



# ET Mapping Tutorial

## Introduction

This tutorial has been set up to help would-be mappers looking for a step-by-step guide from the beginning. It assumes the reader has no idea how to use GtkRadiant and knows nothing about scripting and the like.

These pages were originally created for members of the TibeT clan and regular players on the TibeT servers, but anyone finding this tutorial is welcome to use it :)

I don't claim to know everything about mapping but I've learnt enough from various sources to be able to make [maps](#) of reasonable complexity. I've had a number of people asking various mapping questions and I thought it was a better idea to explain it all once to anyone interested. The tutorial is incomplete but I continue to add pages as I get the time.

Be aware that the subject is very large so there's a lot to explain; and that to make a decent fair-sized map can easily take over 200 hours, so with this as a hobby you need a lot of patience and spare time, but at least the tools to do the job are free. The best maps out there might have taken their authors 18 months to create, depending on how much free time they had.

Disclaimer: I may be wrong in my understanding of any aspect of the subject, but hey, I'm doing my best :)

## Introductory topics

To get from knowing nothing about mapping to being able to run around in your own first little map, follow these introductory topics in order from start to finish.

- [Getting Started](#)
- [Making your first brush](#)
- [Making your first map](#)

## First set of intermediate topics

Create a building inside a landscape, with a destructible window and working door.

- [Creating an environment](#)
- [Making a building outline in your environment](#)
- [Making a door](#)
- [Making a destructible window](#)

## Second set of intermediate topics

A break from brushes: a quick delve into *some* of the other mapping elements.

- [Creating the initial script](#)
- [Creating the mission text to be shown as the map loads](#)
- [Adding ambient sounds](#)

## Third set of intermediate topics

Back to Radiant again, for models, simple destructibles and constructibles. Starting to get more interesting now. Don't attempt these unless you've completed the previous topics.

- [Planting a tree](#)
- [Making stuff you can shoot up](#)
- [Making barbed wire](#)
- [Making a ladder](#)
- [Making a constructible MG42](#)

## Fourth set of intermediate topics

A set of topics with some variety, helping you ease into some of the more complicated subjects..

- [Secure doors](#)
- [Lighting](#)
- [Detail brushes vs structural brushes](#)
- [Health and ammo cabinets](#)

## Fifth set of intermediate topics

Some of the principal game features, the CP, bendy shapes and terrain.

- [Command Posts](#)
- [Curved walls and arches](#)
- [Cylinders, cones and curved roads](#)
- [Making terrain using GtkGenSurf](#)
- [Fine tuning the terrain by dragging vertices](#)
- [Skyboxes](#)

## Sixth set of intermediate topics

Some topics that help to bring interest to your map.

- [Bespoke graphics](#)
- [Making a constructible object like a ramp](#)
- [Making a destructible object like a gate](#)
- [Forward spawn flags](#)
- [Water](#)
- [Team speech](#)

## Final set of intermediate topics

This last set of intermediate topics cover the remaining components you'll want to make a distributable PK3 package.

- [Limbo camera and objectives narrative](#)
- [Making the game end](#)
- [Generating a tracemap](#)
- [Making the command map](#)
- [Making the picture to be shown while the map loads](#)
- [Making a PK3 file](#)

## Advanced topics

This final set of topics covers some of the more complicated stuff that you may never want to use - but if you've made it this far with the tutorial, you probably will :)

- [The Tank](#)
- Scripting in detail
- Grabbing the gold or radar parts, etc
- Making constructible/destructible doors
- Bespoke sound
- Making something simple move
- Trucks
- Shaders
- Custom command map icons



We are a small website-development company, specialising in providing a personal service at affordable prices.

On your behalf we register your domain name and organize the hosting of your new website with the UK's largest web hosting company. We then design and create a professional custom website to meet your needs. In addition we can provide graphic design for your corporate identity to complement the style of the website.

You are welcome to [contact us](#) to chat about your requirement, even if you are not quite sure what it is! We avoid tech-speak and talk in plain English to make the whole process straightforward for both parties.







This page has all my ET map pk3 files available for download. Please ensure you play the maps using these most recent versions.

Thanks to -=Hawk=- for designing this web page.

If you would like to follow my mapping tutorial, please click [here](#).

Map update log	
Download	View info
<a href="#">Berlin 1.2.0</a>	<a href="#">Mar 2009</a>
<a href="#">Breakout 2 1.4.0</a>	<a href="#">Feb 2009</a>
<a href="#">Chartwell 1.4.0</a>	<a href="#">Feb 2009</a>
<a href="#">Glider PanzerDuel 1.0.1</a>	<a href="#">Dec 2008</a>
<a href="#">Glider PanzerDuel LowGrav 1.0.1</a>	
<a href="#">Cluedo 1.3.0</a>	<a href="#">Dec 2008</a>
<a href="#">Radar Summer 1.3.0</a>	<a href="#">June 2008</a>
<a href="#">Troop Train 1.2.0</a>	<a href="#">May 2008</a>
<a href="#">British Bulldog 1.6.0</a>	<a href="#">Dec 2007</a>
<a href="#">Operation Chariot 1.2.0</a>	<a href="#">Oct 2007</a>
<a href="#">RTCW Depot 2 1.0.2</a>	<a href="#">Sep 2007</a>
<a href="#">Battery Recharged 1.3.0</a>	<a href="#">Aug 2007</a>
<a href="#">Breakout 3.7.1</a>	<a href="#">June 2007</a>
<a href="#">110 Factory 2.0.0</a>	<a href="#">June 2007</a>
<a href="#">TankBuster 2.0.0</a>	<a href="#">May 2007</a>
<a href="#">Glider 3.0.2</a>	<a href="#">May 2007</a>
<a href="#">Tiger 1.1.0</a>	<a href="#">Aug 2006</a>
<a href="#">Ludendorff Bridge 1.1.0</a>	<a href="#">Jan 2006</a>
<a href="#">2Tanks 1.7.1</a>	<a href="#">Oct 2005</a>
<a href="#">6Flags 1.1.0</a>	<a href="#">July 2005</a>

Berlin	
<p>A Berlin street, 1945. Allies have to capture the street in house-to-house combat.</p> <p>The first buildings to capture are marked by a red flare (which turns blue when Allies capture the flag and red on Axis recapture).</p> <p>Flags (which are always on the ground floor) must be held by the Allies for a total of 90 seconds to make the capture permanent.</p> <p>At that point the flare turns yellow, indicating that Allies must now eliminate all Axis defenders in the building to secure it. So long as at least one Axis defender remains alive in the building, it remains unsecured. When the flare turns yellow Axis have 20 seconds to re-occupy the building - after that time as soon as no Axis soldiers remain alive in the building it is secured.</p> <p>When the first buildings are secured the yellow flares are extinguished and red flares are lit outside the next buildings to be captured. In this way Allies progress up the street, trying to secure a total of 7 buildings to win.</p> <p>There are no dummy windows in the map, which means every window opening potentially conceals an enemy soldier: so advancing in the open street is a risky business. Allied cov ops will need to provide smoke cover and seek out enemy snipers to assist the advance of Allied troops.</p> <p>The flares are an important element in the game, and are produced using ET smoke generator entities. Some players will have disabled smoke, perhaps to help performance on low-spec machines or maybe to gain a small visual advantage, and so will not be able to see the flares.</p> <p>To see flare smoke you need to set cg_wolfparticles to 1 (on NQ you need cg_smokeparticles set to 1).</p>	<p>Version 1.2.0 March 2009</p> <p>First public release</p> <p><a href="#">screen shots</a></p>
Glider Panzer Duel	
<p>An open version of the Glider map with no objectives, it's just a playground for team panzer deathmatch.</p> <p>There is nothing to build and no obstructions. The glider is pre-built and can be flown by anyone, and shot down by any weapon. It auto-rebuilds when destroyed.</p> <p>The spawn locations change every minute, but you can override the random choices by selecting any of the spawns on the command map. All spawn locations are available, some of which might be shared by the enemy. Spawn times are every 12 seconds.</p> <p>Ammo boxes supplying unlimited ammo are dotted around the map, and recharge times are quicker than normal.</p> <p>The map ends after 30 mins (arbitrarily choosing Axis as the winners).</p>	<p>Version 1.0.1 December 2008</p> <p>Silly festive release</p> <p>Also alternative LowGrav version</p>
Breakout 2	

The sequel to the original Breakout map, Breakout 2 is available now.

Picking up where the original Breakout left off, the sequel has far better graphics and some great new gameplay features.

Allies have to escort the battle-stained Tiger tank along railway track through a station and its surrounding village to escape.

In a departure from the usual tank barrier format, each of the main barrier objectives is a sequential double objective.

Stage 1	Allies escort the tank along the rail track towards the station.
	Axis attempt to dynamite the railway footbridge to prevent the tank reaching the station. If successful, Allies have to dynamite the wreckage to clear the path.
Stage 2	Allies get the tank to the station. Allies must build a ramp to get the tank off the track. Nearby the Axis need to build a tank barrier to prevent the tank entering the village. Both teams will be attempting to dynamite the enemy's construction.
Stage 3	Allies get through the village and approach the road tunnel under the railway. Axis attempt to dynamite the railway tunnel to prevent the tank reaching the bunker. If successful, Allies have to dynamite the rubble to clear the path.
Stage 4	Allies enter the bunker and need to leave via the exit at its far side. Axis attempt to dynamite the control room over the bunker exit to prevent the tank escaping. Note that if the control room is destroyed, it also destroys the Axis CP. If successful, Allies have to dynamite the debris to clear the path. This allows Axis to repair their CP. Allies win when they get the tank out of the bunker.

Version 1.4.0  
February 2009

Axis barracks spawn given extra routes out, to prevent spawners from being mown down by tank MG at tunnel.

Axis barracks spawn and bunker spawn doors given glass panels.

Troop Train

Axis are transporting armour reinforcements to the front line. Allies must destroy the 2 tanks on the trains before they arrive at their destination.

All the action takes place on two moving trains adjacent on parallel tracks.

First objective is to blow up the crates that block the Allies path up the train. This gives them access to the midtrain flag forward spawn, and sets them up for the final push to the front of the train where the Tiger tanks are being transported.

Players can move through the carriages, or along the top or sides, and can jump from one train to the other.

Passing overhead gantries and bridges help to prevent sniper domination, and quick respawn times mean players are never missing from the action for long. The straightforward layout also means it's quick to learn and no-one gets lost.

With a 10 minute map time limit, I believe it is best played in stopwatch mode.

Recommended 2-8 players per team, probably mayhem beyond that.

Version 1.2.0  
May 25th, 2008

Click the image below to view the Troop Train gallery



Cluedo


Set in the country manor of the board game Cluedo (Clue in the U.S.), there is no real scenario - this is played **just for laughs** as a break between "proper" maps. Each game lasts about 10 minutes.

Axis spawn in the Ballroom (centre top of the map) and Allies

Version 1.3.0  
December, 2008

Extra bounce mats added.

Mines disabled (accidentally!), to

<p>in the Hall (centre bottom).</p> <p>A flag then randomly spawns in one of the seven other rooms. The upper edge of the room's walls glow white when the flag is active inside.</p> <p>The flag can be captured by either team, giving the white glow an inner blue (allies) or red (axis) core to show the current possessor. When the flag is held for a <b>total</b> of 90 seconds by a team it is secured and removed from the map. The white glow is removed and the inner core remains to show that the room has been fought over and which team secured it.</p> <p>This process is repeated until one team has secured four flags and is victorious.</p> <p>Players can move freely between the rooms and enter the cellar. The doors are glass lined so that players are aware of anyone camping the other side, which is legitimate and likely to be the case.</p> <p>Each room can be entered or exited through its windows, and the four corner rooms have secret passages which link them. Players can also move around and over the building.</p> <p>Each of the seven flag rooms has a trap which is activated by the lever in the corridor outside the room. Once a trap is activated it cannot be re-used for 30 seconds.</p> <p>The traps make the interior of the room uncomfortable or downright dangerous for its occupants, which helps to shift players camping the flag. The player activating a trap is teleported to a location outside the room, so he can witness the carnage he has caused.</p> <p>To make things even more lively, the players will find there are all sorts of places that engineers can plant mines to surprise the unwary...</p> <p>The action is fast and frantic, with low respawn times to ensure all the players stay in the thick of it.</p> <p>Recommended for teams of between 2 and 12 per team.</p>	<p>be restored in next release.</p> <p>Click the image below to view the Cluedo gallery</p> 
<h2>British Bulldog</h2>	
<p>The entire action centres on the race to deliver 6 gold crates into your team's vaults.</p> <p>Engineers are not necessary: there are no dynamitable objectives and nothing to build.</p> <p><b>This is a manic romp and not to be taken seriously :)</b></p> <p><u>GOLD CRATES</u></p> <p>Grab each of your 6 gold crates one at a time and deliver them to the vault. Allied crates from the East cages must go into the West vault, and vice-versa. Similarly for the Axis crates in the North cages to the South vault and vice-versa. Arrows shown in team colours (Allies=blue Axis=red) guide the way for the current objective.</p> <p><u>FLAGS</u></p> <p>There are four flag poles. They are NOT forward spawn points: instead they function like a Command Post. Each flag captured improves your team's Charge speed:</p> <p>0 flags = no bonus  1 flag = small bonus  2 flags = better bonus, plus Cov Op landmine warnings are transmitted  3 flags = good bonus  4 flags = great bonus</p> <p><u>STARS</u></p> <p>Stars will drop to the ground from time to time. Grab them before they disappear to be granted power-ups and bonuses. Notable powerups are:</p>	<p>Version 1.6.0  4th December, 2007</p>

<p><b>Expressway.</b> This provides a launch pad from the spawn point to the opposite side of the map. Handy when the next objective is over there.</p> <p><b>Sleigh pad.</b> This provides a jump pad (located at the snowman) up to the flying sleigh. The pad operates only when the sleigh is overhead. When ridden, the sleigh will launch dual panzer strikes at the enemy base.</p>	
<h3>Operation Chariot</h3>	
<p>In 1942 the greatest threat to allied shipping was the mighty German battleship, Tirpitz. Her vast size meant that the only dock on the Atlantic seaboard that could accommodate her was at St Nazaire.</p> <p>So Operation Chariot was born, its mission to blow up the dock and neutralize the threat of the Tirpitz.</p> <p>The game commences with the allies having rammed the south gate with the destroyer HMS Campbeltown. They must now attempt to destroy as many of the 6 main dockyard facilities as possible.</p> <p>There are 6 objectives to destroy, with victory being achieved to differing degrees at game end:</p> <p>0-1 objectives destroyed DECISIVE Axis victory 2 objectives destroyed Major Axis victory 3 objectives destroyed Marginal Axis victory</p> <p>4 objectives destroyed Marginal Allied victory 5 objectives destroyed Major Allied victory 6 objectives destroyed DECISIVE Allied victory</p> <p>The map is very open and the allies can attack the objectives in any order - however, for the sake of gameplay, they don't have it all their own way.</p> <p>The Allies must deliver a demolition charge, taken from the ship's hold, to any objective before it can be dynamited and destroyed. Two demolition charges are initially available, and when the first two objectives have been destroyed another two demolition charges are made available in the ship's hold. When the fourth objective has been destroyed, the final two charges become available.</p> <p>This allows the allies to choose their targets, but limits them to attacking up to two at a time.</p> <p>In addition, the axis have two barracks for their spawning. One is near 3 objectives and the second near the other 3 objectives. Axis players always spawn in the South Barracks, but have a passage that links the two barracks instantaneously - that is, an axis player at the South Barracks can move immediately (teleport) to the East Barracks and vice versa.</p> <p>This is like having a choice of 2 spawn points, but without the bother of selecting them on the command map.</p> <p>To guide axis players to the objectives under threat, there are ingame command maps in each Barracks. Indicators on these maps show which objectives are under attack, and which ones are especially in danger because demolition charges have been placed.</p> <p>By checking this map on spawning, axis players can take themselves immediately to the action.</p>	<p>Version 1.2.0 25th October, 2007</p> <p>Corrects the mistake which made the carried obj icon visible through walls.</p> <p>I quite liked the effect but it confused and annoyed some players.</p> <p>This is a joint project with Avoc, author of Nemo.</p> <p>Click the image below to view the gallery of images taken during development</p> 
<h3>RTCW Depot 2</h3>	
<p>RTCW Depot was one of my favourite RTCW maps which I always hoped would turn up on ET. It never did, so I have done a conversion to get this classic played again.</p> <p>Based on the revised Depot 2, it has a few changes from the RTCW version but remains essentially the same.</p> <p>The enemy forces confront each other in a rail depot. The</p>	<p><a href="#">Version 1.0.2</a> September 28th, 2007</p> <p>Some players found v1.0.1 too dark. Version 1.0.2 has had the ambient light increased.</p> <p><a href="#">Version 1.0.1</a></p>

Allies must destroy the Axis Anti-Aircraft gun at the north end of the depot before the Axis destroy the Allied Field HQ in the south.

