots will always lead to adding new bots after the kick.

### Settings to kick bots

Make sure not to set `MinBots` and `MaxBots` in your `et_autoexec.gm`.

- Add **nobot.cfg** to your `etpub` folder on your server

#### **nobot.cfg**

```
//Nobot.cfg
MinBots(0);
MaxBots(0);
kickall;
set g_bot_minPlayers "0"
```

- Add **!nobot** command to your `shrubbot.cfg`

```
[command]
command = nobot
exec = exec nobot.cfg
desc = Kicks all bots
levels = 5
```

### Settings to re-add bots

- Add **bot.cfg** to your `etpub` folder on your server

#### **bot.cfg**

```
//Bot.cfg
MinBots(0);
MaxBots(X);
```

```
set g_bot_minPlayers "X"
```

- Add **!bot** command to your shrubbot.cfg

**[command]**

**command** = bot

**exec** = exec bot.cfg

**desc** = Adds bots to total the number of **X** players

**levels** = 5

*Where **X** is the number of players on your server, to which total you want bots added.*

Papagali 10:50, 23 January 2008 (MST)

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=ETPub:\\_Kicking\\_all\\_bots](http://omni-bot.com/wiki/index.php?title=ETPub:_Kicking_all_bots)"

This page has been accessed 162 times. This page was last modified 17:50, 23 January 2008.

# Useswitch

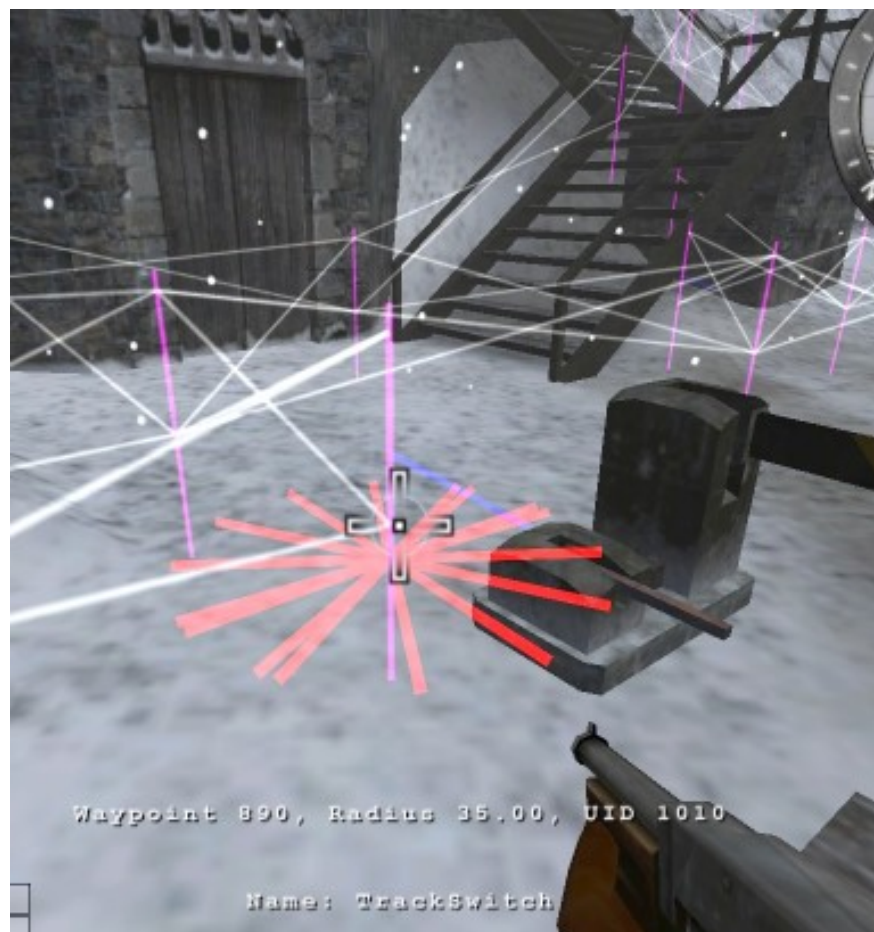
From Omni-bot Wiki

The use\_switch scripted goal is ideally used in maps where the switch has a usable trigger as it relies on setting the availability of the switch. If there is no trigger and the switch is used to control a mover such as an elevator, you can possibly use a thread to enable/disable the switch goal. This scripted goal requires setup in script and in the waypoint file:

- Step 1: Name the waypoint closest to the switch in the waypoint file with the /bot waypoint\_setname command. For example purposes, say we named it lever1.

