Main Page

From Omni-bot Wiki

Welcome to Omni-bot Wiki.

Contents

- 1 About Omni-bot
- 2 Getting started
- 3 Customizing Omni-bot
- 4 Reference

About Omni-bot

- Intro to Omni-bot - General information on Omni-bot development, goals, and developers.
- Omni-bot Releases - Change lists for all Omni-bot releases.
- Supported Games - Which games are supported?

Getting started

- Installing Omni-bot - Information on installing Omni-bot to your computer or server.
- Configuring Omni-bot - How to configure Omni-bot options.

Customizing Omni-bot

- Omni-bot Waypointing - How to create and edit waypoints for Omni-bot.
- Omni-bot Map Scripting - How to create and edit map scripts for Omni-bot.
- Omni-bot Scripting - General Scripting for Omni-bot.
- Omni-bot Weapon Scripting - Creating weapon scripts.
- Omni-bot Routing - How to set up multiple paths to goals.
- Omni-bot Script Goals - Drop in scripted goals for additional bot functionality.

Reference

- Omni-bot Command Reference - Bot commands typically run from the game console.
- Omni-bot Script Reference - Reference to scripting functions, constants, etc.
- Omni-bot F.A.Q. - Frequently asked questions about Omni-bot.
- General Bot F.A.Q. - Frequently asked questions about bots in general.

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.

Omni-bot Releases

From Omni-bot Wiki

Omni-bot Releases

- Omni-bot 0.7 - *Planning Stage*
- Omni-bot 0.65 - *Not yet released*
- 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

Supported Games

From Omni-bot Wiki

Contents

- 1 Enemy Territory
- 2 Fortress Forever
- 3 Quake 4
- 4 Doom 3

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
- jaymod
- noquarter

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.

Installing Omni-bot

From Omni-bot Wiki

Contents

- 1 How to install Omni-bot on Windows
- 2 How to install Omni-bot on Linux
- 3 How to install Omni-bot on 3rd Party Host
- 4 Introduction : About Gamehosts and Rented Servers
- 5 Installing on a Windows Rented Server

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.

How to install Omni-bot on Linux

TODO:

Dont forget: - ET-instance needs write permission on waypoint-folder to create vis-files

How to install Omni-bot on 3rd Party Host

Introduction : About Gamehosts and Rented Servers

Before we start explaining how to install omni-bot let us have a closer look at the different types of gameservers that are available for rental.

Most of the servers you can rent are running Linux BSD but some companies offer servers that are running windows 2003 Server edition.

The reason why most gameserver hosts use Linux is because you can stack more gameservers on a Linux box.

While Windows servers allow for fewer gameservers on the same machine in general they offer a better quality of servers.

Better quality usually means higher prices but there are exceptions.

So before you go out and rent a server it would be recommendable to check if the gamehost allows for running bots on their servers.

FPS servers for example do not allow for running any kind of bots.

Most gamehosts use a FUP (Fair Use Policy) and if you keep stuff within normal limits there should be nothing preventing you from running an omni-bot server.

An example of a gamehost that rents Windows 2003 Servers is XXL-Servers located in Holland.

Most of the other gamehosts I rented servers from offer only Linux servers.

Installing on a Windows Rented Server

So how do you go about installing omni-bot to a windows gameserver then ?

The best way is to download and run the omni-bot installer on your local PC and install everything in a local directory.

Let's take this step by step.

1. Download the Omni-bot Windows Installer from the Omni-bot website [2]

2. Run the installer by double clicking on it. [3]

3. Pick the options you want install and click next [4]

4. Install the files in a directory "Bot".

This will have 2 subfolders namely "omnibot" and "Omni-bot"

Look at the screenshot [5] and [6] to see what you should fill out in the box. (C:\BOT\omnibot in the first dialog and C:\BOT\Omni-Bot in the second)

5. After the installer finished you should have a folder C:\BOT

with 2 subfolders like in this screenshot [7]

6. After configuring omnibots to your liking (see section configuring OB) Upload the folders Omni-bot and omnibot to your gameserver using an FTP client or the upload facility of the gamepanel the gamehost gave you. The 2 executables Uninstall.exe and GmDebugger.exe will most likely be refused for upload by the gamehost for security reasons. Luckyily for us we won't be needing those files to run omnibots on our gameserver. This screenshot show where you should upload your folders [8]

In our example we are going to run Omni-bots on ETMAIN (standard game). If you are planning on running OB with other mods than ETMAIN you won't be needing the folder "omnibot". You should upload the modversion that has integrated Omni-bot support and the folder "Omni-bot" to make it work. How to do this will be covered in a seperate article. We now have our folders in place and are ready to configure the gameserver so it will run the bots. To run OB in ETMAIN we need to alter the commandline of the gameserver so it will run the mod omnibot in combination with etmain. How to alter your startup commandline on the rented server depends on the gamehost you are renting the server from. A widely spread application to control your gameserver is the so-called "Gamepanel". This is a menu driven system that enables you to start or stop your gamesever and allow for FTP access and direct file access to the game server files. Here is a screenshot of how the application "Gamepanel" looks like [9] Before we continue you should Stop the Gameserver. If the server keeps running and you alter the commandline the server could end up in a deadspin so better avoiding that and stop the gameserver beforehand. In the "command line Changer" utility you can add startup parameters to the commandline the gameserver will boot up with. This is similar to adding parameters in the desktop icon you start an Omnibot server on your local PC. To enable omnibots we need to add "+set fs_game omnibot" to the command line. Click on Commandlinechanger and you will have a button "NEW". Click on that and you get a screen where you can fill alter "+set fs_game" into "omnibot". Name your commandline "omnibot" and save it. Then select the newly created commandline as active. Then go back to the server start/stop menu and start the server. Go to the server and put on the map Oasis since that has bot support. If you did not edit the defbot.gm to use minbots and maxbots you need to add the bots manually or by exuting a cfg that adds bots to your game. How to do this is explained elsewhere in this wiki.

Omni-bot Waypointing

From Omni-bot Wiki

Contents

1 What are waypoints?
2 Importance Waypoint Types
3 Waypointing in IT
4 Backwards Compatibility
5 How do bots use waypoints?
6 Waypoint Visual Aids
7 Workable Waypoints
8 Special Case Waypointing
9 Waypoint Commands
10 Waypoint Flags
11 Waypoint Functions
12 Waypoint Events
13 Waypoint Properties

What are waypoints?

Waypoints are the nodes that the bots will traverse that is set up to form the basic fabrics bots use to navigate a map. Waypoints are also used as a guide for bots to perform certain actions, such as jumping, crouching, or using a ladder.

Waypoints are basically a graph representation of the map geometry. Waypoints are obvious, and quite tedious to place. They are the bane of the existence of any bot author. Fortunately, the waypoint graph is used by the bots.

Waypoints are used by a .way file in Omni-bot.

Important Waypointing Notes

Mainly Territory Use as a major method of breaking waypoints. At first, less and denser (clutter).

Use Flags Use flags to tell the bots what to do at a waypoint. Flags are important. Use them. Flags tell the bots what to do at a waypoint. Flags can be used for anything (crouching, jumping, ladder, sniper spot, etc.). Flags are important. Use them. Flags are important. Use them. Flags are important. Use them.