Best played with larger team sizes of say at least 8 per team, to allow some players to attack the enemy objective while some defend their own.

September 6th, 2007

Click the image below to view the gallery



## Battery Recharged

I'm a big fan of Seawall Battery, and like many players always wished you could get up onto the gun.

So I thought I would do a conversion of this classic map that would open up that area for some great firefights.

The original map has a bit of a bottleneck at the ramp, which lead to the development of a scriptfix to allow the backdoor to be dynamited.

In this version there are two frontal routes for the allied assault, so the blowable-backdoor script isn't needed.

### Changes from Seawall Battery

- Three additional primary objectives created for allies, making four in total:
  - destroy radar station (next to bunker)
  - destroy gun platform
  - destroy gun controls (same as in original map)
  - destroy power supply (at bottom of indoor railway slope)
- Assault ramp moved to the slope under the gun barrel, and time taken to build increased by 50%.
- Original location of assault ramp smoothed to allow allies to get up slope on foot.
- Axis can build barricade (satchel objective) to block the new slope.
- Players can now reach gun and gun platform from ramp and from inside gun room.
- New Axis-only door added at beach top leading directly to gun room, to allow Axis to better defend the slope and the gun.
- New constructible (satchel objective) Axis-and-disguised-cov-op-only door added near "power supply room", to slow allied access to it via gun room assault.
- Allied East Spawn moved closer to the lighthouse.
- Axis spawn time reduced to 20 secs.
- More areas now permit landmines.

Version 1.3.0  
August 31st, 2007

## Radar Summer

I've always liked Wurzburg Radar, I just wished it would stop raining.

So I have done a conversion of this classic ET map to make it sunny with unlimited views to the horizon. To make it more interesting than just having nicer weather, I included these changes:

- Extra Axis spawn location in the Road Hut - this helps them fight back more quickly when Allies take the Bunker. They will automatically spawn there if the Allies capture the bunker, *unless* the side door has been blown.

Version 1.3.0  
June 25th, 2008

All 4 objs now required.

Blowable generator added, which opens bunker doors when blown.

Click the image below to view the Radar Summer gallery

		<div>2 extra Allied primary objectives - destroy the North and South Radar Stations.</div> <div><ul style="list-style-type: none"><li>• A lot of the player clipping has been removed, allowing players to get onto high places previously not possible.</li><li>• The tank in the garage can now be repaired by Axis, and driven up the road to a defensive position.</li><li>• New terrain textures, and things like owl hoots replaced with bird song.</li><li>• Axis respawn time reduced.</li><li>• Generator added behind door under wooden bridge. When blown by covops it opens the door in the bunker and the door being the main entrance to give additional passageways.</li></ul></div>	
Glider			
<div>Spring 1943, the Austrian Alps: Allied Intelligence has reported that the enemy are developing a powerful new tank in a secure research facility. It is vital that one of the prototype Jagdpanther models be photographed for assessment.</div> <div>Some Allied 'Horsa' gliders are in enemy hands and are crated in a nearby Axis outpost: capture the outpost and use the gliders to assist in the operation.</div> <div>The Allies are dropped in by parachute and must assault the Outpost in order to gain access to the gliders crated there. They must then fly at least one glider from the Outpost to a hilltop overlooking the research facility. From there they will be able to destroy the generator, which opens the sealed door of the tank storage building.</div> <div>They must steal the tank and transport it to the Outpost, where a Photo Reconnaissance Spitfire will be able to take photographs of it.</div>		<div>Version 3.0.2 May, 2007</div> <div>Major changes to spawning, including when the allies take the outpost, the axis paratroopers can try to get it back; new axis spawn at centre of map; new allied spawn in tank garage; and when the tank reaches the guardpost the allied spawn is set there.</div> <div>The parachute spawns are removed once the allied CP is built.</div> <div>The 2nd tank barrier has been moved to be closer to the outpost.</div> <div>Axis CP moved to the outpost.</div> <div>Axis compound spawn enlarged to give more shoulder room.</div> <div>Ladder added to allow access to the top of the garage.</div>	
Chartwell			
<div>Chartwell, England, 1943.</div> <div>Axis have launched a surprise paratroop attack on Chartwell, the home of Winston Churchill, to steal the plans for the D-Day invasion.</div> <div>This is a small map of 20 mins duration, centred on the Axis attempts to enter the house and obtain the plans, best for 6-16 players per team.</div> <div>Gameplay</div> <div>Axis initially spawn at a building nearby, but they can quickly enable their paratroopers to attack by damaging the AA gun or the AA gun controls. With either of these out of action, the Axis spawn location is randomly changed to any location on the map for each Axis respawn. This allows the Axis team to attack from potentially any direction.</div> <div>Allies need to repair both in order to force the Axis back to the original spawn.</div> <div>The main Allied spawn is in the basement. There is an Allied-only flag in the pavilion, which if captured gives the Allies the option of spawning outside the house.</div> <div>To get into the house, Axis must either dynamite the main entrance (non-repairable) or satchel either of the stairwell doors (repairable).</div>		<div>Version 1.4.0 February, 2009</div> <div>Fixed map exploit which allowed docs to be taken from closed safe..</div> <div></div>	



The Key must be taken from the ground floor dining room to the safe in the top floor study. To get into the study either of its doors must have been destroyed (one by dyna, one by satchel, both non-repairable).

On delivering the Key to the safe, the safe door opens and the Plans can be stolen. They must be taken to the Command Post in the grounds, and the Command Post built to secure an Axis victory.

Although all of the windows visible in picture 3 can be broken, getting out of the house with the Plans is a little trickier than it seems, as the broken windows are too narrow to jump through.

In this map Axis Cov Ops can set booby traps at the AA gun and AA gun controls in an effort to safeguard the paratroop attack from interruption. The booby traps are detonated if an unwary allied soldier touches the faint translucent white bar that reveals their presence. Allied Cov Ops can defuse the booby traps.



### TankBuster

Spring 1944, Italy. The Allies are finally advancing at the Gustav Line when an S.O.E. Intelligence report reveals a hidden reserve of Axis armour near Monte Cassino.

The Allies must breach the depot heavy defences and destroy all the tanks stockpiled there before they can be used in an Axis counter offensive.

Description The Allies must escort a truck laden with a massive bomb to the re-inforced main gate of the Axis tank depot. The bomb is detonated when the Allied Command Post is built.

Once access has been obtained to the interior of the depot, the Allies must assault the tank garage and destroy the 8 jagdpanther tanks inside. Tanks destroyed cannot be repaired.

Version 2.0.0  
9th May, 2007

Extra route to the station, wider tunnels, mines now permitted in tunnels, timelimit reduced to 25 mins

### Tiger!

Axis infantry must support their Tiger tank as it tries to destroy an allied truck convoy.

Allied infantry must assist the convoy in escaping through the town.

This map features a Tiger tank which does not need player escort - it independently hunts the allied trucks.

Allies send false radio signals and damage the axis communication system in an effort to lead the Tiger away from its prey.

Axis can take fuel cans to dips in the road on the convoy route, to create firewalls that hold up the truck movement - giving the Tiger the opportunity to catch and destroy the trucks.

Version 1.1.0  
16th August, 2006





### 110 Factory

Allied raiders must get ashore from their stolen U-boat, storm the Me 110 factory and grab the secret radar components being fitted to aircraft on the production line.

The Allies start inside the U-boat and must assault the forward

Version 2.0.0  
10th June, 2007

Extra cover, indoors and outdoors.  
High level gantries inside the

<h2>2tanks</h2>	
<p>bunker using high-speed dinghies and weather balloons. Destroying the main entrance to the airstrip will give the Allies access to the Me110 factory and the radar components within.</p> <p>Allies can enter the factory via the north doors or the side entrance and will need to fight their way along the production line to the south end where the radar components are kept.</p> <p>The radar components must be brought to the American halftrack outside the factory for an allied victory.</p>	<p>factory give another route to the radar parts.</p> <p>Removed balloon rockets, and tidied some textures.</p> <p>Damaged Me110 engines cause more harm when they explode.</p>
<h2>6flags</h2>	
<p>Both teams have the same objective:</p> <ul style="list-style-type: none"><li>• Get your tank to the Fuel Dump before the enemy gets his there!</li></ul> <p>If the time limit expires the winning team is the one which has made the most progress with their tank.</p>	<p>Version 1.7.1 12th October, 2005</p> <p>Fuel is no longer needed</p>
<h2>Ludendorff Bridge</h2>	
<p>For either team to win:</p> <ul style="list-style-type: none"><li>• Capture both base flags and hold them for 2 minutes, or...</li><li>• Control all 6 flags</li></ul> <p>If the time limit expires the winning team is the one which controls the most flags.</p>	<p>Version 1.1.0 7th July, 2005</p>  
<h2>Breakout</h2>	
<p>1945, the famous bridge at Remagen over the Rhine.</p> <p>The Axis commander has been given orders to blow the bridge - but not too soon (to allow retreating soldiers to escape) and not too late (to prevent pursuing allies crossing the Rhine).</p> <p><b>This map has several unusual <a href="#">features</a></b></p>	<p>Version 1.1.0 24th January, 2006</p> <p>Extra barricades provided at the towers to prevent enemy soldiers from flooding into the spawn and spawnkilling</p>
<h2>2tanks</h2>	
<p>1944, France, soon after D-Day. A company of Allied soldiers are trapped by superior Axis forces near a french village. They must salvage a tank from the depot, force their way through the village and escape along the railway track.</p> <p>This is a compact map, with the action focussed around the railway bridge, but the environment is very open allowing plenty of choice and scope for tactics. All the buildings can be entered.</p>	<p>Version 3.7.1 18th June, 2007</p> <p>Additional route from allied spawn to the tank garage - helps if Axis have infiltrated and are camping the tank in the garage.</p> <p>Allied spawn front window made bulletproof and ladder into spawn removed - Axis unable now to enter allied spawn.</p> <p>Assault ramp given additional ladder, quickens access to the bridge.</p> <p>Ammo / Health cabinets removed from Allied Hut spawn - no reason now for Axis to enter hut.</p> <p>Ammo / Health cabinets moved from Axis bank to adjacent</p>



		<p>terraced house - more accessible to both teams.</p> <p>Additional windows added to Axis bank.</p> <p>Additional opening added to Church tower.</p> <p>Halftrack moved forward a little, giving more cover.</p> <p>Additional cover provided on bridge and at forward Allied window, which has had an MG42 added.</p> <p>Some texture tidying and terrain tweaking.</p> <p>Map time limit reduced by 1 minute to 24 minutes.</p>
--	--	--



# ET Mapping Tutorial

## Lesson 1

### Topics

#### Getting Started

[Setup](#)

[What is Radiant?](#)

[Common folders and associated files](#)

[Configuring Radiant](#)

[The Radiant display](#)

[Back to main menu](#)

### Setup

[\[Top\]](#)

Download and install [GtkRadiant version 1.4](#) (now available from my server). 1.4 is not the latest version, but I prefer it at this time.

It is best to have a clean install of ET separate to the one you play on, for making and testing maps.

However I haven't done this (I was nervous that installing ET twice might make a mess in the registry) so I use my usual ET environment. There is a penalty for using the same environment: before running Radiant I have to drag all of the non-core PK3 files out of etmain and into a temp folder. When I'm finished I drag them all back again. This is because Radiant will otherwise include elements from the other PK3s into the map I am creating, which will cause all sorts of display problems when distributed.

If you don't want a second install and intend to use your existing ET environment then **for now you won't have to remove any PK3 files** - this basic lesson will be unaffected by the presence of additional PK3 files. But if you continue with mapping as a hobby you will need to do what I do. See next for the PK3s you can leave in the \etmain folder.

The PK3s I leave in \etmain are:

- common.pk3
- etmapcycle.pk3
- mp\_bin.pk3
- pak0.pk3
- pak1.pk3
- pak2.pk3
- All the 1KB campaign PK3 files

Everything else should be moved into a temporary folder while you do your mapping/testing - drag them back before playing ET online again. At the later stages of mapping, ie once you don't start using any further new textures, you will be able to leave the other PK3s in place while you map.

Create these folders within **etmain**:

- levelshots
- maps
- scripts
- sound
- textures

Create these folders within your new **sound** folder:

- maps
- scripts

The value "<yourmap>" will be used in this tutorial to represent the name of your map.

What is Radiant and what does it do?

[\[Top\]](#)

Radiant is a map editor which allows you to create the physical geography of your map. You'll be able to create furniture, buildings and terrain in 2D and 3D, and explore and view your creation in 3D without actually running ET. When you save your creation Radiant creates a text file called "<yourmap>.map" in the maps folder. Later on you'll probably edit this file by hand, but don't do this for now.


Radiant also provides a way of compiling your map into the format required by ET to interpret and play it. The compiler will produce a file called "<yourmap>.bsp", also in the maps folder.

Common folders and associated files

[\[Top\]](#)

Maps are stored in the etmain\maps folder and have a .map suffix.

The <yourmap>.map file is compiled to produce a <yourmap>.bsp file, also placed in the \maps folder. Most of the other files that make up your map package contain text and can be created and edited using Wordpad.



When you are making a map, settle on its name early, it becomes more and more tiresome to change it the further you have progressed.

The commonly used folders and files within etmain are:

levelshots	command map (<yourmap>_cc.tga) and map loading photo (<yourmap>.tga)
maps	map object (<yourmap>.bsp)
	the script that powers your map (<yourmap>.script)
	objective descriptions for the limbo views (<yourmap>.objdata)
scripts	map description shown while the map loads (<yourmap>.arena)
	special information regarding the textures in your map (<yourmap>.shader and <yourmap>_levelshots.shader)
sound/maps	ambient sound effect placement (<yourmap>.sps)
sound/scripts	speech, like "They've stolen the tank!" (<yourmap>.sounds)
sound/<yourmap>	If you create bespoke sounds (WAV files of a particular format)
textures/<yourmap>	If you create bespoke textures (TGA and JPG files of particular dimensions)

Configuring Radiant

[\[Top\]](#)

Run Radiant and identify your ET installation folder to the program.

Press P to bring up program preferences.

Select the following tabs and ensure the suggested settings are applied:

--	--

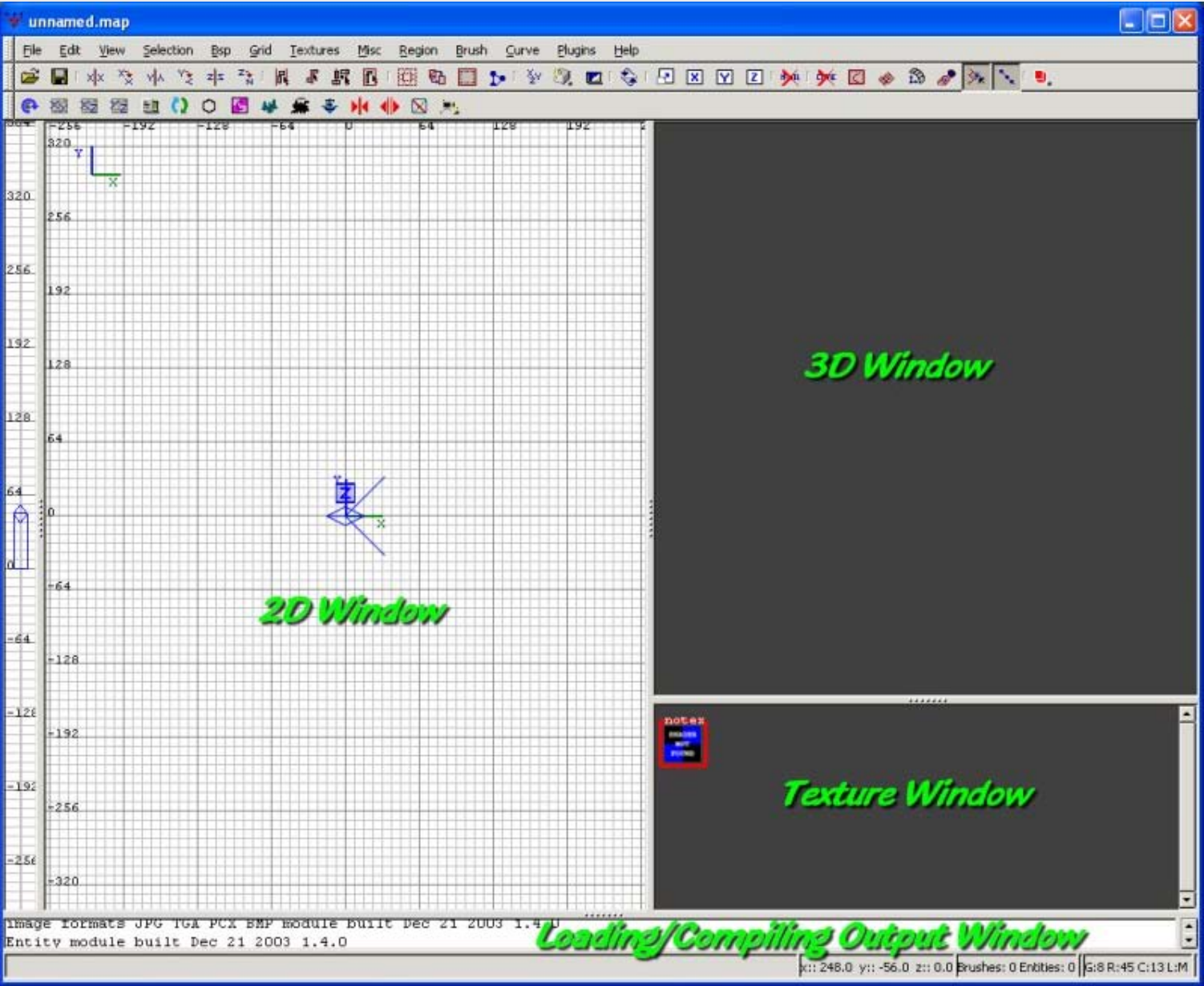
Game Settings	Select ET as the default game
	Tick Auto load selected game
2D display/rendering	Tick first 2 boxes
3D view	Tick boxes 1 4 and 5, and 3 if you want inverted mouse
Editing	Tick first 3 boxes
	Undo levels = 30
Startup/autosave	Tick boxes 2 and 4, set autosave every 5
BSP monitoring	Tick boxes 1, 2 and 5