Make sure the waypoint's facing is set so that the bot will turn to the correct position when using the switch. The easiest way to make sure the facing is correct is to go to the switch and use it yourself, then add a waypoint without changing your position. The waypoint's facing will automatically be set to your current facing. If you want to use an existing waypoint, stand over the waypoint, turn to a position in which you can use the switch, then do /bot waypoint\_setfacing. You may have to experiment with this.

The following images show the waypoint at the switch in railgun with viewfacing turned on as an example. The facing is indicated by the blue line:





- Step 2: Create a Switches table inside the Map table of your map script (create a Map table if you don't have one):

```
global Map =
{
    Switches =
    {
    },
};
```

- Step 3: Create a table for your switch inside the Switches table:

```
global Map =
{
    Switches =
    {
        MySwitch =
        {
        },
    },
};
```

- Step 4: Set the WaypointName property for the switch:

```
global Map =
```

```
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
        },
    },
};
```

- Step 5: Set the LimitTeam property for the switch:

```
global Map =
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
        },
    },
};
```

- Step 6: Set the LimitClass property for the switch:

```
global Map =
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
        },
    },
};
```

- Step 7: Update the switch table. This step is the step that processes the information and makes the switch usable

```
global OnMapLoad = function()
{
    Util.UpdateSwitchData();
};
```

- Step 8: Set up triggers for making the switch available / unavailable

```
global Map =
{
    Switches =
```



```

{
    MySwitch =
    {
        WaypointName = "lever1",
        LimitTeam = (1<<TEAM.AXIS), //only axis can use it
        LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
    },
},

door_open = function( trigger )
{
    Map.Switches.MySwitch.LimitTeam = (1<<TEAM.AXIS);
},

door_closed = function( trigger )
{
    Map.Switches.MySwitch.LimitTeam = (1<<TEAM.ALLIES);
},
};

global OnMapLoad = function()
{
    OnTrigger( "The Door is Opening!", Map.door_open );
    OnTrigger( "The Door is Closing!", Map.door_closed );

    Util.UpdateSwitchData();
};

```

In cases where the levers do not have a wm\_announce message associated, you can use the TagName given with the /bot debugtriggers command.

All is set up. At this point the axis and allies engineers will be fighting over keeping the door open or closed in this example.

Valid LimitTeam values are:

```

0 //no team can use it. use this to turn it off
(1<<TEAM.AXIS)
(1<<TEAM.ALLIES)

```

Valid LimitClass values are:

```

(1<<CLASS.SOLDIER)
(1<<CLASS.FIELDOPS)
(1<<CLASS.ENGINEER)
(1<<CLASS.SOLDIER)
(1<<CLASS.MEDIC)

```

Setting up for multiple classes is done like this:

```

global Map =

```

```

{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            LimitClass = (1<<CLASS.ENGINEER) | (1<<CLASS.SOLDIER), //soldiers and engineers
can use
        },
    },
};

```

## Optional Settings

### LimitBots

The number of bots considering a switch for usage can be limited further by setting the LimitBots property in the switch table. By default, this setting is 64 and it is not required. To Limit the total number of bots that will use a switch, simply add the property to the switch table:

```

global Map =
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
            LimitBots = 2, //only two bots at a time
        },
    },
};

```

### LimitDistance

By default, bots will consider using a switch at all distances. In some cases you may want them to only consider using the switch if they are within a specific range:

```

global Map =
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
            LimitDistance = 200, //only bots within 200 units can use

```

```

    },
  },
};

```

note: you may consider using this in conjunction with routing to get the bots close enough to use the switch while heading to a specific goal.

## ExitConditions

In cases where a switch or button may not have a trigger, Exit condition functions can be defined to help with setting availability of the switch / button. The functions must return either true or false. If one of the functions returns true, the exit condition will be met:

```

global Map =
{
  Switches =
  {
    MySwitch =
    {
      WaypointName = "lever1",
      LimitTeam = (1<<TEAM.AXIS), //only axis can use it
      LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
      ExitConditions =
      {
        someFunc = function(bot)
        {
          return true; //as soon as they hit switch, return true
        },
      },
    },
  },
};

```

Note that 'bot' is passed to this function so you can run tests against the individual bot that has used the switch before returning true or false.