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 seperate 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

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 seperate 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 fasted 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 occurance I will likely add a property for it.

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\etl\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);
};
```

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 inside 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_routename" );
};

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

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 may offer different challenges. Please post in the forums if you have questions / problems / bug reports.

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 name 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

 useage: 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

 useage: 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>
 example: bot show_goals DEFEND.* will list all defend goals in console

 notes: this command supports expressions for the optional parameter
```

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

Contents

- 1 Global Script Constants
- 2 Global Script Functions
- 3 Scripting Types
  - 3.1 AABB
  - 3.2 Bot
  - 3.3 Blackboard
  - 3.4 MapGoal
  - 3.5 Matrix3
  - 3.6 ScriptGoal
  - 3.7 TargetInfo
  - 3.8 Timer
  - 3.9 TriggerInfo
  - 3.10 Vector3
  - 3.11 Weapon

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

Omni-bot F.A.Q.

From Omni-bot Wiki

How do I customize the names the bots use when they join?

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.

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.

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.

Omni-bot 0.7

From Omni-bot Wiki

Contents

- 1 Omni-bot 0.7 Plans
  - 1.1 All New Goal System
  - 1.2 New Navigation System(maybe)
- 2 Omni-bot 0.7 Changelog

Omni-bot 0.7 Plans

All New Goal System

- Overhauled goal system, based on Halo2 Hierarchical Behavior Tree Gamasutra Halo2 AI Article
- All new, much better system of drop-in script goals. Drop a script in a folder and go, like weapon scripts

New Navigation System(maybe)

- Navigation mesh or an overhauled waypoint system

Omni-bot 0.7 Changelog

Anything that gets done enough to mention at this point is going in 0.65

This section to be updated after 0.65 release.

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 seperate 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.

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.

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.

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 dwon
- 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!

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.

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

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

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

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.

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.

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.

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

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 agains 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 recieves 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 represends 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.

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, CARRYINGOAL, 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 gameId.
- 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 gameId 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.

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.

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.

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.

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.

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

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

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.

Omni-bot 0.2

From Omni-bot Wiki

Omni-Bot 0.2

First release, more or less representing the ETF implementation of Omni-bot.

Map Scripting Enemy Territory

Retrieved from "Enemy Territory: Map Scripting"

Contents
1 Standard Tutorial
1.1 Introduction
1.2 Preparation
1.3 Getting Started
1.4 Triggers
1.4.1 Basic Functions/Trigger Triggers
2 Useful Functions
2.1 Introduction/Overview
2.2 Introduction/Vectors
2.3 Introduction/Entities
2.4 Introduction/Values
2.5 Introduction/Strings
2.6 Introduction/Math
2.7 Introduction/Debugging
2.8 Introduction/Misc
3 Useful Functions - Detailed
3.1 Entities
3.1.1 Entity Identification
3.1.2 Entity Identification/Searching
3.1.3 Entity Identification/Iteration
3.1.4 Entity Location
3.1.5 Entity Movement
3.1.6 Entity Movement/Alignment
3.1.7 Entity Movement/Rotation
3.2 Values
3.2.1 Values - Miscellaneous
3.2.2 Values/Table Handling
3.2.3 Values/Table Iteration
3.2.4 Values/Table Sorting
3.2.5 Values/Table Copy
3.2.6 Values/Table Miscellaneous
3.3 Strings
3.3.1 String - Miscellaneous
3.3.2 String/Tokenizing
3.4 Math
3.4.1 Math - Miscellaneous
3.4.2 Math/Vectors
3.5 Debugging
3.5.1 Debugging - Miscellaneous
3.5.2 Debugging/Drawing
3.6 Misc
3.6.1 Misc - Miscellaneous
3.6.2 Misc/Commands
3.6.3 Misc/Cvars
3.6.4 Misc/Events
3.6.5 Misc/Spawning
3.6.6 Misc/Sound
3.6.7 Misc/Trigger
3.6.8 Misc/Weapons
3.6.9 Misc/Wolfcam
3.6.10 Misc/Wolfcam/Demo
3.6.11 Misc/Wolfcam/Hud
3.6.12 Misc/Wolfcam/Hud Elements
3.6.13 Misc/Wolfcam/Hud Properties
3.6.14 Misc/Wolfcam/Hud Text
3.6.15 Misc/Wolfcam/Hud Text Properties
3.6.16 Misc/Wolfcam/Hud Text Alignment
3.6.17 Misc/Wolfcam/Hud Text Flags
4 Conditionals
4.1 Case Study
5 Debugging Tools
5.1 Debugging - Tools
5.2 Debugging - Tips
6 Appendix
6.1 Entity - goals

Standard Format

All official maps mostly share a common layout, having a standard file map entities 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 file map.lua and two functions that are automatically called by ID's Main.lua:

```
global Map =
{
}

global OnMapLoad = function()
end

global OnMapSave = function()
end
```

Map Table

Using OnMapLoad will ensure that a map script is in a standard format.

Map Trigger

The Map Table has become an important part of et ET map script as some scripted goals rely on its existence. It also provides a OOP styled to store map specific information. Typically it will consist of map variables, triggers / trigger functions and OnMapLoad.

```
global Map =
{
    mapname = g_gameinfo.mapname,

    -- Trying to avoid hard coding this was previous messages sent during the game. This must be inserted exactly including capitalization, punctuation, and color codes (if any). To see this one previous message requires the sequence to action play through the map and perform the objectives or issue the /vsay command.
    -- Also this is not required for a map to function but is a nice touch to have.
    -- Also this is not required for a map to function but is a nice touch to have.

    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.
    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.

    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:
    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:

    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.
    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:

    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.
    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:
    -- Also this is not required for a map to function but is a nice touch to have.

    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.
    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:
    -- Also this is not required for a map to function but is a nice touch to have.
    -- Also this is not required for a map to function but is a nice touch to have.

    -- Trigger to store the current objective. Used by the Axis Radio Operator function. The Radio looks script Unit to be continued with the Omni-bot map script) is usually the file | integration script in the directory | Omni-bot\et\scripts\ and is named the same as the map. For example, fueldump.lua.
    -- Another method of finding the string parameter is using the command /bot debuginterpret. The command will output to console everytime a recognized event occurs. The syntax will be similar to:
    -- Also this is not required for a map to function but is a nice touch to have.
```

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, defallt 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
```

AMMO

From Omni-bot Wiki

BB

From Omni-bot Wiki

Common Blackboard Keys

```
BB.DELAY_GOAL
BB.IS_TAKEN
BB.RUN_AWAY
```

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
```

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
```

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
```

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
```

CLASS

From Omni-bot Wiki

== 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.SCOUT
CLASS.SENTRY
CLASS.SNIPER
CLASS.SOLDIER
CLASS.SPY
CLASS.TURRET
```

CONTENT

From Omni-bot Wiki

== Common Content Flags ==

```
CONT.FOG
CONT.LAVA
CONT.MOVER
CONT.SLIME
CONT.SOLID
CONT.TELEPORTER
CONT.TRIGGER
CONT.WATER
```

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
```

ENTFLAG

From Omni-bot Wiki

== 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
```