Click OK

The Radiant display

[\[Top\]](#)

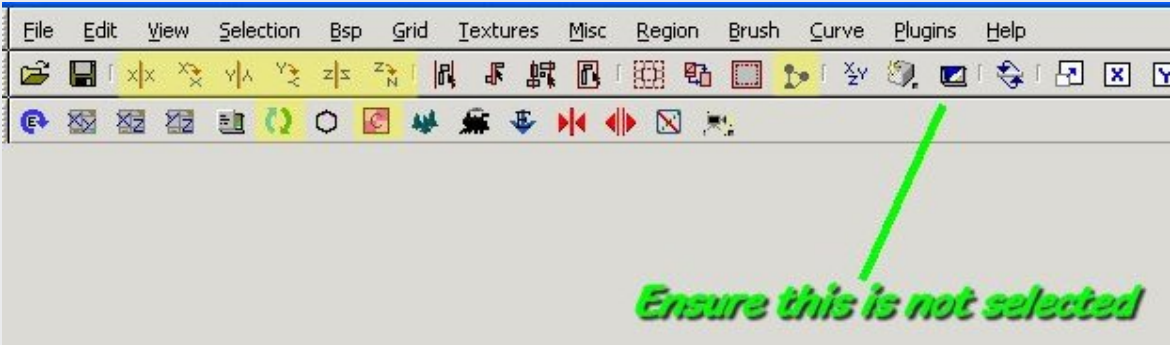
Adjust your Radiant windows so that they have roughly the same dimensions as shown below.



Maximize the radiant window if you haven't, you're going to need all the screen space you can get.

The 2D window	Overhead and two side views available, view from X Y or Z dimension. Use ctrl+tab to cycle thru the views. Use mouse wheel to zoom in/out.
---------------	--

<b>The 3D window</b>	Free movement around your creation in 3D. Right click in 3D window to enable free movement. Move mouse to look around and arrow keys to move. Right click to quit free movement.
<b>Texture window</b>	The textures used so far in your creation, plus any others you have selected ready to choose from. Click on a texture to see its name shown bottom right.
<b>Output window</b>	The log of what loads when you open a map or load up textures. Also the log of your compiler activity.
<b>Toolbar</b>	This looks intimidating, but happily you only actually use a very few of the buttons in practice. In the picture below I have coloured yellow the buttons I use - I don't use the others at all. Make sure the cubic clipping (indicated in the pic) is turned off, or you won't see all of your map in the 3D window.



[Next lesson](#)



# ET Mapping Tutorial

## Lesson 2

### Topics

#### Making your first brushes

[The building blocks of a map](#)

[Creating a brush](#)

[Moving a brush](#)

[Resizing a brush](#)

[Applying texture to a brush](#)

[Back to main menu](#)

### The building blocks of a map

[\[Top\]](#)

Imagine making something using lego bricks - the lego bricks of mapping are called Brushes. By putting brushes together you can make terrain, buildings and all the things inside them. Given enough time, and brushes, you can create a whole environment in which the players can run around shooting each other.

But unlike lego bricks where you are limited to the number, shape and colour of the bricks in your lego set, in Radiant you can create as many bricks, with whatever size, shape and colours as you want.

### Creating a brush

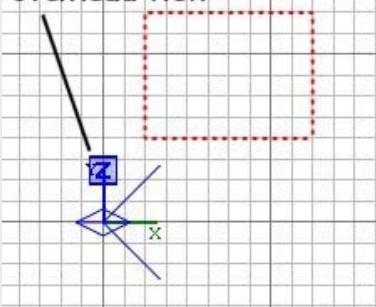
[\[Top\]](#)

Brushes are created in the 2D window.

To create a brush click and drag in the 2D window. A newly created brush is automatically selected, so it shows in a dashed red outline. To deselect your selection, press ESC. To select a brush, press shift+click. You can keep adding to a selection by selecting additional brushes. **You can delete a selection by pressing Backspace.**

You can see your brush in the 3D window - move the 3D view around until you can see your brush nice and centrally (right click in the 3D window and use the mouse/arrow keys). It will be a blue/black check (when not selected) with "Shader not found" written on it. We'll sort out some proper textures later.

The boxed blue z means we are in overhead view



The size of the brush you create is up to you, but its edges are always aligned to the grid. It is **very important to use the right grid scale** when creating brushes. You can specify the grid size by using the keys 1 to 9. Use 8-9 for large scale creation; use 5-7 for medium; use 3-4 for small. Use 1-2 only when you *really* need to get down to the tiny level. By default Radiant starts up with a grid size of 4, which is quite small. If you make large walls etc using a small grid scale, you will find it very hard to get the major structures to line up as you put walls together.

So **use the largest grid scale you can for your large scale building blocks**. It will be painful later on if you don't!

It doesn't matter for this first build-a-room tutorial, but it certainly will when your development projects are bigger.

Quick mention of scale: the grid unit measurement equates more or less to 1 inch in the real world. So I assume a player is 72 units high and make ordinary doors 84 units high to give about a foot clearance.



## Moving a brush

[\[Top\]](#)

Brushes can be moved in the 2D or 3D window. The 2D allows precise movement in any axis, while the 3D gives you a more intuitive feel for placing something just where you want it. Ensure the brush to be moved is selected. You can move multiple brushes if they are all selected.

To move a brush in the 2D window, put the cursor within the brush outline and click/drag. You can move in any axis by first changing the 2D view (cycle through the view by ctrl+tab). Generally if you have used ctrl+tab, make sure you end up in the overhead view (the blue Z will be large and in a box).

To move a brush in the 3D window, put the cursor within the brush and click/drag. The axes you can move it in will depend on your viewpoint and in which direction you are looking.

Have a go at moving the brush in each of the 3 dimensions. When you're done, leave the brush roughly around the 0,0,0 co-ordinate, ie where the XYZ axes are shown on the grid.

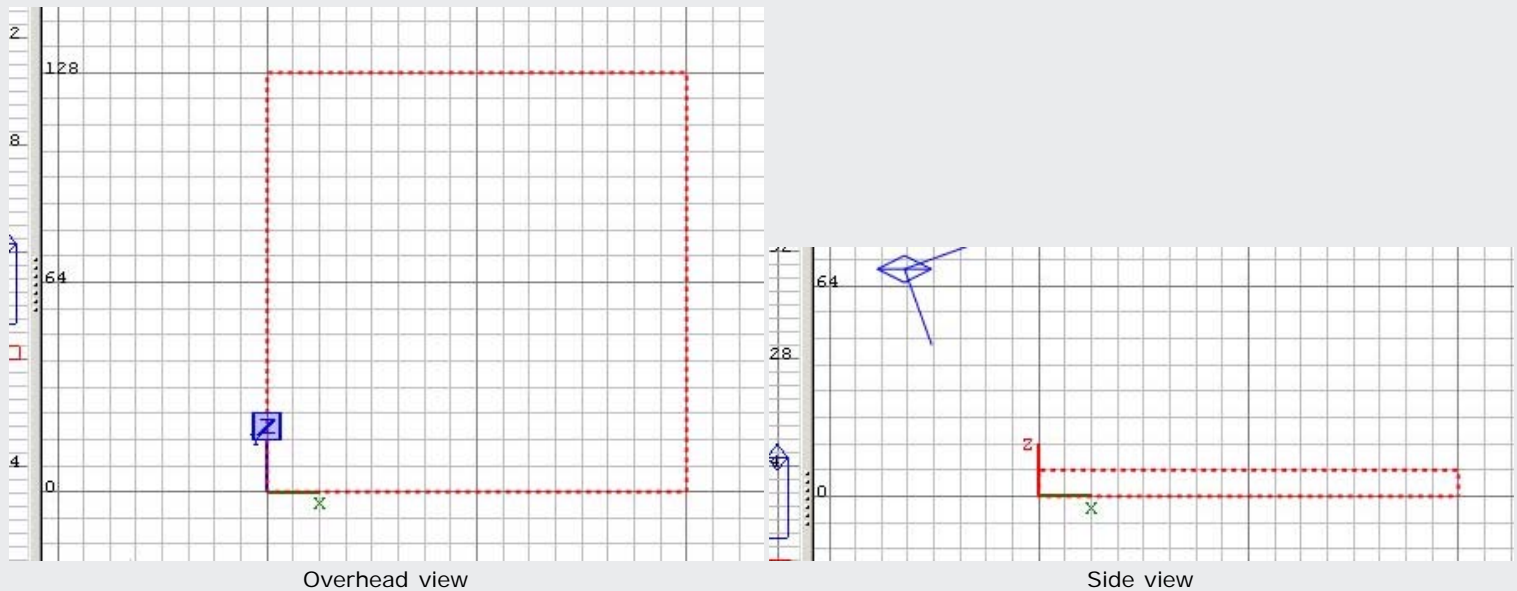
## Resizing a brush

[\[Top\]](#)

Brushes can be resized in the 2D or 3D window. The 2D allows precise movement in any axis, while the 3D is best used only when the axis for change is clearly seen and controlled via the mouse, otherwise you are likely to drag the brush into the wrong size in an unintended axis. 2D is clearest. Ensure the brush to be resized is selected. You can resize multiple brushes if they are all selected and you want to resize them in the same dimension.

To resize a brush in the 2D window, put the cursor just outside the edge of the brush and click/drag. When you drag you resize in the direction you drag in - drag away to make larger, drag inward to make smaller.

Have a go at resizing the brush in each of the 3 dimensions. When you're done, leave the brush as shown in these pictures (a flat square). On subsequent views of the map you may need to reverse a little in the 3D window in order to see the square. This is because the camera will start at the origin (0,0,0) and will be looking at the inside.



## Applying texture to a brush

[\[Top\]](#)

Textures - what's that all about then?

A texture is the name given to the image applied to the face of a brush.

A brush typically has 6 faces, all of which will have a texture applied. At the simplest level a texture on a face will be a JPG or TGA file of anything from a single colour up to an image of the Mona Lisa. Some special textures are used for the faces that don't need to be drawn, for example where two faces are flush against each other, or a face faces away from any player and will never be seen (and so need not have a visible texture applied to it).

One special texture for this purpose is called Caulk. Faces with caulk on them are not drawn, and so must never be somewhere that a player could look at it. You've probably played a map where a caulked face is accidentally visible, and you get a weird hall of mirrors effect while you look at it. So you must ensure all caulked faces cannot be seen.

**In making a map you should be trying to achieve your design aims using the least number of brushes and the least number of visible faces as possible.** One way to achieve the latter is to create your brushes initially entirely from the caulk texture. Then you "paint" the visible faces with the required texture, safe in the knowledge that all the other faces are caulked.

For example, say you created a brush to be used as floor in a room. You decide to make the whole brush a wood texture and you

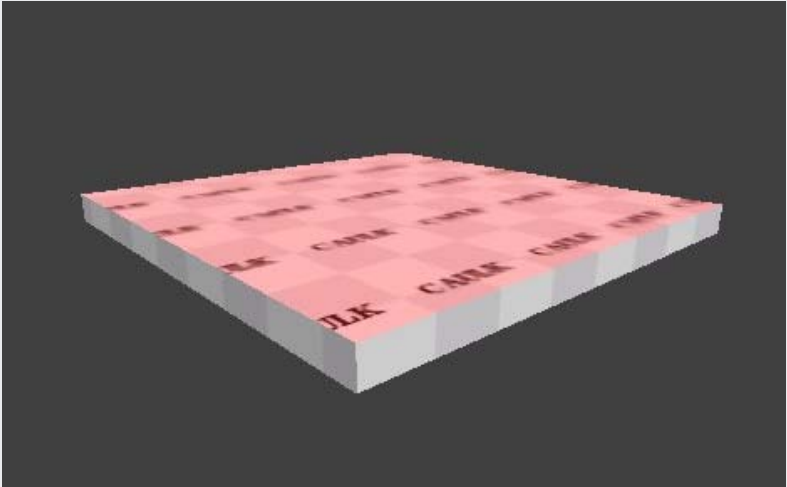
slot the brush into place. You forget to later caulk the bits you can no longer see, so you end up with 5 other faces that the engine must consider when deciding what to draw, when if they had all been caulked it would not present this overhead. It may seem like small fry, but a map like 2tanks has 8,000 brushes, and so about 48,000 faces, which translates to around 96,000 triangles to be considered and possibly drawn. As you want to try to keep the maximum number of triangles drawn to under perhaps 40,000 you need to ensure you **use caulk wherever possible**.

So let's start correctly with the flat square we've created. We're going to make that the floor. We caulk an entire brush by selecting




the brush and clicking the Caulk button.

Textures are applied in the 3D window. Point at the top face of the flat square and press ctrl+shift+click. (To cancel the face selection, press ESC. You can select multiple faces by pointing at more faces one at a time and ctrl+shift+click them into the selection. You can also deselect a selected face in the same way.)



Now we need to tell Radiant what texture to apply.



At this stage it is unimportant that your \etmain folder contains extra PK3 files, like custom maps. But as you get deeper into the mapping it becomes important that your \etmain folder does not have them. Radiant will show you the textures in those PK3s as available for you to choose. And you will not realize that the town you have lovingly created uses 25 textures scattered across lots of other PK3s, and because they won't be in your PK3 when you proudly distribute your new map, everyone else will see the ugly yellow/black squares texture all over the place - oops :(

Click the "textures" menu item. You can now choose to see the textures available under the various folder names. It's a bit of trial and error here as you go looking for the texture you'd like. For now choose Wood. All the textures in the folder will now show in your textures window. Don't worry about those that show as red/black squares.

Click on one you like; I am using wood\_m05a\_usata.

Ok, we've created a floor!

[Next lesson](#)





# ET Mapping Tutorial

## Lesson 3

### Topics

#### Making your first map

[Duplicating and rotating brushes](#)

[Placing the player start points](#)

[Defining the world co-ordinates](#)

[Compiling the map](#)

[Testing the map](#)

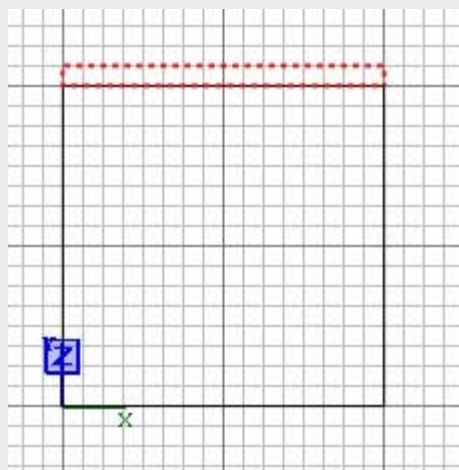
[Back to main menu](#)

### Duplicating and rotating brushes

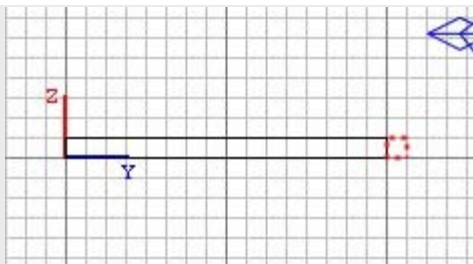
[\[Top\]](#)

Ok we have a floor, let's add a wall.

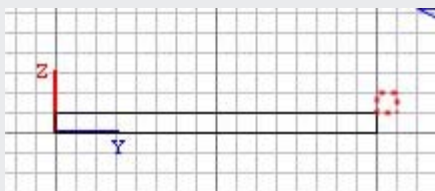
In the 2D view make sure we have the top down view, and draw a rectangle next to one side of the floor. Don't worry about whatever texture gets used at the moment.