## PressOnce

This property **must** be set in conjunction with ExitConditions and will limit the bots usage of a switch to once per spawn. Useful for cases where an exit condition may not set global availability for a switch:

```

global Map =
{
  Switches =
  {
    MySwitch =
    {
      WaypointName = "lever1",
      LimitTeam = (1<<TEAM.AXIS), //only axis can use it

```

```

LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
PressOnce = true,
ExitConditions =
{
    someFunc = function(bot)
    {
        return true; //as soon as they hit switch, return true
    },
},
},
};

```

## Debug

Setting Debug to true in the switch table will give output to console about the bot with the switch as a goal. For instance if LimitDistance is set, Debug information will include the bots current distance from the switch as it considers whether or not to use it. It will also give an indication when an ExitCondition has been met, whether or not they have already used the switch (if PressOnce has been set), and when the bot is heading for the switch:

```

global Map =
{
    Switches =
    {
        MySwitch =
        {
            WaypointName = "lever1",
            LimitTeam = (1<<TEAM.AXIS), //only axis can use it
            LimitClass = (1<<CLASS.ENGINEER), //only engineers can use it
            Debug = true,
            PressOnce = true,
            ExitConditions =
            {
                someFunc = function(bot)
                {
                    return true; //as soon as they hit switch, return true
                },
            },
        },
    },
};

```

## Case Study: Using debugtriggers to determine direction of a door

The levers in Battery represent what may be a common implementation of the useswitch goal when levers do not have a wm\_announce trigger. The front door has one lever that triggers two doors. To find the triggers for use with the useswitch goal, /bot debugtriggers is used. Once the lever is used, console output for all three movers is listed:

```
<---> Trigger: TagName: frontdoor_lever1_goto Action: 0.00 0.00 -16.00 Entity: 23A6A364
Activator: 00000000
<---> Trigger: TagName: frontdoor_right_goto Action: 0.00 18.00 0.00 Entity: 23A747EC
Activator: 00000000
<---> Trigger: TagName: frontdoor_left_goto Action: 0.00 -15.00 0.00 Entity: 23A74270
Activator: 00000000
```

The Action listed is a string representing the velocity of each mover. It is important to note any non zero values in the string as they will help determine the direction of the mover. It is only necessary to use one door for the trigger for a map script. In this case, the right door is opening when the Action has a y value of 18. When closing the doors, the y value of the Action is -18. These values can be checked in map script to determine direction of the door to set availability of the switch:

```
global Map =
{
    frontdoor_trigger = function( trigger )
    {
        //turn the trigger action string into a vector for referencing
        vel = ToVector(trigger.Action);

        //reference the changing value to determine door direction
        if ( vel[1] > 0 )
        {
            //doors opening
        }
        else if ( vel[1] < 0 )
        {
            //doors closing
        }
    }
};
```

For a complete example of this setup, look in the battery.gm released with version 0.65.

*note: ToVector is a new function with version 0.65.*

---

Retrieved from "<http://omni-bot.com/wiki/index.php?title=Useswitch>"

This page has been accessed 365 times. This page was last modified 18:35, 4 June 2008.

# WayPoint DataBase WPDB - FAQ

From Omni-bot Wiki

## Contents

- 1 WayPoints DataBase WPDB - FAQ
  - 1.1 How to use WPDB?
  - 1.2 Where to put downloaded files?
  - 1.3 How to contribute waypoints?
  - 1.4 How to upload files?
  - 1.5 I get an upload error !?!
  - 1.6 How to update files?

## WayPoints DataBase WPDB - FAQ

Currently the database and this page is still in development, but it's time to do some notes :) Feel free to extend page ...

### How to use WPDB?

- browse
- search
- download

### Where to put downloaded files?

- path - extract zips or not

### How to contribute waypoints?

Send a personal message to Dr.Evil or Magik and ask for access.

### How to upload files?

- wps
- images

## **I get an upload error !?!**

- File is not a valid zip file. Make sure you chose the right file to upload.
- Filesize limit is reached. Current limit is 100kb. If you reach this filesize contact an admin.
- Check your upload zip file. Content is invalid! Your zip file contains invalid files. Do not put any other files than one \*.way and/or one \*.gm file into your upload zip file.
- No wayfile found in zip file! Just in case you forget to put your waypoint file into your zip ;)

## **How to update files?**

- explain version-counter

- updating form-fields in 'manage-waypoints' is not an update!

---

Retrieved from "[http://omni-bot.com/wiki/index.php?title=WayPoint\\_DataBase\\_WPDB\\_-\\_FAQ](http://omni-bot.com/wiki/index.php?title=WayPoint_DataBase_WPDB_-_FAQ)"

This page has been accessed 172 times. This page was last modified 16:30, 8 June 2008.