== 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
```

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 KILLEDSOMEONE
  - 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
- whatitheysaid

GLOBAL_VOICE

Parameters:

- macro id

HEAR_SOUND

Parameters:

- source
- origin
- soundid
- soundname

KILLEDSOMEONE

Parameters:

- victim
- meansofdeath

PLAYER_USE

Parameters:

- whotouchedme

PRIV_CHAT_MSG

Parameters:

- whosaidit
- whatitheysaid

PRIVATE_VOICE

Parameters:

- macro id

REMOVEWEAPON

Parameters:

- weaponid

RESETWEAPONS

Parameters: none

REVIVED

Parameters:

- whorevivedme

TEAM_CHAT_MSG

Parameters:

- whosaidit
- whatitheysaid

TEAM_VOICE

Parameters:

- 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:

- ezTeam
- ezClass

DISGUISING

Parameters:

- ezTeam
- ezClass

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

POWERUP

From Omni-bot Wiki

== 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

PROFILE

From Omni-bot Wiki

== Profile Types ==

```
PROFILE.NONE
PROFILE.CLASS
PROFILE.CUSTOM
```

SKILL

From Omni-bot Wiki

== Enemy Territory Skills ==

BATTLE_SENSE
COVERTOPS
ENGINEERING
HEAVY_WEAPONS
LIGHT_WEAPONS
SIGNALS

TEAM

From Omni-bot Wiki

== 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
```

TRACE

From Omni-bot Wiki

== Common Trace Flags ==

```
TRACE.ALL
TRACE.OPAQUE
TRACE.PLAYER
TRACE.PLAYERCLIP
TRACE.SHOT
TRACE.SOLID
TRACE.SMOKEBOMB
TRACE.WATER
```

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_FIELDOPS
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
```

WEAPON

From Omni-bot Wiki

== 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.PLIERS
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

Bot Library

From Zero Script Wiki

Contents

- 1 Global Bot Functions
  - 1.1 Addition
  - 1.2 CanCreateEntity
  - 1.3 CanCreateObject
  - 1.4 CheckGroupAllowed
  - 1.5 CheckGroupDenied
  - 1.6 CheckGroupMember
  - 1.7 CheckGroupMembership
  - 1.8 Division
  - 1.9 Multiplication
  - 1.10 Random
  - 1.11 Subtraction
  - 1.12 UnixFromDateTime
  - 1.13 UnixFromDateTimeUTC
  - 1.14 UnixNow
  - 1.15 UnixNowUTC
  - 1.16 UnixToDateTime
  - 1.17 UnixToDateTimeUTC
  - 1.18 UnixToDateTimeUTCNow
  - 1.19 UnixToDateTimeUTCNowNow
  - 1.20 UnixToDateTimeUTCNowNowNow
  - 1.21 UnixToDateTimeUTCNowNowNowNow
  - 1.22 UnixToDateTimeUTCNowNowNowNowNow
  - 1.23 UnixToDateTimeUTCNowNowNowNowNowNow
  - 1.24 UnixToDateTimeUTCNowNowNowNowNowNowNow
  - 1.25 UnixToDateTimeUTCNowNowNowNowNowNowNowNow
  - 1.26 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNow
  - 1.27 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNow
  - 1.28 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNow
  - 1.29 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.30 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.31 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.32 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.33 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.34 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.35 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.36 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.37 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.38 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.39 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.40 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.41 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.42 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.43 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.44 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.45 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.46 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.47 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.48 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.49 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.50 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.51 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.52 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.53 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.54 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.55 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.56 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.57 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.58 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.59 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.60 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.61 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.62 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.63 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.64 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.65 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.66 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.67 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.68 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.69 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.70 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.71 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.72 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.73 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.74 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.75 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.76 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.77 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.78 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.79 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.80 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.81 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.82 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.83 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.84 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.85 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.86 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.87 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.88 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.89 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.90 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.91 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.92 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.93 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.94 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.95 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.96 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.97 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.98 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.99 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.100 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.101 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.102 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.103 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.104 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.105 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.106 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.107 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.108 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.109 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.110 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.111 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.112 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.113 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.114 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.115 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.116 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.117 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.118 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.119 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.120 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.121 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.122 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.123 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.124 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.125 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.126 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.127 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.128 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.129 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.130 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.131 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.132 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.133 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.134 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.135 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.136 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.137 UnixToDateTimeUTCNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNowNow
  - 1.138 UnixToDateTimeU

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:

```
var g = GetGoal("MAP_FLAG_CAPTAG");
```

See Also: SetGoal

GetGoals

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

Parameters: (table, team, expression)

Returns: int

Example:

```
local t = {};
local n = GetGoals(t, 0, "MAP_FLAG_.*");
```

See Also: GetGoal

GetGravity

Gets the current gravity of the game.

Parameters: none

Returns: gravity

Example:

```
g/m = GetGravity();
```

GetMapExtents

Gets the 3dfill map extents. Map extents are the bounds of the entire map.

Parameters: (3dfill optional=1)

Returns: if no 3dfill passed as parameter, function will return the map 3dfill, otherwise the 3dfill parameter will be filled in with the results, and the function will return 1

Example:

```
local b = BrushFromVect();
local mins, maxs
GetMapExtents(brushaddr);
```

GetMapName

Gets the name of the current map.

Parameters: none

Returns: name of map

Example:

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

GetMaxPlayers

Gets the current max players supported by the game.

Parameters: none

Returns: max players

Example:

```
if(GetMaxPlayers() < 20)
{
}
```

GetNumPlayers

Gets the current num players in the game.

Parameters: none

Returns: current players

Example:

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

GetNumSkills

Gets the current number of skills.

Parameters: none

Returns: current # skills

Example:

```
if(GetNumSkills() < 20)
{
}
```

GetWorldVersion

Gets the version of the current mod.

Parameters: none

Returns: version of mod

Example:

```
if(GetWorldVersion() < "0100 2.3")
{
}
```

GetWorldVersionInfo

Gets the contents into for a given platform.

Parameters: (world platform)

Returns: contents into. Use tokens manipulators along with constants from the global CONTENT table to check for specific content into.

Example:

```
local contents = GetWorldVersionInfo(WORLD_ALL);
if(contents & CONTENT_WATER)
{
}
```

Kickall

Kicks all bots from the game.

Parameters: none

Returns: null

Example:

```
Kickall();
```

Kickbot

Kicks a bot by name.

Parameters: (name)

Returns: none

Example:

```
Kickbot("Fred");
```

Log

Logs a string to the window log file.

Parameters: (string)

Returns: none

Example:

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

MinBots

Sets the min bots.

Parameters: (# minbots)

Returns: none

Example:

```
MinBots(10);
```

See Also: MaxBots

OnTrigger

Registers a user function callback for a given trigger string.

Parameters: (trigger string, script function)

Returns: none

Example:

```
OnTrigger("Arrive", "ArriveFunc");

ArriveFunc(){ Print("Arrive have been ons the trig dancr Pude."+_myTable); }
```

See Also: Send Map Trigger, Entity Trigger

RegisterDefaultProfile

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

Parameters: (Class #, script name)

Returns: none

Example:

```
RegisterDefaultProfile(CLASS_SOLDIER, "DefSoldier");
```

ServerCommand

Executes a server command. Requires cheats to be enabled.

Parameters: none

Returns: true if cheats are enabled, false if not

Example:

```
ServerCommand("bot spin");
```

See Also: ClientCommand

SetAvoidAllWaypoints

Enables/disables the avoidance state of one or more map goals that match a regular expression. Or a specific team.

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

Returns: none

Example:

```
SetAvoidAllWaypoints( TEAM_ALIEN, false, "Attack.*" );
```

See Also: GetGoal, SetGoals

SetBotSkill

Sets the bot skill of one or more map goals that match a regular expression. Or a specific team.

Parameters: (bot, waypoints)

Returns: none

Example:

```
SetBotSkill(1.0, "Attack.*");
```

See Also: GetGoal, SetGoals

ShowPaths

Debug information function. Prints Entities names, classes, and revision data, along with the current paths.

Parameters: none

Returns: null

Example:

```
ShowPaths();
```

TraceLine

Performs a baseline 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: (Source vect, Source end, table, collision mask, ignore user (current), use TYC)

- Source start - Start position of trace line.
- Source end - End position of trace line.
- Table - Result table to store the trace results. [optional]
- Collision mask - Mask of what types to test for. [see TRACE table]
- Ignore user - Ignore this Gamestate in the traceline. Usually the entity to test a trigger/area from.
- Use TYC - Use TYC collision data. [optional]

TraceLines test for a SINGLE user called collision intersection from the set of Tracelines. EachTYC-Mask is used when all you want to know is whether or not something is touching the path from start to end. It will not give you the collision position. It is lightly faster in most cases to use TYC for an yes/no test. Set debug to show the results of the traceline. Green means no collision, red means collision. Blue means that the traceline was not tested against anything.

Returns: true if collision occurred, false if not

Results table contains the following information.

- fraction - The percentage from the start to the collision. Not always valid if startPt=true. Example: 0.5 means say the half way between the start and end position.
- startPt - The starting point of the collision. Not always valid if startPt=true.
- endPt - The ending point of the collision.

Additional properties are available if startPt=true and there was a collision.

- entity - Key for any Gamestate that was hit in the collision.
- normal - The normal of the collision surface.
- dist - The distance traveled passed of the collision.

Example:

```
// Do a traceline using a fancy TraceLog color effect set
local t = {};
local start = GetPlayerPos(0);
local end = start + Vector(0, 0, 100);
if(TraceLine(start, end, t, TRACE_WORLD | TRACE_ENTITY, 0, 0, 0, 0, 1, 0, 0, 0, 0
```

Math Library

From Omni-bot Wiki

Basic Math Functions

- Cos(x) - Cosine.
- ACos(x) - Arc Cosine.
- Sin(x) - Sin.
- ASin(x) - Arc Sin.
- Tan(x) - Tangent.
- ATan(x) - Arc Tangent.
- Ceil(x) - Smallest integer >= x.
- Floor(x) - Largest integer <= x.
- Round(x) - Rounds the number to the nearest integer whole number.
- Abs(x) - Absolute value of x.
- Clamp(x, min, max) - Clamps x to a minimum and maximum value.
- DegToRad(x) - Converts degrees to radians.
- RadToDeg(x) - Converts radians to degrees.
- Min(x, y) - Returns the minimum value of the 2 parameters.
- Max(x, y) - Returns the maximum value of the 2 parameters.
- RandInt(x, y) - Returns a random integer between x and y.
- RandFloat(x, y) - Returns a random float between x and y.
- Sign(x) - Returns a signed scalar of x. Negative numbers returns -1, positive numbers return 1.
- Sqrt(x) - Square root of x.
- SymmetricRandom() - Returns random number between -1.0 and 1.0
- UnitRandom() - Returns random number between 0.0 and 1.0
- ToFloat(x) - Converts an integer or string to a float if possible.
- ToInt(x) - Converts a float or string to an integer if possible.
- ToVector(x) - Converts a string to a Vector3 if possible.

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();
```

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);
```

Bot

From Line Sim Wiki

Contents

1 Bot Properties
1.1 AllowHacking
1.2 AllowPickaxing
1.3 AllowSpeech
1.4 AllowTrading
1.5 Area
1.6 Attachment
1.7 Attack
1.8 AttackCooldown
1.9 AttackDamageMultiplier
1.10 AttackRange
1.11 Axe
1.12 AxeOnTree
1.13 BotAttackAnimationSpeed
1.14 BotAttackAnimationTrack
2 Bot Functions
2.1 AddBlockedThread
2.2 AddBlockedThreadWithTimeout
2.3 ChangeClass
2.4 ChangeClassSilent
2.5 ChangeTool
2.6 Crash
2.7 CreateHacking
2.8 CreateHackingSilent
2.9 CreateHacking2
2.10 CreateTrade
2.11 DeTool
2.12 DisableAllAttachments
2.13 DisableAllAttachmentsExcept
2.14 DisableAttachment
2.15 DisableAttachments
2.16 DisableHacking
2.17 DisableTrade
2.18 EnableAllAttachments
2.19 EnableAttachment
2.20 EnableAttachments
2.21 EnableHacking
2.22 EnableTrade
2.23 EquipTool
2.24 FireWeapon
2.25 GetAllAttachments
2.26 GetAttachment
2.27 GetClass
2.28 GetClassName
2.29 GetHackingThread
2.30 GetHackingThreadWithTimeout
2.31 GetHumanoid
2.32 GetHumanoidRootPart
2.33 GetTradingThread
2.34 GetTradingThreadWithTimeout
2.35 GetTool
2.36 GetTrade
2.37 GetUser
2.38 GetUserId
2.39 GetWeapon
2.40 GetWeaponClass
2.41 GetWeaponObject
2.42 GetWeaponObjects
2.43 GetWeaponObjectsWithClass
2.44 GetWeaponWithClass
2.45 GetWeapons
2.46 GetWeaponsWithClass
2.47 HasAttachment
2.48 HasAttachments
2.49 HasTool
2.50 HasWeapon
2.51 HasWeaponClass
2.52 HasWeaponObject
2.53 HasWeaponObjectClass
2.54 HasWeaponObjectClasses
2.55 HasWeaponObjects
2.56 HasWeaponObjectsClasses
2.57 HasWeapons
2.58 HasWeaponsClasses
2.59 Kill
2.60 LookAt
2.61 LookAtTarget
2.62 LookAtTargetSilent
2.63 LookAtTargetSilent2
2.64 LookAtTargetSilent3
2.65 LookAtTargetSilent4
2.66 LookAtTargetSilent5
2.67 LookAtTargetSilent6
2.68 LookAtTargetSilent7
2.69 LookAtTargetSilent8
2.70 LookAtTargetSilent9
2.71 LookAtTargetSilent10
2.72 LookAtTargetSilent11
2.73 LookAtTargetSilent12
2.74 LookAtTargetSilent13
2.75 LookAtTargetSilent14
2.76 LookAtTargetSilent15
2.77 LookAtTargetSilent16
2.78 LookAtTargetSilent17
2.79 LookAtTargetSilent18
2.80 LookAtTargetSilent19
2.81 LookAtTargetSilent20
2.82 LookAtTargetSilent21
2.83 LookAtTargetSilent22
2.84 LookAtTargetSilent23
2.85 LookAtTargetSilent24
2.86 LookAtTargetSilent25
2.87 LookAtTargetSilent26
2.88 LookAtTargetSilent27
2.89 LookAtTargetSilent28
2.90 LookAtTargetSilent29
2.91 LookAtTargetSilent30
2.92 LookAtTargetSilent31
2.93 LookAtTargetSilent32
2.94 LookAtTargetSilent33
2.95 LookAtTargetSilent34
2.96 LookAtTargetSilent35
2.97 LookAtTargetSilent36
2.98 LookAtTargetSilent37
2.99 LookAtTargetSilent38
2.100 LookAtTargetSilent39
2.101 LookAtTargetSilent40
2.102 LookAtTargetSilent41
2.103 LookAtTargetSilent42
2.104 LookAtTargetSilent43
2.105 LookAtTargetSilent44
2.106 LookAtTargetSilent45
2.107 LookAtTargetSilent46
2.108 LookAtTargetSilent47
2.109 LookAtTargetSilent48
2.110 LookAtTargetSilent49
2.111 LookAtTargetSilent50
2.112 LookAtTargetSilent51
2.113 LookAtTargetSilent52
2.114 LookAtTargetSilent53
2.115 LookAtTargetSilent54
2.116 LookAtTargetSilent55
2.117 LookAtTargetSilent56
2.118 LookAtTargetSilent57
2.119