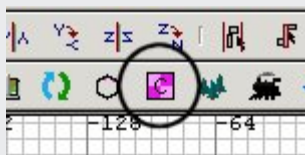
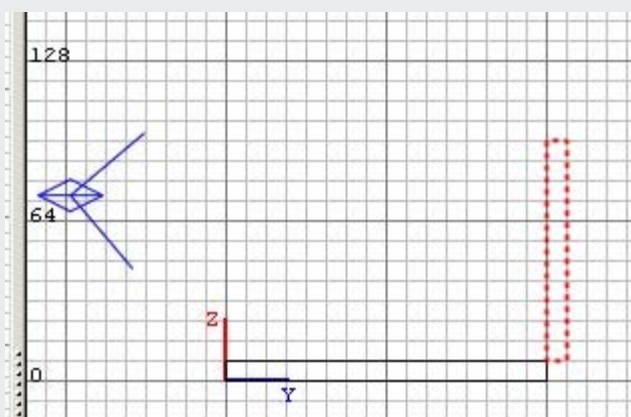
Press ctrl+tab twice so we are looking down the length of the (currently very short) wall.



Lift it so that the bottom edge of the wall touches the top edge of the floor. (Click inside the selected area and drag up).



Make the wall higher by putting the cursor above the selected area and dragging upwards.

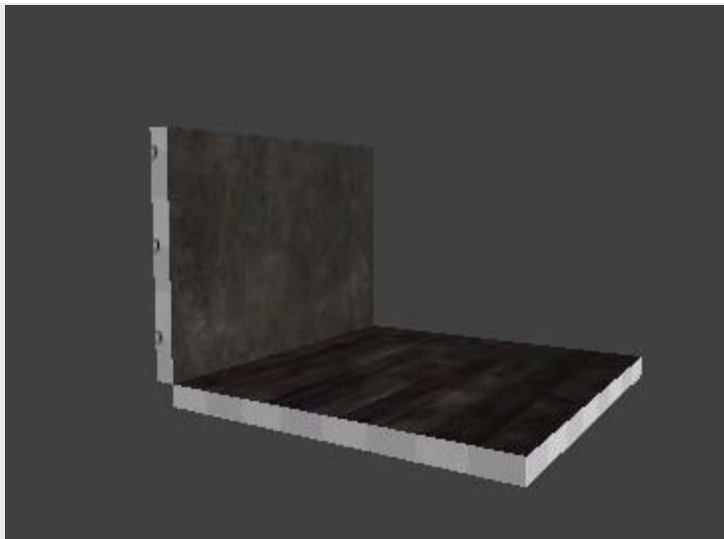


Ok we have a serviceable wall. Now caulk it. and press ESC to deselect the brush.

As we are going to see only the inside of the room, we only need to texture the inner wall.

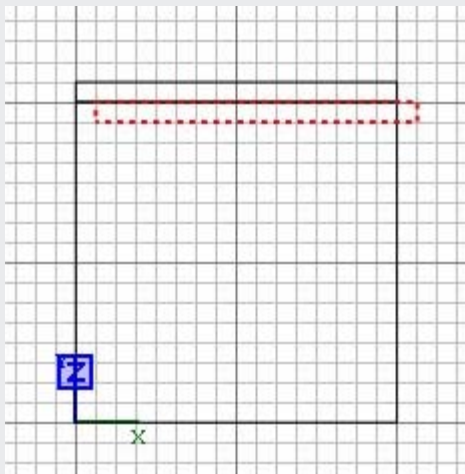
I am going to use a drab stone effect: click Textures, Town, Town Wall. We get the town wall textures shown.

In the 3D window, select the inner wall face (ctrl+shift+click) and click on texture town\_c61a and press ESC to deselect the face.



Now make sure the 2D is topdown, and select the wall brush (shift+click).

**To duplicate a selection: press SPACE BAR.**



We want to rotate the new wall 90 degrees around the Z axis. Press the button shown, to do this.

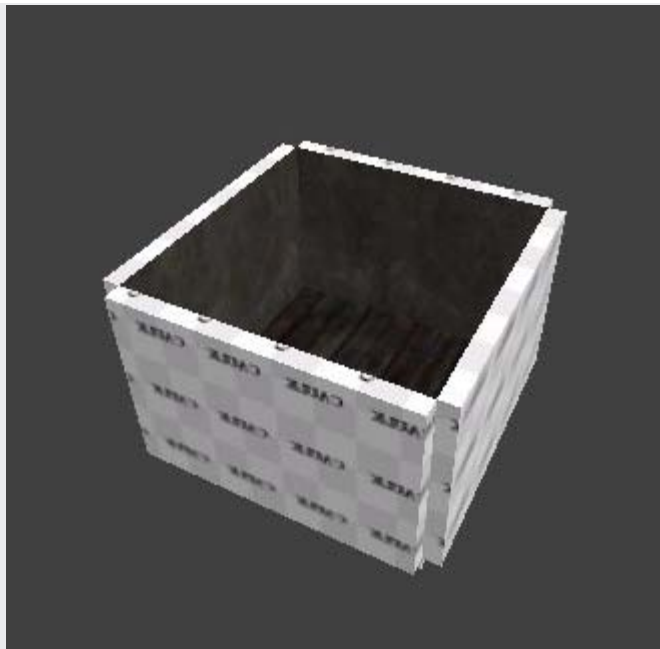
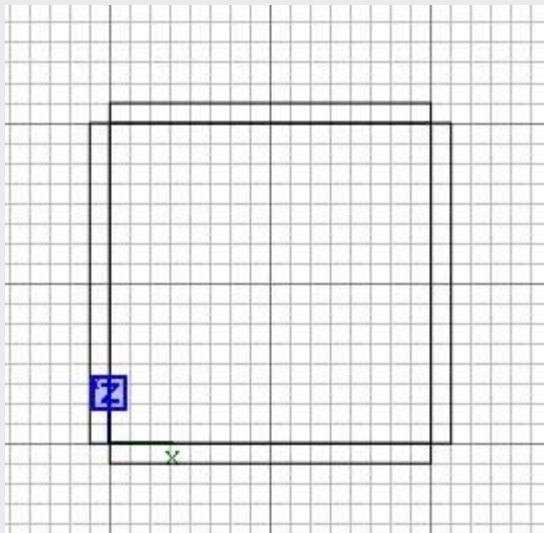


In the 2D window, drag the newly rotated wall against the right edge of the floor.

Duplicate the new wall, rotate 90 degrees again, and drag it to the bottom edge of the floor.

Duplicate the new wall, rotate 90 degrees again, and drag it to the left edge of the floor.

Press ESC to deselect.

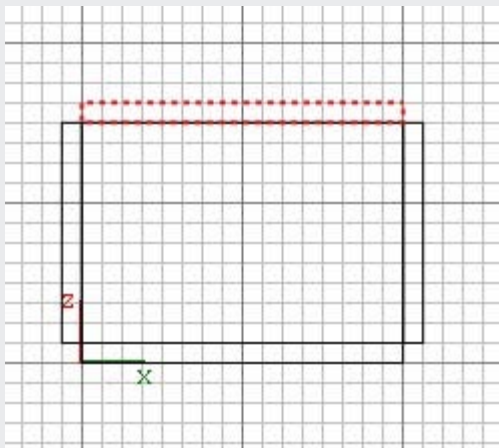


Now we just need a ceiling.



Be aware that in the 2D view when you try to select a brush, Radiant will select the uppermost brush you click on, if there is more than one under your cursor. This can be misleading, as you can believe you have selected the floor when actually you have selected the ceiling. Another method of selecting is to repeatedly use shift+alt+click, which will drill down through each brush under your cursor, selecting one after the other.

Select the floor, press ctrl+tab to get a side 2D view, and duplicate the floor. Move it to the top of the walls, then caulk it.



Deselect it. Move your 3D viewpoint inside the room and look at the ceiling face - select it with ctrl+shift+click. We'll give it a sky effect. Sky textures provide light, so we won't need to put explicit light sources in the room.

Click Textures/FuelDump and pick the FuelDumpSky texture.

Deselect the face - you have made a small room with a sky ceiling.



We will put the player inside the room. He will only be able to see the inside of the room. The sky is actually a ceiling. Everything outside the room is in the volume space called void. You must ensure when making a map that the player could never trace a path to the void. That is, the 3D space that encloses the players must be airtight, with no gaps that lead to the void, even if a player could never see it. If you do, your map compile will fail with a "Map is leaked" error msg.

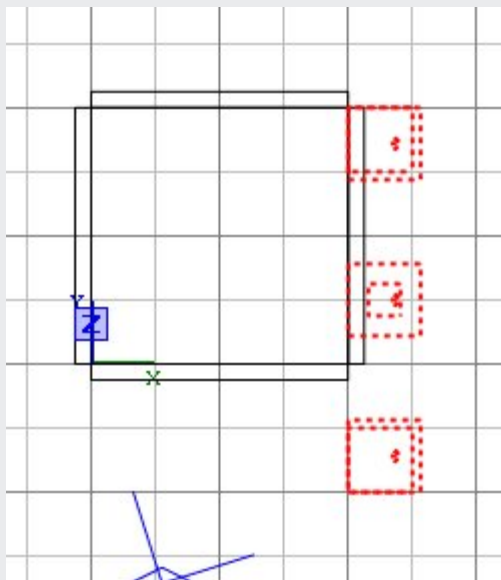
## Placing the player start points

[\[Top\]](#)

You need to include a minimum of 6 particular entities before a map can function. **An entity is a game component** other than an ordinary brush, but you can convert brushes into special purpose entities which you'll see later on.

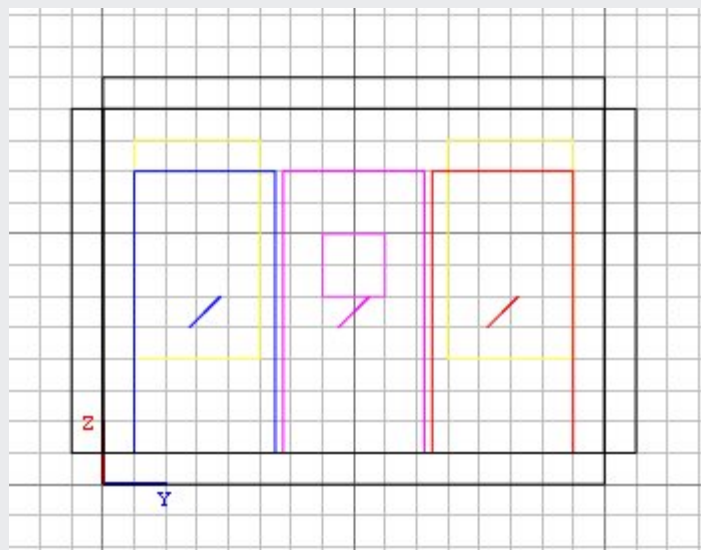
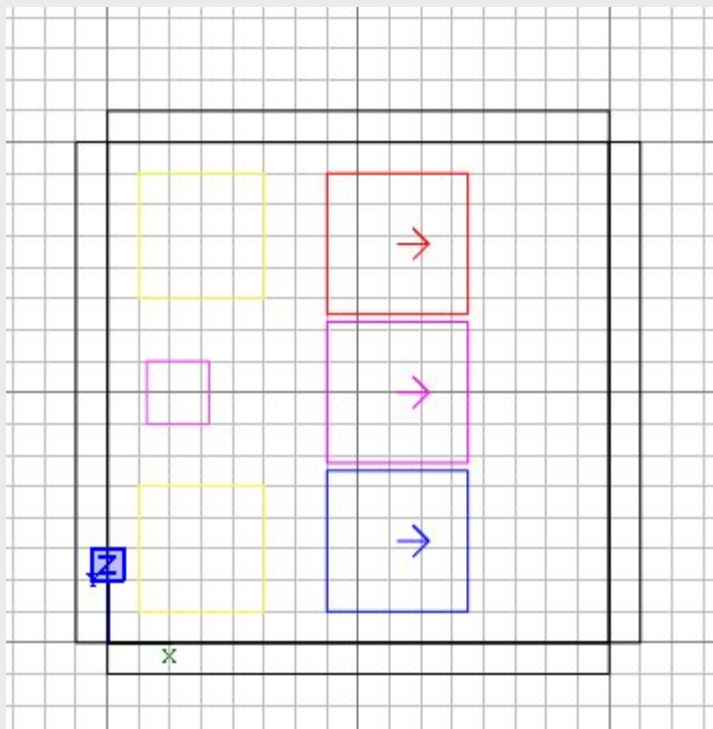
To save the time involved in inserting them, download (right-click) this [players.zip](#) file which has them already defined. Extract the players.map inside and put it into your \maps folder.

To include the contents of that file into your map, click File/Import and select "players.map". The 6 entities arrive in your map, already selected.



Deselect them, then select each in turn and move them inside the room. Do not have any part of an entity sticking into a wall, floor or ceiling, as if you do the map compile will fail with the "Map is leaked" error.

This is a little cramped with this tiny example room; with a large map there is plenty of space :)



The entities you have just placed are:

<b>Blue box</b>	Allied player spawn point - you need 32 of these per Allied spawn location. One will do for testing.
<b>Red box</b>	Axis player spawn point - you need 32 of these per Axis spawn location. One will do for testing.
<b>Yellow box near the blue box</b>	The clickable flag for the allies spawn point shown on the command map - one per spawn location
<b>Yellow box near the red box</b>	The clickable flag for the axis spawn point shown on the command map - one per spawn location
<b>Large mauve box</b>	Viewpoint within map until player selects a team
<b>Small mauve box</b>	The entity that connects to your script - you need exactly one of these in a map

Defining the world co-ordinates

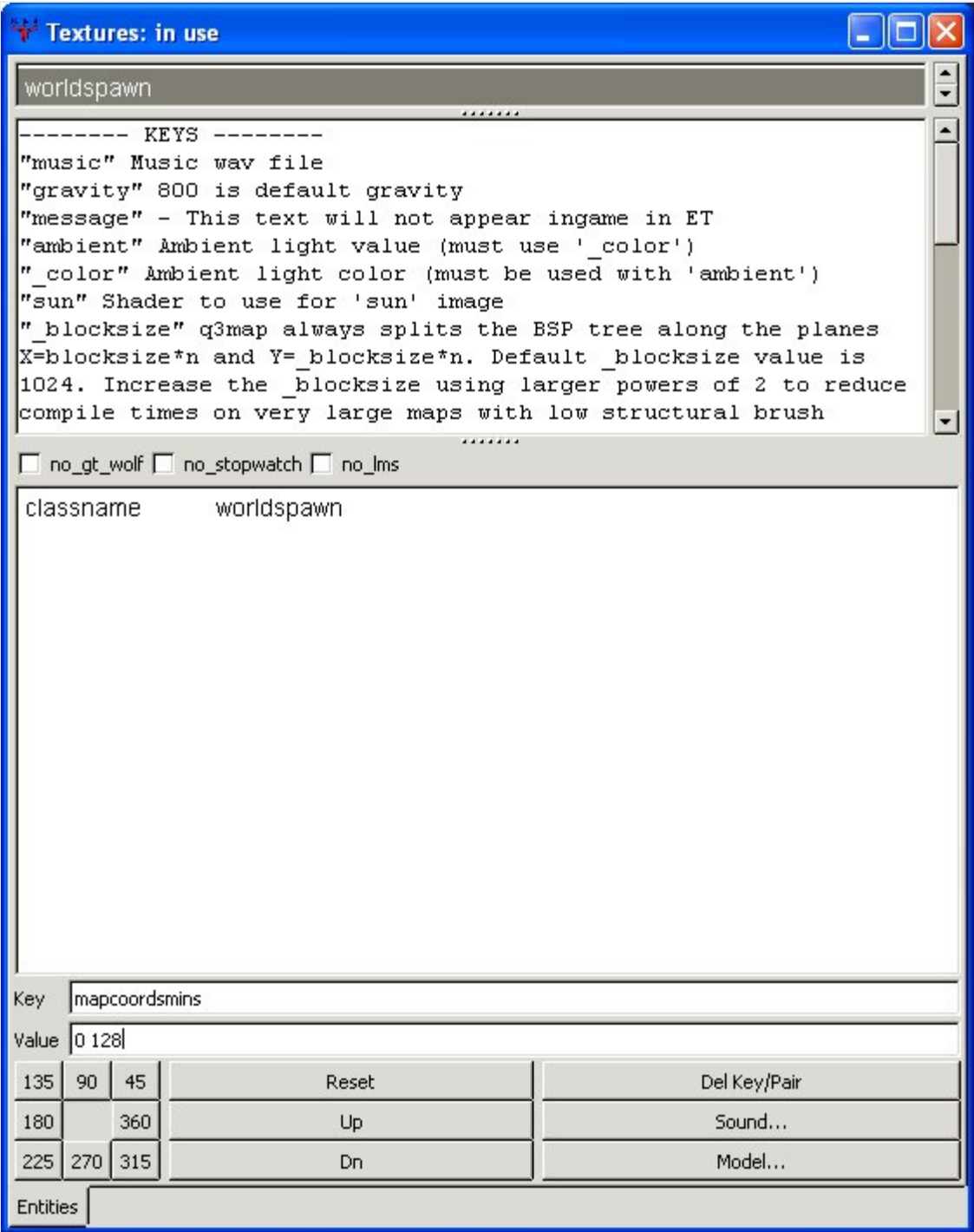
[\[Top\]](#)

You need to tell ET how big your map is now, so that it can position the icons on the command map in the right place.

Select any ordinary brush, like the ceiling brush. Don't select an entity such as a player spawn point at this time, as to access the worldspawn values you have to have an ordinary brush selected.

Press N. This brings up the Entities window.

You'll see the worldspawn entity. You would see this for any regular brush you had selected, it's just a way to set worldspawn values which tell ET some basic map stats. Enter "mapcoordsmins" as a Key, and "0 128" as the Value and press return. This is the top left co-ordinate of your map.



Then enter "mapcoordsmaxs" as a Key, and "128 0" as the Value and press return. This is the bottom right. The mins and maxs must define a square, regardless of the actual shape of your map.



Close the entity window and press ESC to deselect the brush.

Compiling the map

[\[Top\]](#)

It's about time you saved the map to disk. Do a file save - I am using the name "tutorial" for this map example. Ensure the file is saved into your \maps folder.



Save your work often. It's very painful to have to redo a lot of creative work if Radiant crashes (which it might) or you have a power cut. I also save my work after a fair amount of work to a copy of the map, eg tutorial\_051124a.map, tutorial\_051124b.map, etc. Sometimes an error might creep in and you'll need to recover to a map file a few versions earlier.

Click BSP menu item and pick the longest compile option, the one that starts **Simulate old style**.

3 DOS windows will open one after the other, and you will see a mirror of their output whizz past in the Radiant output window.

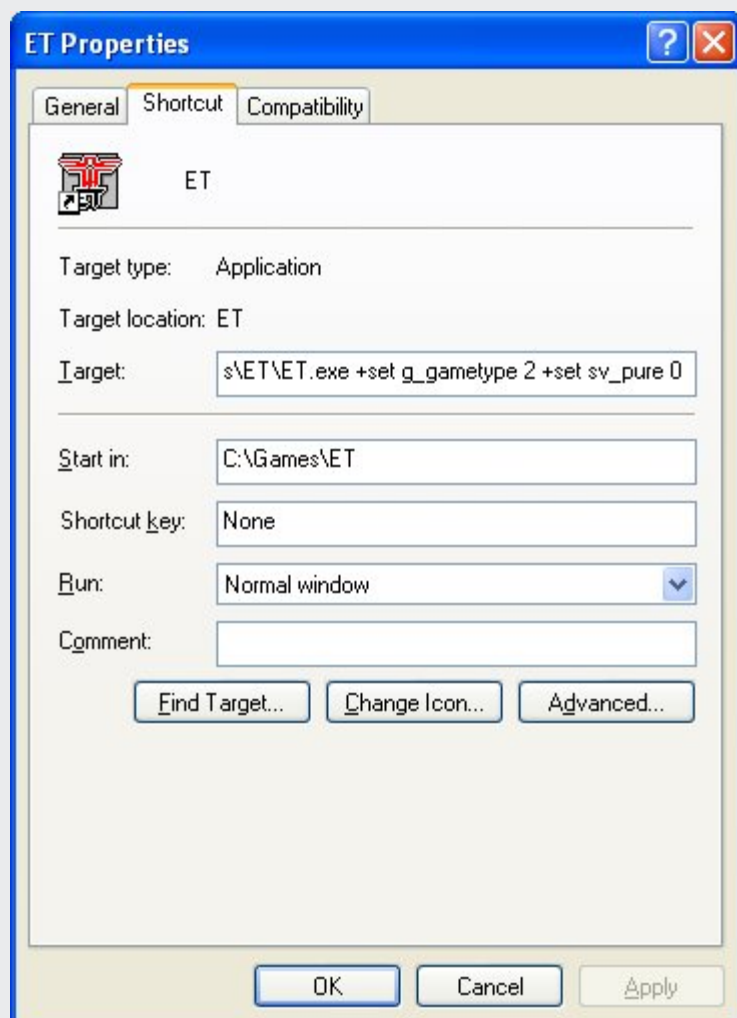
When the compiles finish the last lines will say **Disconnecting** and **Connection closed**.

If you haven't made an error and your compile worked, you now have a tutorial.bsp file created in the \maps folder.

## Testing the map

[\[Top\]](#)

You will need to add a couple of parameters to the startup for ET. Add the text "+set g\_gametype 2 +set sv\_pure 0" to the target string. If you are copying from this web page, paste the text into a flat .txt file then copy and paste into your shortcut from there. Otherwise you may paste some non-printable chars which will stop them from working within your shortcut. The symptom for that would be ET being unable to find your map when you try to test it, when you know it is there.



Run ET, and when the game comes up press ESC and go into the console (tilde key). Type "\devmap tutorial" - if you didn't use tutorial as the map name, use your map name in place of "tutorial".



**Revel in being able to run around in your own (very tiny) map!** Don't worry about the command map, it will only be a large black area edged in orange for now.



[Back to the main menu](#)



# ET Mapping Tutorial

## Lesson 4

### Topics

#### Creating an environment

[Making a large volume](#)

[Hull caulking](#)

[Snowy ground](#)

[Back to main menu](#)

#### Making a large volume


[\[Top\]](#)

By managing to achieve a fully workable, albeit tiny, map in the previous lesson, you have actually got over a significant hurdle. Much of what follows will be easy to assimilate now that you have acquired the knowledge to get the guts of a map working.

Our first room was tiny. We will expand it to something rather larger, convert it into an "outside" environment to contain some buildings, and then make a building to put in it. At the end of this second tutorial you'll be able to run around inside and outside a building that contains a few rooms.

Run Radiant if you haven't already.

Open the tutorial.map.

 You may have spotted on the Preferences tabs that you can get Radiant to open the last map you were working on. Don't do it. You can mangle your map so that it causes Radiant to crash on opening it, and if it's set to open it on startup it can get confusing and tiresome trying to sort it out.

You will notice that the textures window shows by default all the textures used in your map. As you get more and more textures in your map, that list will get longer. This brings a useful editor feature into use: Click Textures/Textures Window Scale and select 25%. That way you can see more textures more easily. There will be times though that you can't easily read the texture names because the text is overlapped with the next texture. Set the scale to 100% or 200% to overcome that. In other words, set the scale to whatever you are comfortable with and that meets your needs.

Let's make this tiny room into a large outdoor container for a building we will then create inside it.

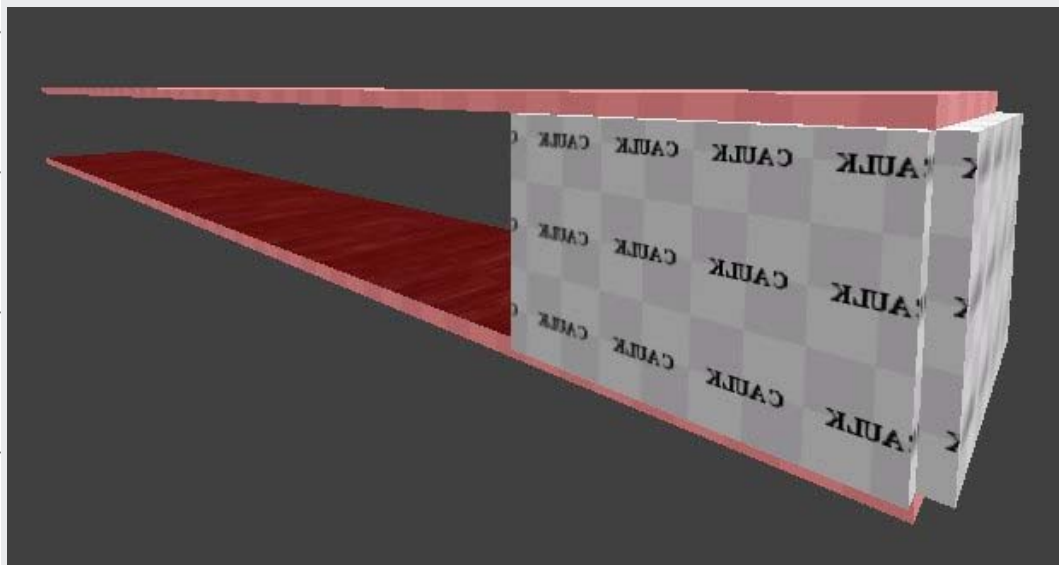
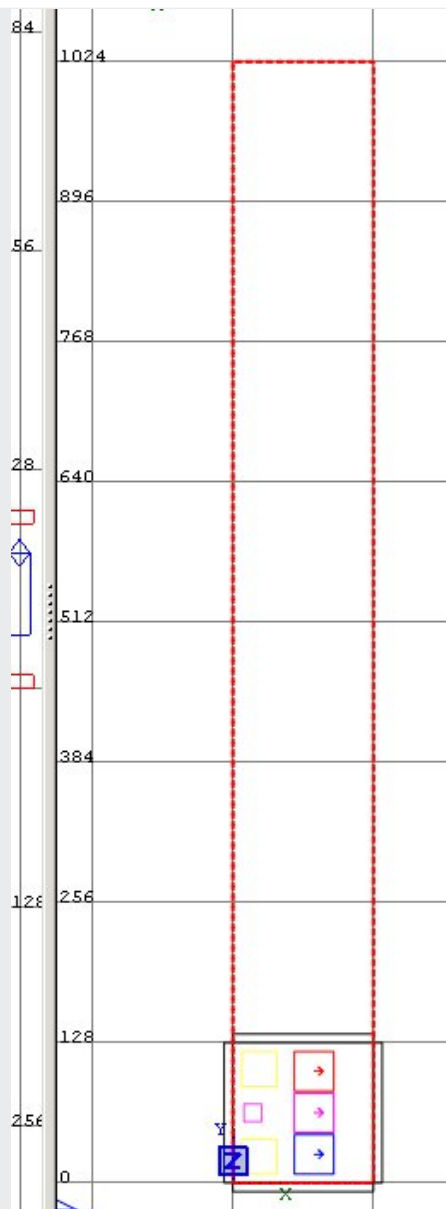
It might be quicker to start afresh with making a large cubic space - but we will instead transform our tiny room into the large volume because it demonstrates a number of handy techniques, including hiding, revealing and deleting brushes, plus manipulating multiple brushes simultaneously.

In the 2D window, top down view, shift+alt+click on the ceiling of your room, twice. This will end up selecting the floor (yes!) as you will easily verify by either pressing ctrl+tab or looking in the 3D view.

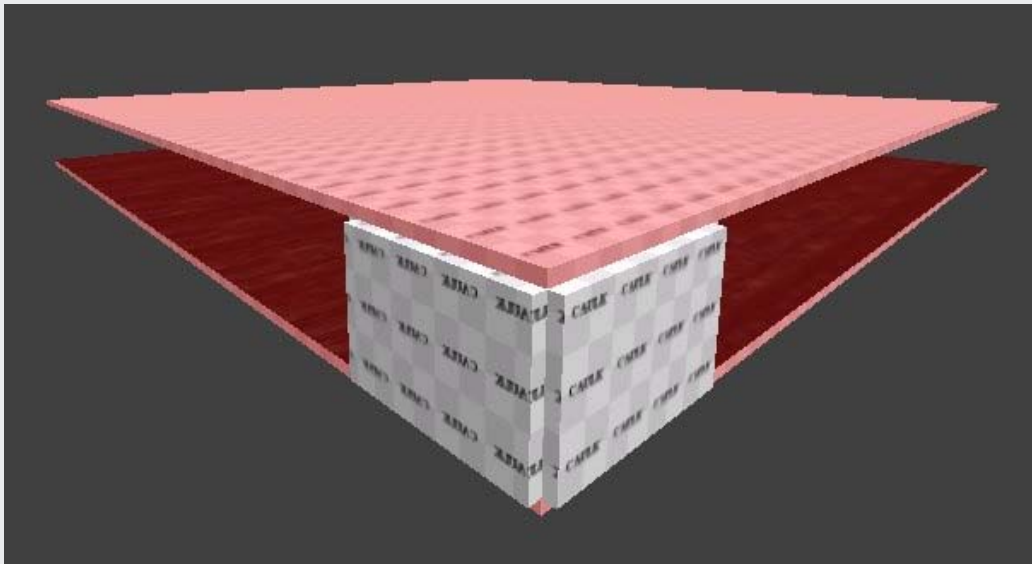
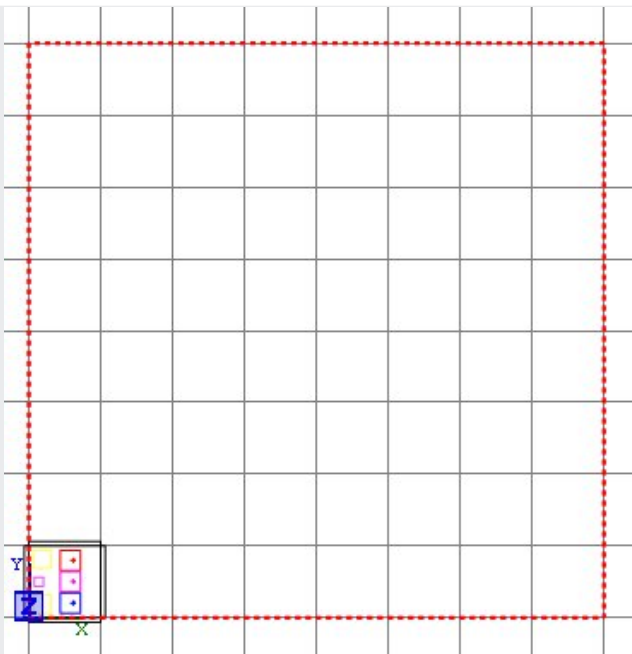
Still in the overhead 2D view, shift+click on the ceiling area again. This will add the ceiling to your selection set, ie you now have the ceiling and floor selected.

Press 8 to get a nice big grid scale.

In the 2D view, click outside the selected area in the upper part of the window, and drag the selection larger in that direction, until the selection reaches the 1024 Y-axis mark.

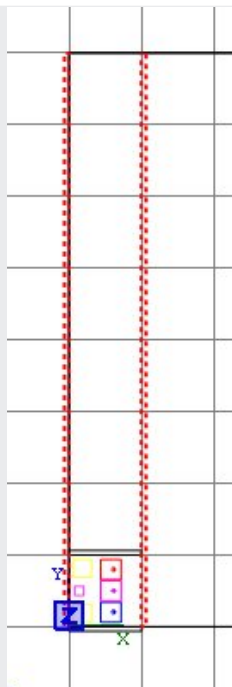


Now click in the 2D window to the right of the selection and drag it to the right, making the selection a square of dimension 1024\*1024.



Deselect the brushes. Now we want to stretch the walls to match. Put the cursor over the right hand wall in the 2D window and press shift+alt+click twice. The first click will select the ceiling, but the second will select the wall that we want.

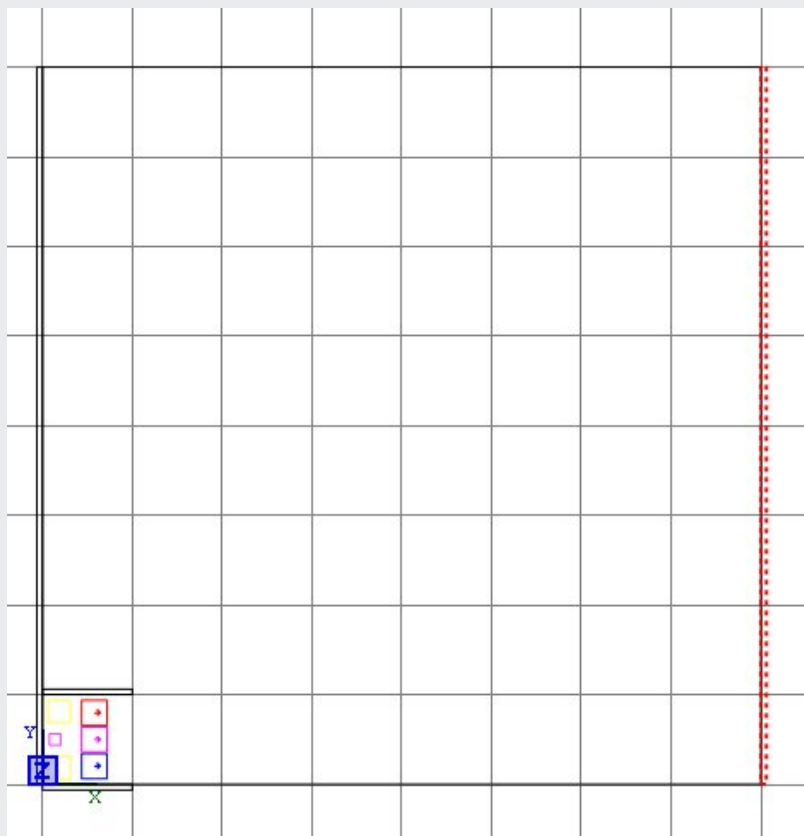
Then shift+click the left hand wall, so now we have both side walls selected. Put the cursor above and between them and drag upwards until they are 1024 long.