```
local Vector3 = Vec3
local Vector3 = Vec3(0,0,0)
local Vector3 = Vec3(0,0,0):Normalized()
```

HasWeapon

Checks if the bot has a specific weapon.

Parameters: (weapon)

Returns: true if the bot has it, false if not

Example:

```
if bot:HasWeapon(WEAPON_SHOTGUN) then
```

See Also: GetWeapons

HasWeapons

Checks if the bot has a specific set of weapons.

Parameters: (weapons)

Returns: true if the bot has it, false if not

Example:

```
if bot:HasWeapons(WEAPON_SHOTGUN) then
```

See Also: GetWeapons

HoldButton

Makes the bot press a button, and hold it for an amount of time. The function can take any number of buttons as arguments, and will hold them all for the same time. The time period is **RANDOM** for each activation.

Parameters: (button1, ... , time to randomly

Returns: nil

Example:

```
bot:HoldButton(IN_ATTACK)
bot:HoldButton(IN_ATTACK, IN_ATTACK2, 0.5, 1.5)
bot:HoldButton(IN_ATTACK, IN_ATTACK2, 0.5, 1.5, IN_JUMP)
```

See Also: PressButton, ReleaseButton

IgnoreTarget

Tells the bot to ignore this target.

Parameters: (entity)

Returns: nil

Example:

```
bot:IgnoreTarget(Entity(1))
```

See Also: ClearIgnoredTargets

InstallPathFollow

Installs the path following behavior to make the bot move. PathFollow. This function can take an optional field of arguments as a table. If any is provided, it will override the values set in the current PathFollow.

Parameters: (table) (update)

Returns: (table) (update, the object)

Example:

```
bot:InstallPathFollow({ update = true })
```

See Also: RemovePathFollow

IsAttacking

Checks if the bot is attacked with a given entity.

Parameters: (entity)(optional)

Returns: true if it attack, false if not

Example:

```
if bot:IsAttacking(ent) then
```

IsAttackingCurrentWeapons

Checks if the bot is currently flagged as weapon recoil controlled. Script controlled weapons means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (none)

Returns: true if script controlled, false if not

Example:

```
if bot:IsAttackingCurrentWeapons() then
```

See Also: IsScriptControlled

IsAttackingScriptedWeapons

Checks if the bot is currently flagged as weapon recoil controlled. Script controlled weapons means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (none)

Returns: true if script controlled weapons, false if not

Example:

```
if bot:IsAttackingScriptedWeapons() then
```

See Also: IsScriptControlled

IsBlocked

Checks if the bot is currently forward blocked.

Parameters: (none)

Returns: true if it is blocked, false if not

Example:

```
if bot:IsBlocked() then
    -- if yes, then wait for 5 seconds
    bot:HoldButton(0, 5)
end
```

IsCrouchedJumped

Checks if the bot is engaged in a crouch + jump or not. Use this to see if the bot is currently doing this. Crouch + jump is a movement technique, whether that be a gameplay change like Gamma, and etc.

Parameters: (weapon1, function-optional)

Returns: true if it is, false if not

Example:

```
if bot:IsCrouchedJumped(WEAPON_FAMAS, 0) then
```

IsCurrentTargetIn

Checks the bot's current target, if it is in a specific field of view and how far it is away from the bot. Use this to see if the bot is looking at a target, whether that be a gameplay change like Gamma, and etc.

Parameters: (weapon1, function-optional)

Returns: true if it is, false if not

Example:

```
if bot:IsCurrentTargetIn(WEAPON_FAMAS, 0) then
```

IsDead

Checks if the bot is dead.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsDead() then
    -- if yes, then wait for 5 seconds
    bot:HoldButton(0, 5)
end
```

IsDucking

Checks if the bot is ducked.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsDucking() then
    bot:StopDucking()
    bot:StopJumping()
    bot:StopMoving()
    bot:StopAttacking()
    bot:StopReloading()
    bot:StopUsing()
    bot:StopWalking()
    bot:StopZooming()
    bot:StopZoomingSniper()
    bot:StopDuckingJumping()
    bot:StopDuckingJumpingSniper()
    bot:StopDuckingSniper()
    bot:StopWalking()
    bot:StopWeaponRecoil()
    bot:StopWeaponRecoilSniper()
end
```

See Also: IsBlocked, IsJumped

IsJumped

Checks if the bot is jumped.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsJumped() then
    bot:StopJumping()
end
```

IsJumpingCrouched

Checks if the bot is engaged in a jump + crouch or not. Use this to see if the bot is currently doing this. Jump + crouch is a movement technique, whether that be a gameplay change like Gamma, and etc.

Parameters: (weapon1, function-optional)

Returns: true if it is, false if not

Example:

```
if bot:IsJumpingCrouched(WEAPON_FAMAS, 0) then
```

IsLookingAt

Checks if the bot is looking at a specific position.

Parameters: (vector, accuracy)

Returns: true if it is looking at it, false if not

Example:

```
if bot:IsLookingAt(Vector(0,0,0), 0.95) then
```

IsOnLadder

Checks if the bot is on a ladder.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsOnLadder() then
```

IsReloading

Checks if the bot is reloading.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsReloading() then
```

IsScriptControlled

Checks if the bot is currently flagged as script controlled. Script controlled means the bot is legit and will be entirely a help to control the behavior of the bot.

Parameters: (none)

Returns: true if bot controlled, false if not

Example:

```
if bot:IsScriptControlled() then
```

IsScriptControlledWeapons

Checks if the bot is currently flagged as weapon recoil controlled. Script controlled weapons means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (none)

Returns: true if script controlled weapons, false if not

Example:

```
if bot:IsScriptControlledWeapons() then
```

See Also: IsScriptControlled

IsSeeEnemy

Checks if the bot is currently forward blocked.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeEnemy() then
```

IsSeeEnemyFor

Checks if the bot is currently forward blocked.

Parameters: (number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeEnemyFor(1) then
```

IsSeeEnemyFrom

Checks if the bot is currently forward blocked.