Press ESC. We want to move that right hand wall over to the far right, but the ceiling is in the way and the shift+alt+click is a little tiresome, so we'll employ another technique to make it easier.

Select the ceiling. Press **H** to **hide** the selection. Now we can get at the walls without always having to drill down through the ceiling.

Select the right hand wall. Put the cursor within it in the 2D window and move it over to the far right.



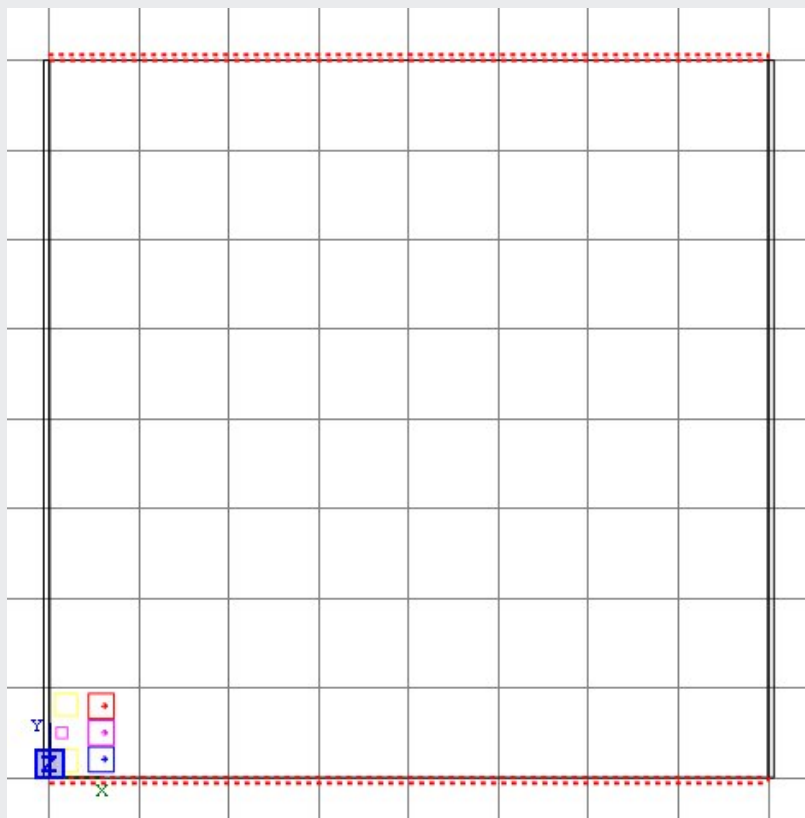
Press ESC - select the remaining 2 short walls and **delete them by pressing BACKSPACE**. (If you accidentally select say the floor when trying to select a wall, just repeat the click and the wrongly selected brush will get deselected again.)

Now select the 2 big walls - the quickest way here is probably to click on them directly in the 3D window.

Duplicate them by pressing the Space Bar, then rotate them through 90 degrees by pressing the key shown in the picture:



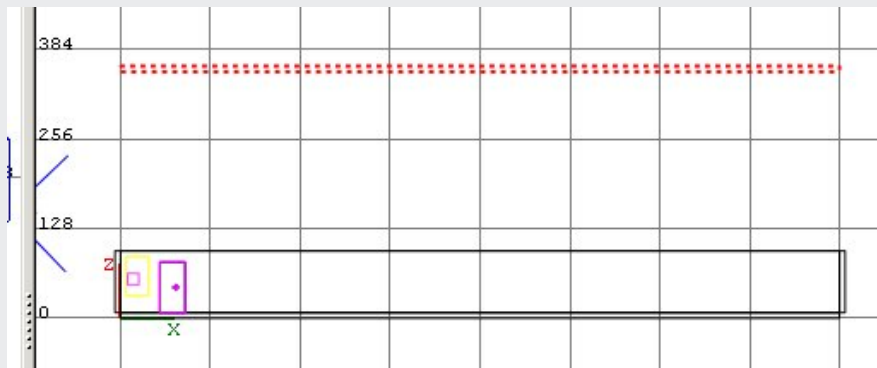
Put your cursor within one of the selected brushes in the 2D window, and drag the walls into place to complete the square around the floor.



Press ESC to deselect and finally press **shift+H** to reveal all hidden brushes (ie the ceiling).

Ok we have a wide area now, but the sky is pressing down a bit, so we need to give ourselves some more headroom.

Select the ceiling brush. Press ctrl+tab to get a side view in the 2D window. Move the ceiling up a couple of grid lines.



Press ESC. Select all 4 walls (easiest by shift+clicking them in the 3D window).

Return to the 2D window and put the cursor above the walls, then drag them up to meet the ceiling.

Press ESC. We've now made ourselves a larger volume, and we're going to make it represent the outdoors.

Now is as good a time as any to set the new worldspawn values. Select any normal brush and press N.

Click on the "mapcoordsmaxs" line in the table, and then replace "128 0" in the Value box with "1024 0". Press return.

Click on the "mapcoordsmins" line in the table, and then replace "0 128" in the Value box with "0 1024". Press return. Press ESC.

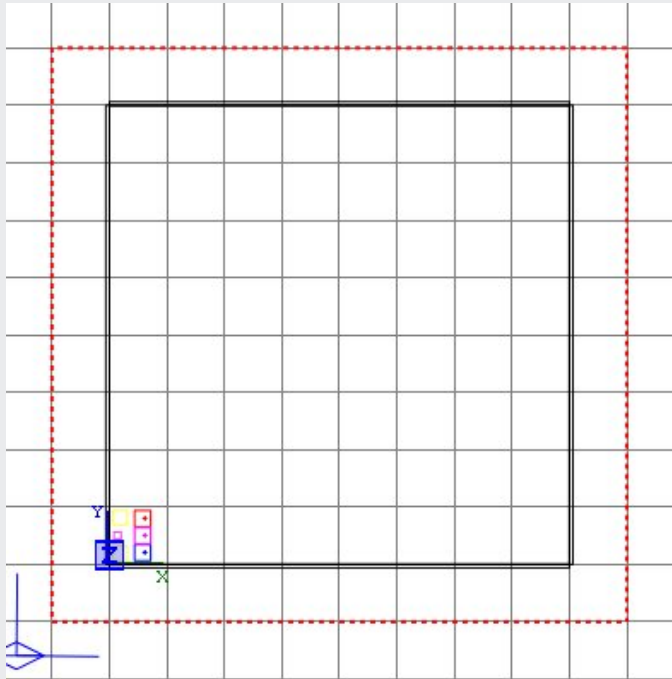
## Hull caulking

[\[Top\]](#)

This next bit is not strictly necessary, but it will aid you when your map grows larger and more complicated. Because you will have a lot of caulked brushes in your 3D view, it can be confusing trying to spot what are supposed to be the outer boxes that contain everything. There is another purpose which I will cover later on.

So we will apply another texture to the walls of our resized box, to reflect its role as the container of the play environment within it. It will also demonstrate one of the most usual ways to select multiple brushes.

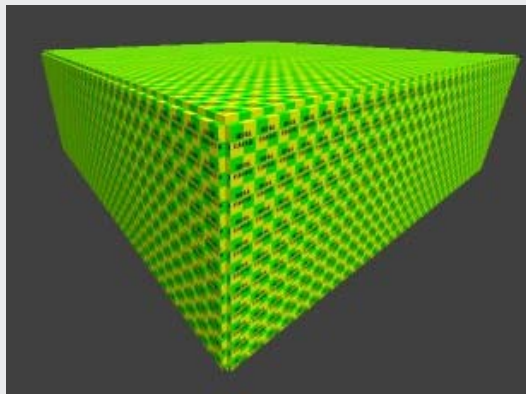
In the 2D view, make sure you have the top down view, and create a brush that envelopes the entire cubic creation made so far. Don't worry about what texture it is nor how high the brush happens to be (as seen in the 3D window).



Somewhere in the 2D window, doesn't matter where, right click. Choose Select/Select Complete Tall. This will select all brushes that are completely within the 2D box you've just drawn, even if they happen to extend beyond it, upwards/downwards, in the 3D view. It also deletes the brush we had created because it was created to define the set of brushes we wanted to select.

Now we want to make the Hull Caulk texture available to us for picking. So click the Textures menu item, then Common. You'll see lots of chequered multi-coloured squares in your textures window. You might want to set the texture scale to 50% or 100% to make it clearer (Textures/Texture Window Scale). You might also want to resize the textures window for a minute to help you see what you've got.

Find the Yellow/Green box labelled Hull Caulk and click it. Then press ESC and your box will now look like this:



## Snowy ground

[\[Top\]](#)

In the 3D view, go inside the box - we need to restore the interior textures.

Shift+ctrl+click the ceiling. Click Textures/skies/skies\_sd (note you want the **skies** that has a subordinate skies\_sd menu) and click the sd\_siwasky texture in the textures window. The ceiling will now have red/black check - don't worry about it.

Shift+ctrl+click the floor. Click Textures/fueldump and click the snowfloor texture (don't worry about the Hong Phong text).

Shift+ctrl+click a wall. Right-click so you can rotate the view in the 3D window, and shift+**alt**+ctrl+click the other 3 walls. Then right-click to get your arrow cursor back.

Click Textures/battery\_wall and click wall03\_mid texture. Press ESC to deselect the faces.

Save your work and compile the map.

Run ET to check how your work is looking so far - with luck it looks like this:



Run around a bit. Notice that you are making snowy footstep sounds? That's pretty neat, how does the program know that a messy white/grey texture painted on the ground should sound like snow? The answer is "shaders", which we'll cover later on.

[Next lesson](#)





# ET Mapping Tutorial

## Lesson 5a

### Topics

#### Making a building outline in your environment

[Making an L-shaped outline](#)

[Back to main menu](#)

#### Making an L-shaped outline

[\[Top\]](#)

I have chosen an L-shape structure to demonstrate the best practices for creating joined walls, with the aim of using the least number of brushes and visible faces.

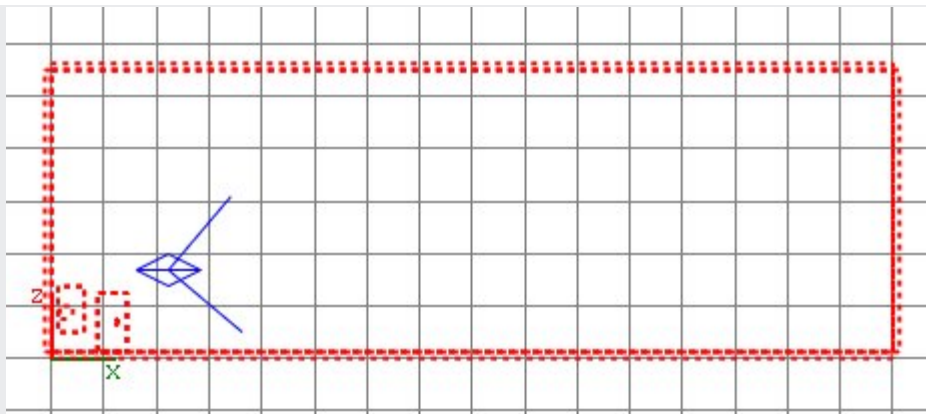
I am now assuming you know how to create, caulk, duplicate, move and resize brushes - please refer to earlier lessons for a refresher if you don't remember.

Run Radiant. Open the tutorial map. Leave the grid size at the default 4 for the moment. Press ctrl+tab to see the side 2D view. We are going to move the environment down a little, in order to make the upper face of the floor run along the 0 (zero) Z co-ordinate. This will make it easier when we are creating new brushes, because they will sit just on the ground by default, rather than just through it.

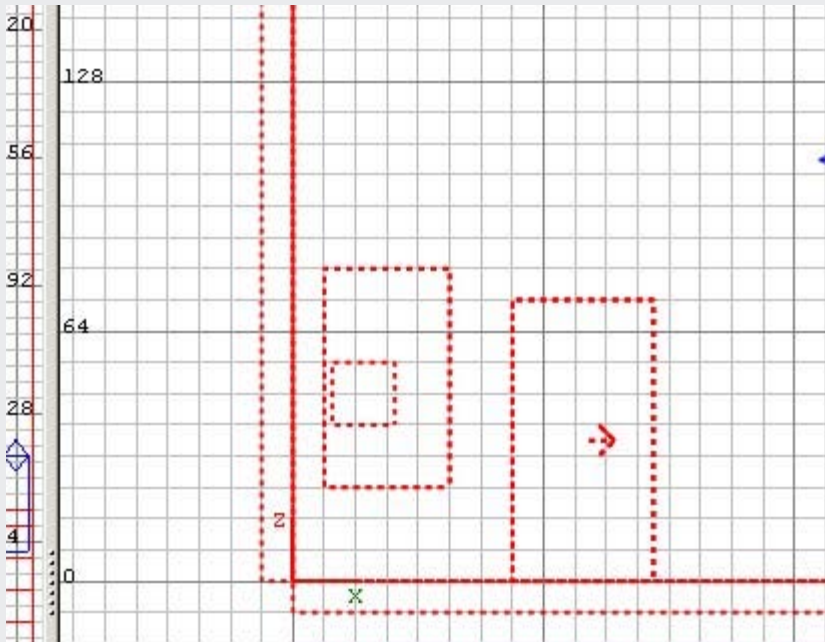
Draw a box around the whole environment (mousewheel zoom out to see it all if needed), and either right-click and Select/Select Complete Tall or click the indicated button, whichever you feel comes more naturally. If you can't see everything in the 2D window you can **scroll the view by right-clicking it and dragging around**.



You should see this:

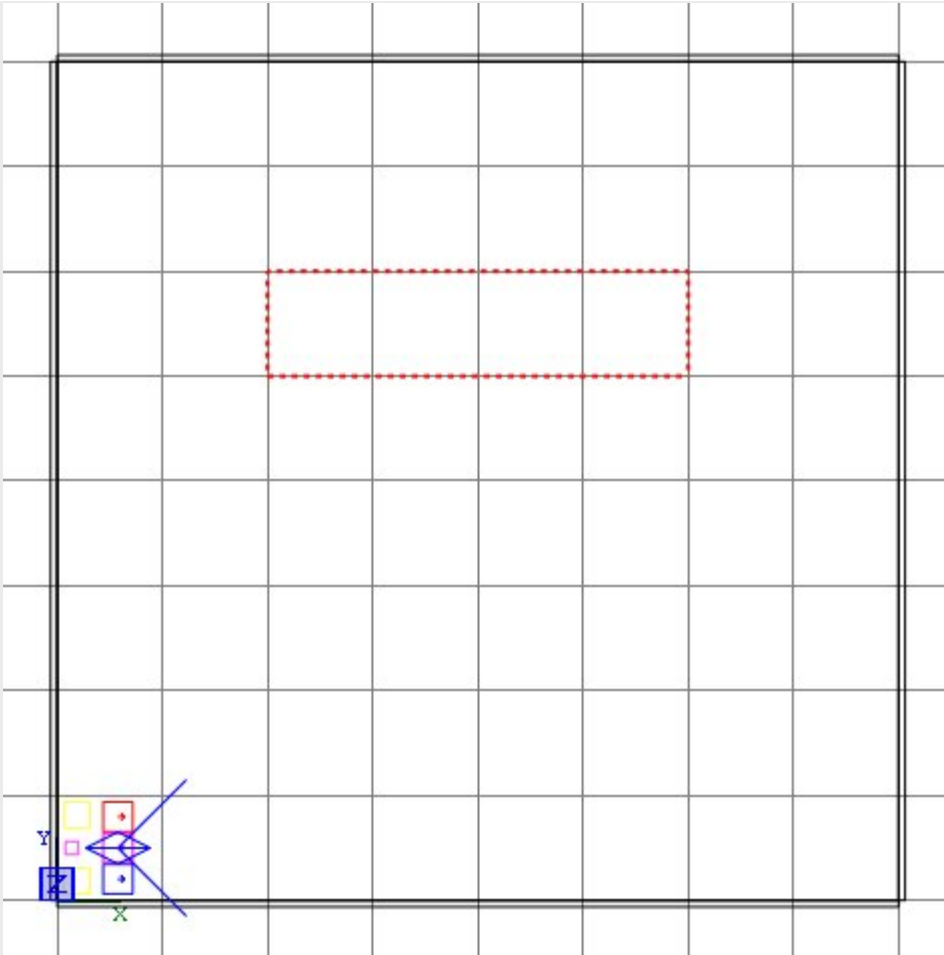


Zoom in so you can see the grid, and move the whole lot down one grid notch.

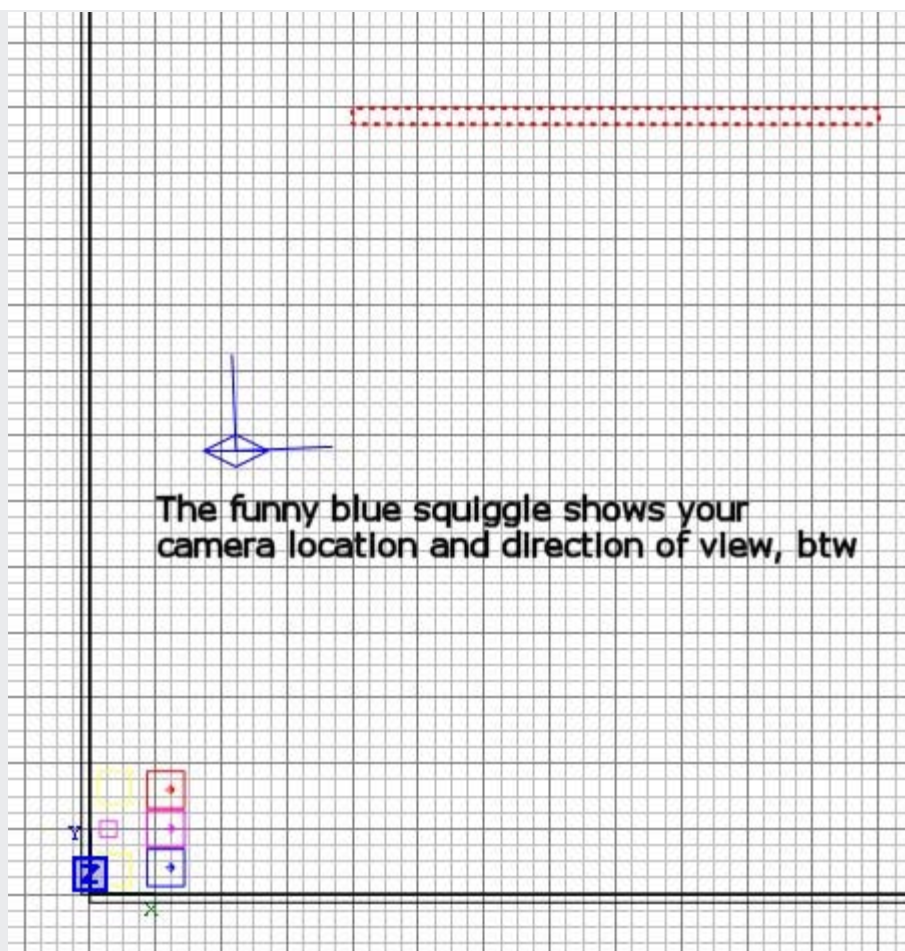


Press ESC. Press ctrl+tab twice to get the overhead view, and zoom out so you can see the whole area again. **Set the grid size to 8.**

Draw a brush as shown below. Cycle thru ctrl+tab to confirm the brush is sitting on the floor level and is 128 high. If it isn't (because Radiant remembers the last brush manipulation you were doing) adjust it so that it is.



Caulk it. **Change the grid scale to 5**, and reduce the width of the wall by putting the cursor under the selection and dragging up. You may want to zoom in a little to clarify the grid lines for yourself.



Press ESC. In 3D view, select the face that will be the inner wall - it's the south facing wall that faces back to where the player start points are.

Apply Textures/chateau wood\_test texture, or some other wall texture if you like. Press ESC.



In 3D view take yourself around to the outside face and apply an exterior texture. I'm using Textures/town/town\_wall church\_c01dm. Press ESC again to deselect the face.



Now select the wall brush and duplicate it, then rotate the new wall by 90 degrees, in the usual way...



...for making a wall at right-angles to the previous wall, and position it as shown.



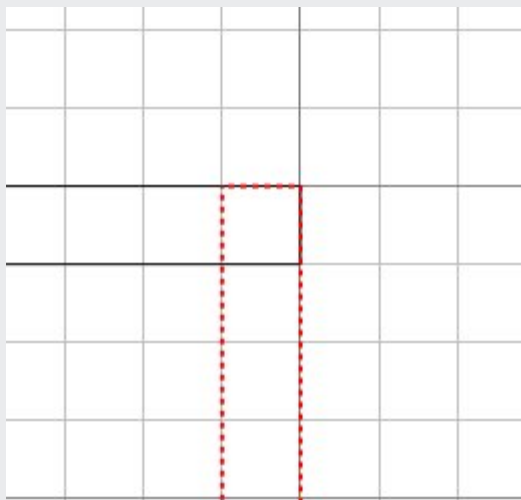
What you have here is the **wrong** way to make adjacent walls. We will correct this in a minute, but I'm doing this to show you why you should avoid this construct when possible (sometimes it will be unavoidable).

With walls butted up like this, the face **A** will need an outer wall texture applied (making 3 outer faces so

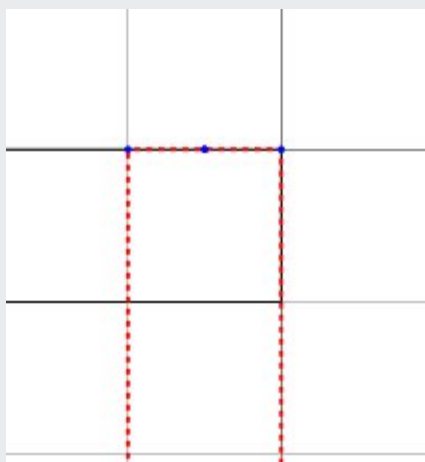
far), and the face **B** will overlap some of the inner texture applied to the other brush, which is wasteful because it means a face is textured for its full length yet part of it cannot be seen.

There is a more efficient way to arrange the two walls:

Move the second wall up one notch so that the brushes overlap.

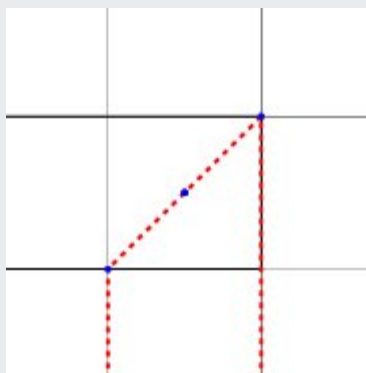


Press **E** to bring up the **edges** markers.



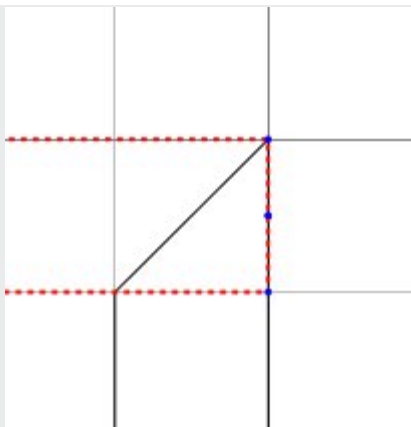
The little blue dots mark the edge points we can manipulate.

Click on the left blue dot and drag it one notch down.

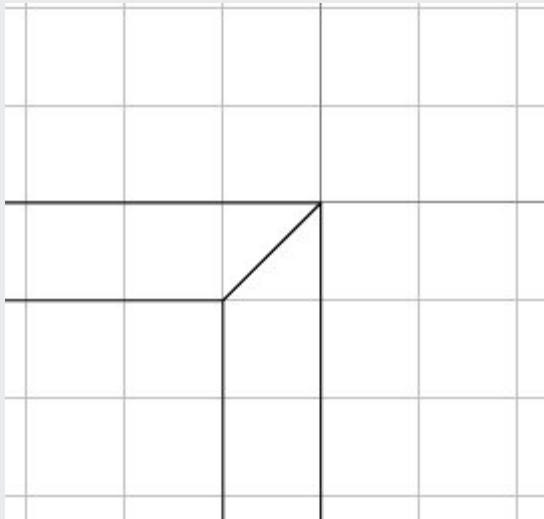


Press shift+alt+click on the other wall brush to select it, and press **E** to show its edge points.





Click on the bottom blue dot and drag it left one notch. Press ESC twice to turn off edge point display and deselect the brush.

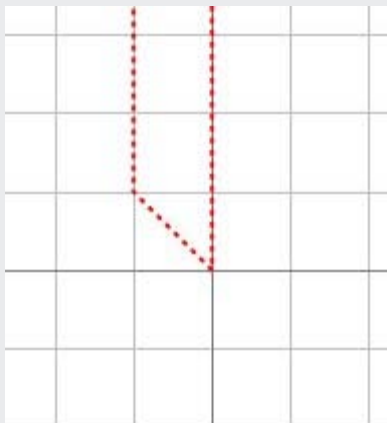


You can see we have made a nice join now, that means we only need 2 outer faces and there is no wasted display of inner faces because there is nothing partially overlapping a textured face.

We're going to this trouble, when we hadn't for the tiny room, because the outer faces are going to be visible so it isn't ok to leave chunky gaps on the outside faces - it's all got to join and seal so that people outside can't see ugly joins or see through gaps, and vice versa.

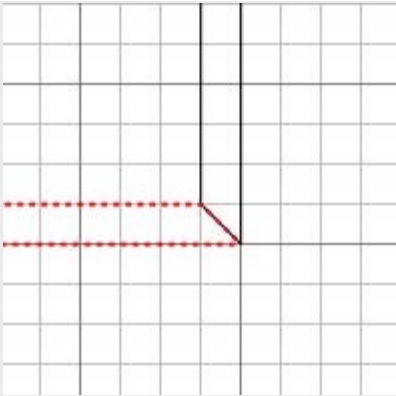
We'll create more walls next to complete the building outline.

Select the second brush and angle its bottom end using the edge points (you may want to first select the main area ceiling and hide (H) it).

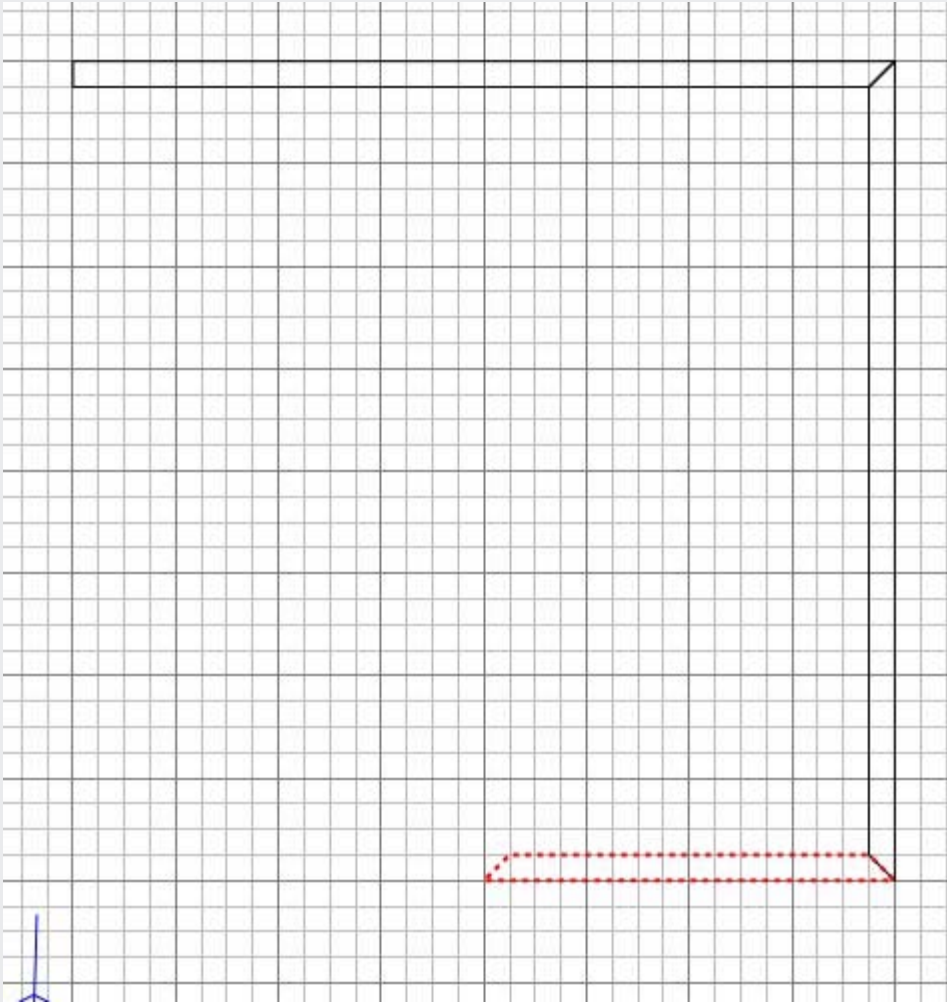


Duplicate the brush, rotate it 90 degrees and attach it to the second brush:

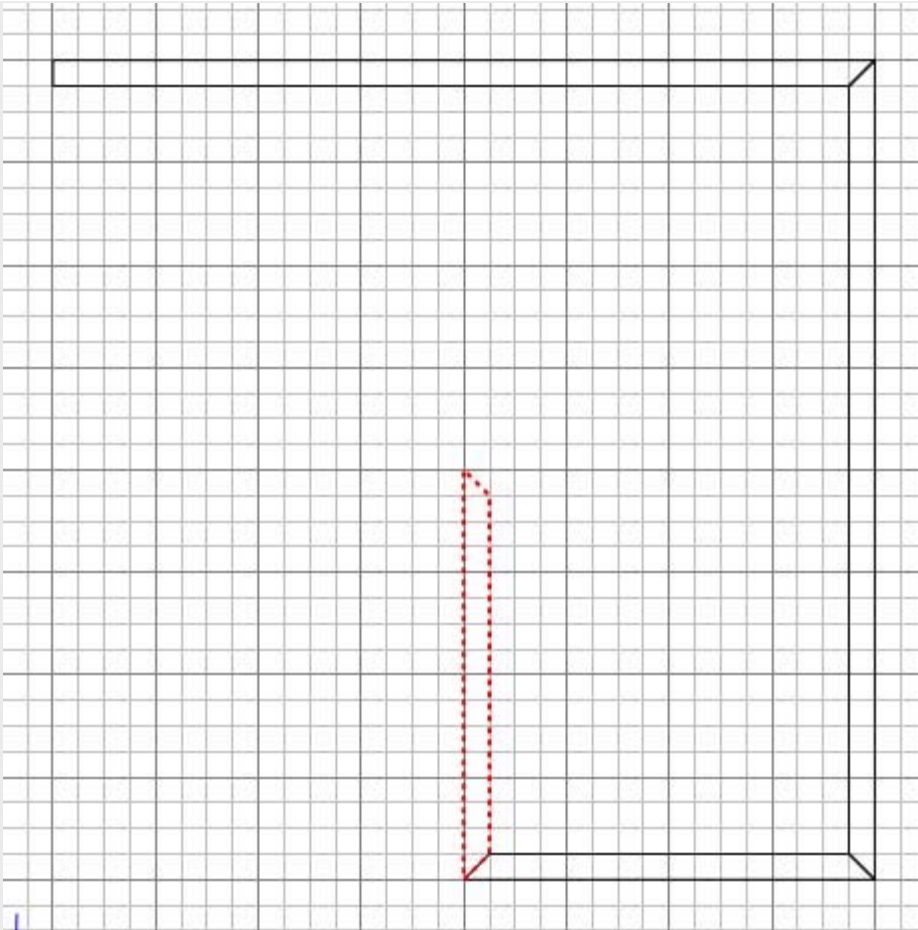




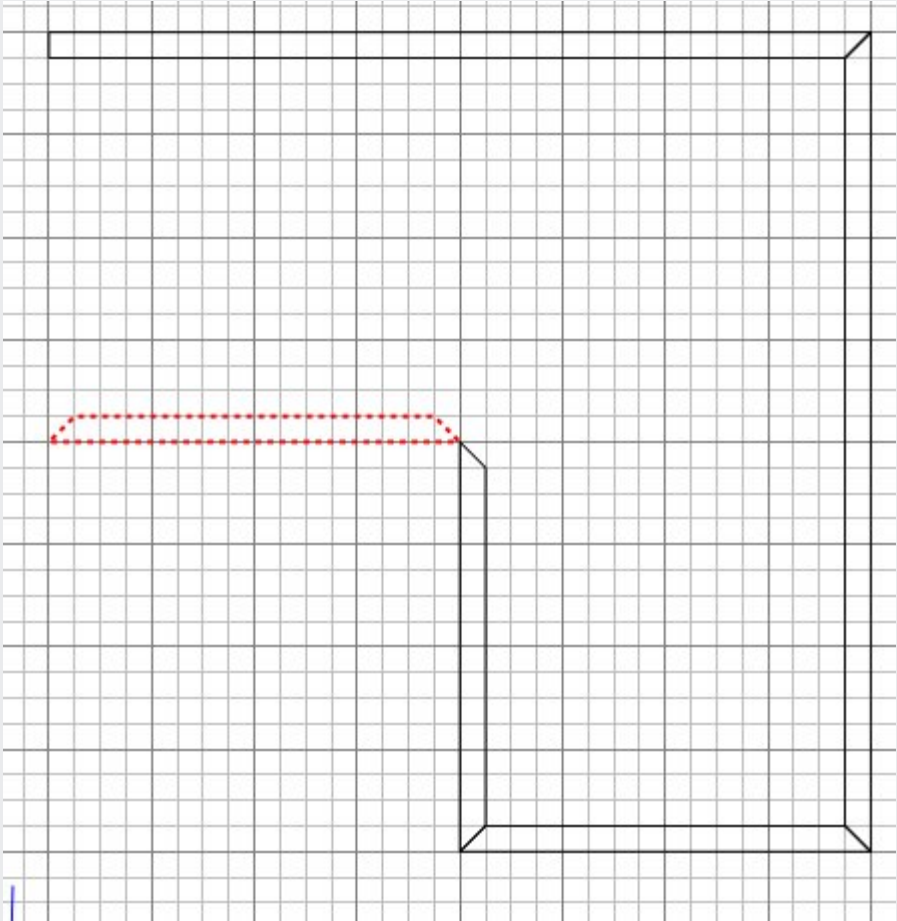
Reduce the length of the selected brush.