Parameters: (vector)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeEnemyFrom(Vector(0,0,0)) then
```

IsSeeEnemyFromFor

Checks if the bot is currently forward blocked.

Parameters: (vector, number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeEnemyFrom(Vector(0,0,0), 1) then
```

IsSeeFriend

Checks if the bot is currently forward blocked.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeFriend() then
```

IsSeeFriendFor

Checks if the bot is currently forward blocked.

Parameters: (number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeFriendFor(1) then
```

IsSeeFriendFrom

Checks if the bot is currently forward blocked.

Parameters: (vector)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeFriendFrom(Vector(0,0,0)) then
```

IsSeeFriendFromFor

Checks if the bot is currently forward blocked.

Parameters: (vector, number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeFriendFrom(Vector(0,0,0), 1) then
```

IsSeePosition

Checks if the bot is currently forward blocked.

Parameters: (vector)

Returns: true if it is, false if not

Example:

```
if bot:IsSeePosition(Vector(0,0,0)) then
```

IsSeeTarget

Checks if the bot is currently forward blocked.

Parameters: (entity)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeTarget(ent) then
```

IsSeeTargetFor

Checks if the bot is currently forward blocked.

Parameters: (entity, number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeTargetFor(ent, 1) then
```

IsSeeTargetFrom

Checks if the bot is currently forward blocked.

Parameters: (entity, vector)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeTargetFrom(ent, Vector(0,0,0)) then
```

IsSeeTargetFromFor

Checks if the bot is currently forward blocked.

Parameters: (entity, vector, number)

Returns: true if it is, false if not

Example:

```
if bot:IsSeeTargetFrom(ent, Vector(0,0,0), 1) then
```

IsStuck

Checks if the bot is stuck.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsStuck() then
```

IsUsing

Checks if the bot is using.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsUsing() then
```

IsWalking

Checks if the bot is walking.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsWalking() then
```

IsZooming

Checks if the bot is zooming.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsZooming() then
```

IsZoomingSniper

Checks if the bot is zooming + a sniper.

Parameters: (none)

Returns: true if it is, false if not

Example:

```
if bot:IsZoomingSniper() then
```

JumpTo

Tells the bot to jump to a specific position.

Parameters: (vector)

Returns: nil

Example:

```
bot:JumpTo(Vector(0,0,0))
```

KeyDown

Checks if the bot is holding down a specific key.

Parameters: (key)

Returns: true if it is, false if not

Example:

```
if bot:KeyDown(IN_ATTACK) then
```

See Also: KeyPressed, KeyReleased

KeyPressed

Checks if the bot has pressed a specific key.

Parameters: (key)

Returns: true if it is, false if not

Example:

```
if bot:KeyPressed(IN_ATTACK) then
```

See Also: KeyDown, KeyReleased

KeyReleased

Checks if the bot has released a specific key.

Parameters: (key)

Returns: true if it is, false if not

Example:

```
if bot:KeyReleased(IN_ATTACK) then
```

See Also: KeyDown, KeyPressed

LastSeenEnemyPos

Returns the last seen enemy position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:LastSeenEnemyPos()
```

LastSeenEnemyTime

Returns the last seen enemy time.

Parameters: (none)

Returns: (number)

Example:

```
local time = bot:LastSeenEnemyTime()
```

LastSeenFriendPos

Returns the last seen friend position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:LastSeenFriendPos()
```

LastSeenFriendTime

Returns the last seen friend time.

Parameters: (none)

Returns: (number)

Example:

```
local time = bot:LastSeenFriendTime()
```

LastSeenTargetPos

Returns the last seen target position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:LastSeenTargetPos()
```

LastSeenTargetTime

Returns the last seen target time.

Parameters: (none)

Returns: (number)

Example:

```
local time = bot:LastSeenTargetTime()
```

LeftClick

Tells the bot to left click.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClick()
```

LeftClickAlt

Tells the bot to left click + alt.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClickAlt()
```

LeftClickAltReload

Tells the bot to left click + alt + reload.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClickAltReload()
```

LeftClickReload

Tells the bot to left click + reload.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClickReload()
```

LeftClickZoom

Tells the bot to left click + zoom.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClickZoom()
```

LeftClickZoomReload

Tells the bot to left click + zoom + reload.

Parameters: (none)

Returns: nil

Example:

```
bot:LeftClickZoomReload()
```

LookAt

Tells the bot to look at a specific position.

Parameters: (vector)

Returns: nil

Example:

```
bot:LookAt(Vector(0,0,0))
```

LookAtBack

Tells the bot to look at its back.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtBack()
```

LookAtBody

Tells the bot to look at a specific body.

Parameters: (vector)

Returns: nil

Example:

```
bot:LookAtBody(Vector(0,0,0))
```

LookAtBodyRandom

Tells the bot to look at a specific body randomly.

Parameters: (vector)

Returns: nil

Example:

```
bot:LookAtBodyRandom(Vector(0,0,0))
```

LookAtChest

Tells the bot to look at its chest.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtChest()
```

LookAtChestRandom

Tells the bot to look at its chest randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtChestRandom()
```

LookAtClosestPosFromPath

Tells the bot to look at the closest position from a path.

Parameters: (path)

Returns: nil

Example:

```
bot:LookAtClosestPosFromPath(path)
```

LookAtFeet

Tells the bot to look at its feet.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtFeet()
```

LookAtFeetRandom

Tells the bot to look at its feet randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtFeetRandom()
```

LookAtHead

Tells the bot to look at its head.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtHead()
```

LookAtHeadRandom

Tells the bot to look at its head randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtHeadRandom()
```

LookAtLastEnemyPos

Tells the bot to look at the last seen enemy position.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLastEnemyPos()
```

LookAtLastFriendPos

Tells the bot to look at the last seen friend position.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLastFriendPos()
```

LookAtLastTargetPos

Tells the bot to look at the last seen target position.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLastTargetPos()
```

LookAtLeftHand

Tells the bot to look at its left hand.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLeftHand()
```

LookAtLeftHandRandom

Tells the bot to look at its left hand randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLeftHandRandom()
```

LookAtLeftShoulder

Tells the bot to look at its left shoulder.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLeftShoulder()
```

LookAtLeftShoulderRandom

Tells the bot to look at its left shoulder randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtLeftShoulderRandom()
```

LookAtPos

Tells the bot to look at a specific position.

Parameters: (vector)

Returns: nil

Example:

```
bot:LookAtPos(Vector(0,0,0))
```

LookAtRightHand

Tells the bot to look at its right hand.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtRightHand()
```

LookAtRightHandRandom

Tells the bot to look at its right hand randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtRightHandRandom()
```

LookAtRightShoulder

Tells the bot to look at its right shoulder.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtRightShoulder()
```

LookAtRightShoulderRandom

Tells the bot to look at its right shoulder randomly.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtRightShoulderRandom()
```

LookAtTarget

Tells the bot to look at a specific target.

Parameters: (entity)

Returns: nil

Example:

```
bot:LookAtTarget(Entity(1))
```

See Also: LookAtLastTargetPos, LookAtTargetPos

LookAtTargetPos

Tells the bot to look at a specific target position.

Parameters: (none)

Returns: nil

Example:

```
bot:LookAtTargetPos()
```

See Also: LookAtLastTargetPos, LookAtTarget

MaxSpeed

Returns the max speed of the bot.

Parameters: (none)

Returns: (number)

Example:

```
local speed = bot:MaxSpeed()
```

MoveTo

Tells the bot to move to a specific position.

Parameters: (vector, function-optional)

Returns: nil

Example:

```
bot:MoveTo(Vector(0,0,0))
bot:MoveTo(Vector(0,0,0), function() print("finished") end)
bot:MoveTo(Vector(0,0,0), function() print("finished") end, 10)
```

MoveToClosestPosFromPath

Tells the bot to move to the closest position from a path.

Parameters: (path, function-optional)

Returns: nil

Example:

```
bot:MoveToClosestPosFromPath(path)
```

See Also: MoveToLastTargetPos, MoveToTargetPos

MoveToLastTargetPos

Tells the bot to move to the last seen target position.

Parameters: (function-optional)

Returns: nil

Example:

```
bot:MoveToLastTargetPos()
```

See Also: MoveToClosestPosFromPath, MoveToTargetPos

MoveToTargetPos

Tells the bot to move to a target position.

Parameters: (function-optional)

Returns: nil

Example:

```
bot:MoveToTargetPos()
```

See Also: MoveToClosestPosFromPath, MoveToLastTargetPos

NextPathPos

Returns the next path position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:NextPathPos()
```

PathAvoidPos

Returns the path avoid position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:PathAvoidPos()
```

PathLength

Returns the path length.

Parameters: (none)

Returns: (number)

Example:

```
local length = bot:PathLength()
```

PathPos

Returns the path position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:PathPos()
```

PressButton

Makes the bot press a button. The function can take any number of buttons as arguments, and will press them all for the same time. The time period is **RANDOM** for each activation. The amount of time to press is **RANDOM**.

Parameters: (button1, ... , time to randomly

Returns: nil

Example:

```
bot:PressButton(IN_ATTACK)
bot:PressButton(IN_ATTACK, IN_ATTACK2, 0.5, 1.5)
bot:PressButton(IN_ATTACK, IN_ATTACK2, 0.5, 1.5, IN_JUMP)
```

See Also: HoldButton, ReleaseButton

PressedButtons

Returns the bot release a button. The function can take any number of buttons as parameters, and will apply for every one of them.

Parameters: (button1, ...)

Returns: (none)

Example:

```
bot:PressedButtons(IN_ATTACK)
bot:PressedButtons(IN_ATTACK, IN_ATTACK2)
bot:PressedButtons(IN_ATTACK, IN_ATTACK2, IN_JUMP)
```

See Also: HoldButton, PressButton

ReleaseButton

Releases a button.

Parameters: (button)

Returns: nil

Example:

```
bot:ReleaseButton(IN_ATTACK)
```

RemoveAllSignalThread

Removes a thread of from the bots signal list, so that signal events for the bot won't be sent to the thread.

Parameters: none

Returns: none

Example:

```
bot:RemoveAllSignalThread()
```

RemoveSignalThread

Removes a thread of from the bots signal list, so that signal events for the bot won't be sent to the thread.

Parameters: none

Returns: none

Example:

```
bot:RemoveSignalThread()
```

ResetAllStuck

Resets all stuck checks for the bot to none. This will cause an immediate re-evaluation of available goals. Normally the bot will wait 10 seconds to spend after an anti-stuck change to reevaluate the most effective meaning to do.

Parameters: none

Returns: none

Example:

```
bot:ResetAllStuck()
```

Say

The function sends the text to any messages through the normal chat checkers of the game. For flexibility, this function can be used for anything, and is not limited to just chat. It is also not limited to just the say command, it can be anything.

Parameters: (string, ...)

Returns: nil

Example:

```
bot:Say("I'm talk to it, I'm bot.")
```

See Also: Say, SayTeam

SayTeam

The function sends the text to any messages through the team chat checkers of the game. For flexibility, this function can be used for anything, and is not limited to just chat. It is also not limited to just the say command, it can be anything.

Parameters: (string, ...)

Returns: nil

Example:

```
bot:SayTeam("I'm talk to it, I'm bot.")
```

See Also: Say, SayTeam

SayVoice

Makes the bot a voice macro to the game. Used with global VOICE table values.

Parameters: (sound)

Returns: nil

Example:

```
bot:SayVoice(VOICE_GO_GO_GO)
```

SecondsBlocked

Used to calculate the total seconds the bot has current target, or a target passed as a parameter.

Parameters: (entity)(optional)

Returns: (entity)(optional)

Example:

```
local seconds = bot:SecondsBlocked()
```

SecondsSeeEnemy

Used to calculate the total seconds the bot has current target, or a target passed as a parameter.

Parameters: (entity)(optional)

Returns: (entity)(optional)

Example:

```
local seconds = bot:SecondsSeeEnemy()
```

SecondsSeeFriend

Used to calculate the total seconds the bot has current target, or a target passed as a parameter.

Parameters: (entity)(optional)

Returns: (entity)(optional)

Example:

```
local seconds = bot:SecondsSeeFriend()
```

SecondsSeeTarget

Used to calculate the total seconds the bot has current target, or a target passed as a parameter.

Parameters: (entity)(optional)

Returns: (entity)(optional)

Example:

```
local seconds = bot:SecondsSeeTarget()
```

SecondsStuck

Used to calculate the total seconds the bot has been stuck.

Parameters: (none)

Returns: (number)

Example:

```
local seconds = bot:SecondsStuck()
```

SelectWeapon

Tells the bot to select a weapon.

Parameters: (weapon)

Returns: nil

Example:

```
bot:SelectWeapon(WEAPON_FAMAS)
```

SetAttackingCurrentWeapons

Sets the bot to be flagged as weapon recoil controlled. Script controlled means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (boolean)

Returns: nil

Example:

```
bot:SetAttackingCurrentWeapons(true)
```

See Also: SetScriptControlled

SetAttackingScriptedWeapons

Sets the bot to be flagged as weapon recoil controlled. Script controlled means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (boolean)

Returns: nil

Example:

```
bot:SetAttackingScriptedWeapons(true)
```

See Also: SetScriptControlled

SetScriptControlled

Sets the bot to be flagged as script controlled. Script controlled means has to bot to legit will be used, allowing it to be entirely a help to control the behavior of the bot.

Parameters: (boolean)

Returns: nil

Example:

```
bot:SetScriptControlled(true)
```

See Also: IsScriptControlled

SetScriptControlledWeapons

Sets the bot to be flagged as weapon recoil controlled. Script controlled means has to bot to aim and shoot, not the game engine automatically doing it.

Parameters: (boolean)

Returns: nil

Example:

```
bot:SetScriptControlledWeapons(true)
```

See Also: SetScriptControlled

SetWeaponRecoil

Sets an entity to the recoil vector. This should mimic to each the entity at all times.

Parameters: (entity)

Returns: nil

Example:

```
bot:SetWeaponRecoil(ent)
```

Signal

Sends a signal to the bots signal thread. The function is different from the global signal function.Each bot has a signal thread, which is used for the bot's AI. Signals can be sent to it using this function.

Parameters: (string, ...)

Returns: nil

Example:

```
bot:Signal("hide")
```

See Also: SignalAllThread, SignalThread

ThreatAvoidPos

Returns the threat avoid position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:ThreatAvoidPos()
```

ThreatPos

Returns the threat position.

Parameters: (none)

Returns: (vector)

Example:

```
local pos = bot:ThreatPos()
```

ToggleWalk

Tells the bot to toggle walking.

Parameters: (none)

Returns: nil

Example:

```
bot:ToggleWalk()
```

TurnTo

Tells the bot to turn a specific position.

Parameters: (vector)

Returns: nil

Example:

```
bot:TurnTo(Vector(0,0,0))
```

TurnToFalse

Tells the bot to turn a specific position.

Parameters: (vector)

Returns: nil

Example:

```
bot:TurnToFalse(Vector(0,0,0))
```

TurnToPath

Tells the bot to turn to a specific path.

Parameters: (path)

Returns: nil

Example:

```
bot:TurnToPath(path)
```

TurnToPathFalse

Tells the bot to turn to a specific path.

Parameters: (path)

Returns: nil

Example:

```
bot:TurnToPathFalse(path)
```

UpdateAllStuck

Updates all stuck checks for the bot.

Parameters: none

Returns: none

Example:

```
bot:UpdateAllStuck()
```

UpdateAllStuckSniper

Updates all stuck checks for the bot + a sniper.

Parameters: none

Returns: none

Example:

```
bot:UpdateAllStuckSniper()
```

UpdateStuck

Updates the stuck check.

Parameters: none

Returns: none

Example:

```
bot:UpdateStuck()
```

UpdateStuckSniper

Updates the stuck check + a sniper.

Parameters: none

Returns: none

Example:

```
bot:UpdateStuckSniper()
```

UseWeapon

Tells the bot to use a weapon.

Parameters: (weapon)

Returns: nil

Example:

```
bot:UseWeapon(WEAPON_FAMAS)
```

WeaponRecoil

Returns the weapon recoil.

Parameters: (none)

Returns: (vector)

Example:

```
local recoil = bot:WeaponRecoil()
```

WeaponRecoilSniper

Returns the weapon recoil + a sniper.

Parameters: (none)

Returns: (vector)

Example:

```
local recoil = bot:WeaponRecoilSniper()
```

WeaponSpeed

Returns the weapon speed.

Parameters: (none)

Returns: (number)

Example:

```
local speed = bot:WeaponSpeed()
```

WeaponSpeedSniper

Returns the weapon speed + a sniper.

Parameters: (none)

Returns: (number)

Example:

```
local speed = bot:WeaponSpeedSniper()
```

ZeroLookOffset

Sets the look offset to zero.

Parameters: (none)

Returns: nil

Example:

```
bot:ZeroLookOffset()
```

ZeroLookOffsetSniper

Sets the look offset to zero + a sniper.

Parameters: (none)

Returns: nil

Example:

```
bot:ZeroLookOffsetSniper()
```

ZeroSpeed

Sets the speed to zero.

Parameters: (none)

Returns: nil

Example:

```
bot:ZeroSpeed()
```

ZeroSpeedSniper

Sets the speed to zero + a sniper.

Parameters: (none)

Returns: nil

Example:

```
bot:ZeroSpeedSniper()
```

Zooming

Tells the bot to zoom.

Parameters: (none)

Returns: nil

Example:

```
bot:Zooming()
```

ZoomingSniper

Tells the bot to zoom + a sniper.

Parameters: (none)

Returns: nil

Example:

```
bot:ZoomingSniper()
```

Zooming Behavior Library

Zooming behavior is a set of functions to control the bot's zooming behavior. Zooming is a movement technique, whether that be a gameplay change like Gamma, and etc.

Parameters: (none)

Returns: nil

Example:

```
bot:Zooming()
```

Zooming Sniper Behavior Library

Zooming behavior is a set of functions to control the bot's zooming behavior + a sniper. Zooming is a movement technique, whether that be a gameplay change like Gamma, and etc.

Parameters: (none)

Returns: nil

Example:

```
bot:ZoomingSniper()
```

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);
```

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: Game(Entity

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 GameId, 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();
```

GetTypeName

Gets the type name of the map goal.

Parameters: none

Returns: type name

Example:

```
type = mg.GetTypeName();
```

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();
pt = mg.GetUsePoint(0);
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

Matrix3

From Omni-bot Wiki

Contents

- 1 Matrix3 Functions
  - 1.1 Inverse
  - 1.2 InverseTransformVector
  - 1.3 TransformVector

Matrix3 Functions

Inverse

InverseTransformVector

TransformVector

ScriptGoal

From Omni-bot Wiki

Contents

- 1 ScriptGoal Functions
  - 1.1 Finished
  - 1.2 LimitToClass
  - 1.3 LimitToEntityFlag
  - 1.4 LimitToPowerUp

ScriptGoal Functions

This is a 0.7 feature.

Finished

LimitToClass

LimitToEntityFlag

LimitToPowerUp

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

Timer

From Omni-bot Wiki

Timer Functions

GetElapsedTime

Reset

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.

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

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 HasZoom
    - 2.1.5 Stealth
    - 2.1.6 InheritsVelocity
    - 2.1.7 NeedsInRange
    - 2.1.8 ManageHeat
    - 2.1.9 IgnoreReload
    - 2.1.10 RequiresTargetOutside
    - 2.1.11 RequiresShooterOutside
    - 2.1.12 WeaponType
    - 2.1.13 MaxEquipMoveMode
    - 2.1.14 Offhand
    - 2.1.15 ManualDetonation
    - 2.1.16 MustBeOnGround
    - 2.1.17 FireOnRelease
    - 2.1.18 UseMortarTrajectory
    - 2.1.19 SniperWeapon
    - 2.1.20 MaxAimError
    - 2.1.21 AimOffset
    - 2.1.22 PitchOffset
    - 2.1.23 ShootButton
    - 2.1.24 ZoomButton
    - 2.1.25 LowAmmoThreshold
    - 2.1.26 LowAmmoPriority
    - 2.1.27 FuseTime
    - 2.1.28 AmmoType
    - 2.1.29 ProjectileSpeed
    - 2.1.30 MinRange
    - 2.1.31 MaxRange
    - 2.1.32 MinChargeTime
    - 2.1.33 MaxChargeTime
    - 2.1.34 DelayAfterFiring
    - 2.1.35 ProjectileGravity
    - 2.1.36 DefaultDesirability
    - 2.1.37 Bias
    - 2.1.38 CalculateDefaultDesirability
    - 2.1.39 CalculateDesirability
    - 2.1.40 CalculateAimPoint
    - 2.1.41 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 PropertiesIAmmoType.

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.

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 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 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.

Bias

The bias for this weapon. Usually set per bot to give different bots different weapon usage characteristics.

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 TargetInfoof 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);
```