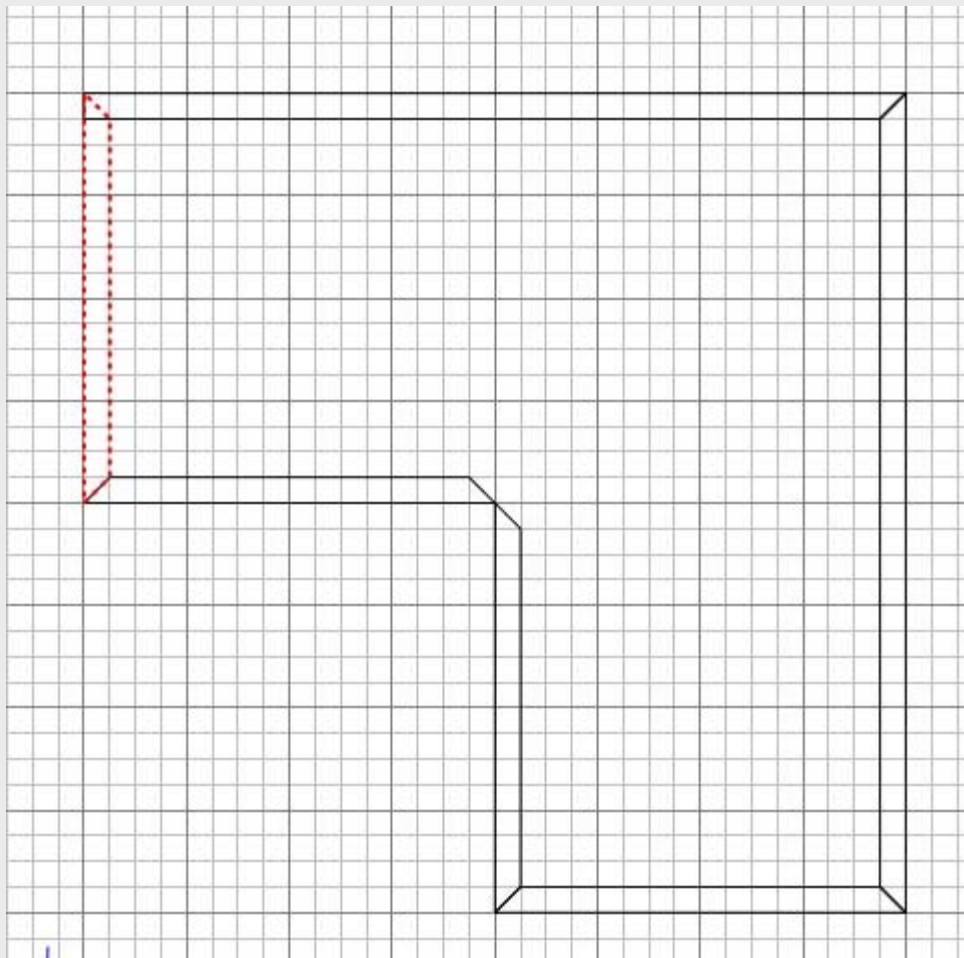
Duplicate it, rotate 90 degrees and move into place.



Duplicate it, rotate 90 degrees **3 times** and move into place.

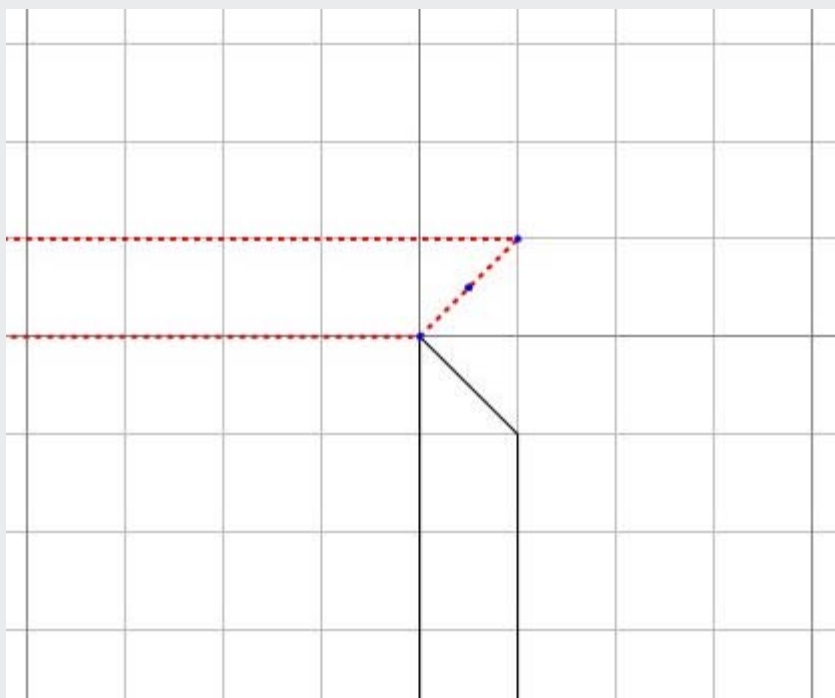


Duplicate it, rotate 90 degrees and move into place.



Press ESC. Select the top brush, press E and drag the bottom blue dot to the right to make the join 45 degrees.

Press ESC. Select the horizontal brush where the join isn't right, press E and drag the top blue dot to the right 2 notches.



Press ESC. Finally select the lower brush and use the Edge blue dot to drag its inner face up to meet the other at 45 degrees.



You've made an L-shaped outline with an efficient use of faces. You could compile here if you like and have a quick run around to see how it looks. When you come back, we'll add a floor, ceiling, some windows you can smash, a door, and for good measure, an outdoor MG42.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 6

### Topics

#### Making a door

[Making a door frame](#)

[Making a door](#)

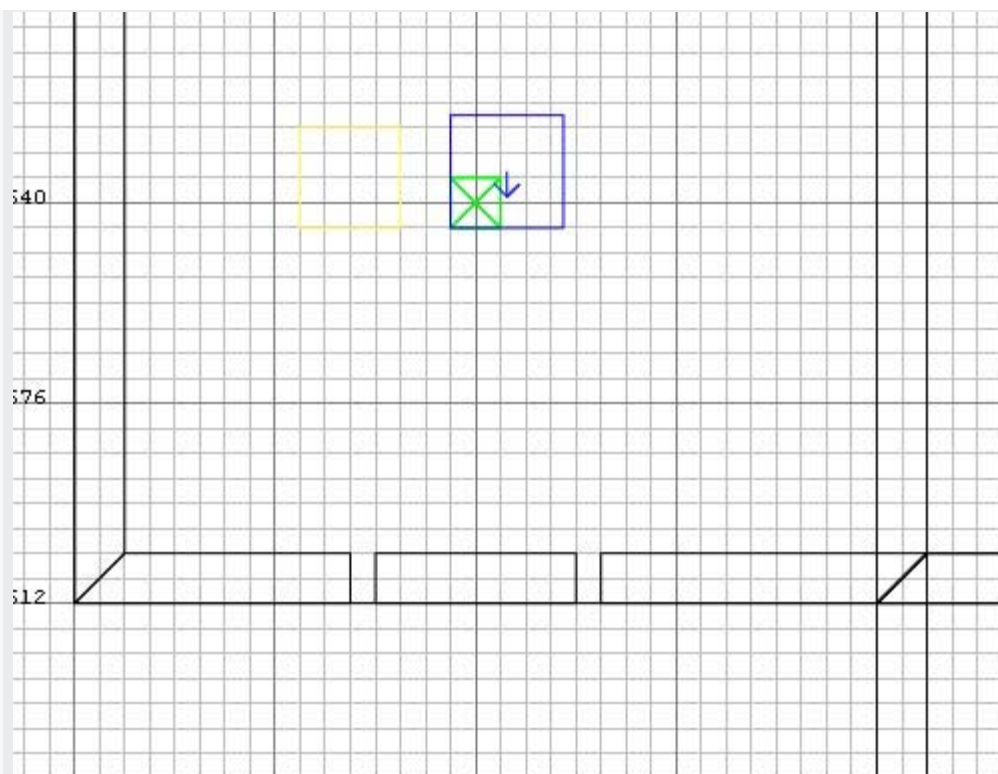
[Back to main menu](#)

#### Making a door frame

[\[Top\]](#)

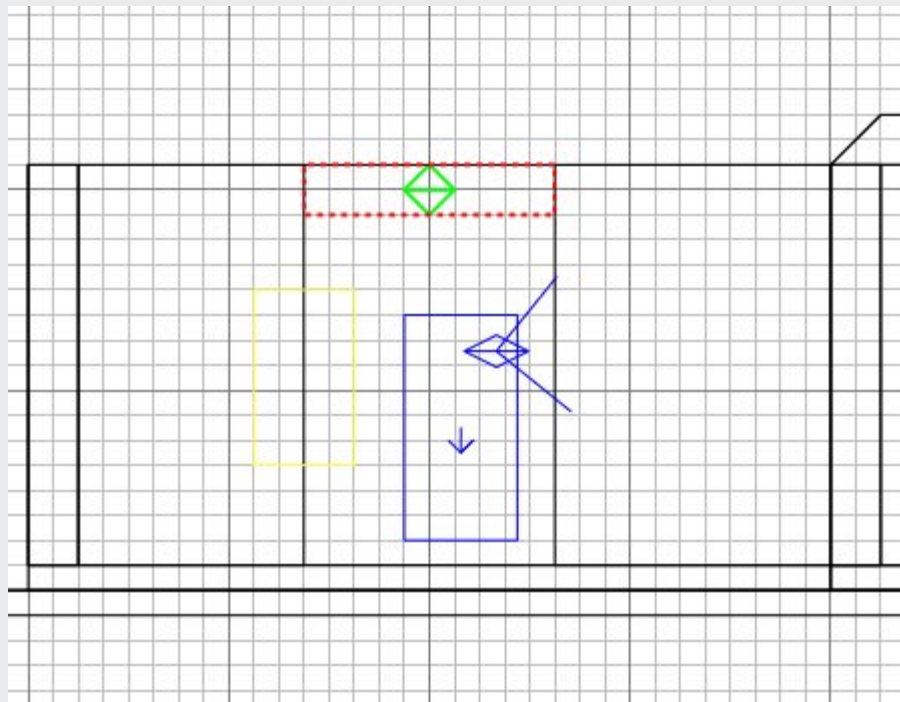
Run Radiant. Open the tutorial map. Select the environment ceiling and Hide it. Select the roof over the doorway and Hide it too.

The default 4 grid size is fine this time. Look at the doorway in the overhead view and select one of the doorway walls. Drag the wall 1 notch sideways to make the door opening a little larger. Do the same with the opposite wall. Make sure you deselect after each wall adjustment.



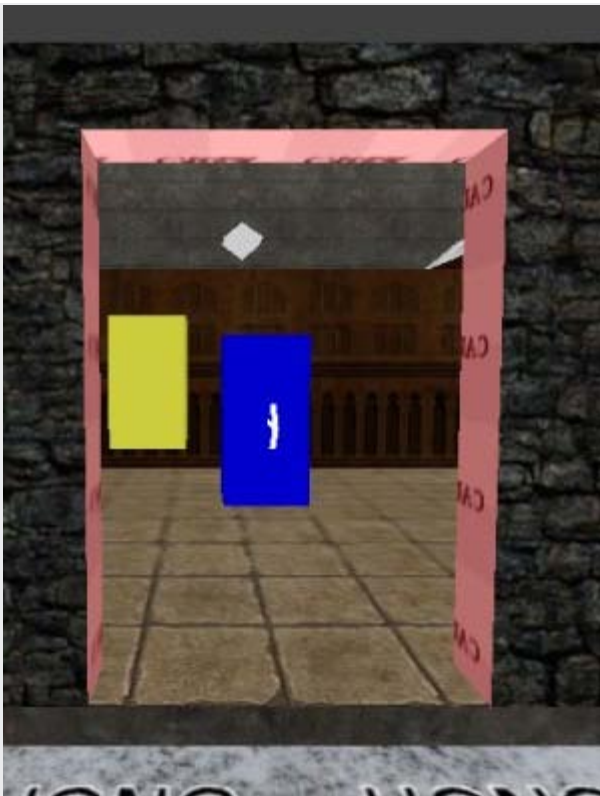
Then select the wall above the opening and stretch it both ways to meet the receded side walls.

Press ctrl+tab to see the side view, so you can shrink the wall over the door upward a little. We have now made a space all the way round the doorway to accommodate a door frame.



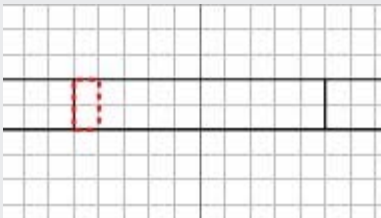
Press ESC. Select in the 3D view all the wood effect faces in the current door frame, and click the Caulk texture in the textures window.



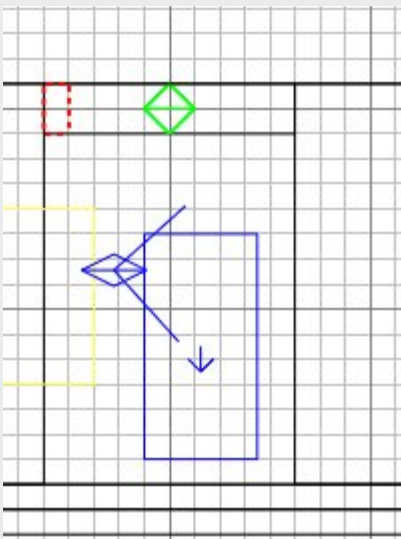


Press ESC. Now we will create a door frame, by creating brushes in the usual way.

Ctrl+tab until you get the overhead view. Then create a brush in the doorway as shown.

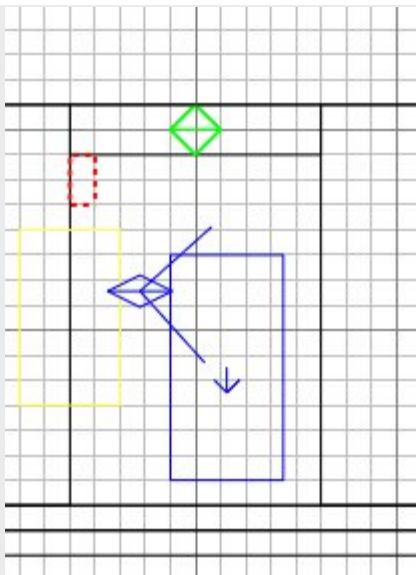


Ctrl+tab.

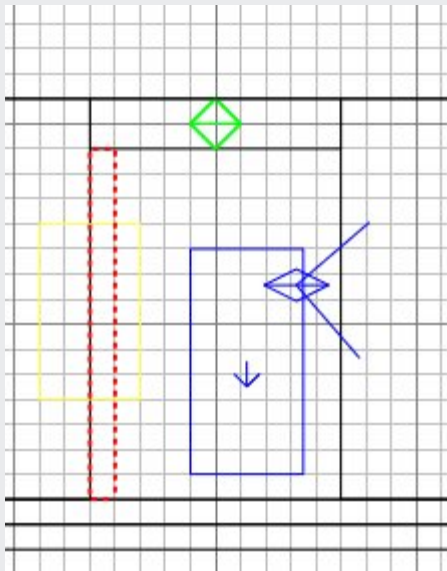


Move the brush down a couple of notches.

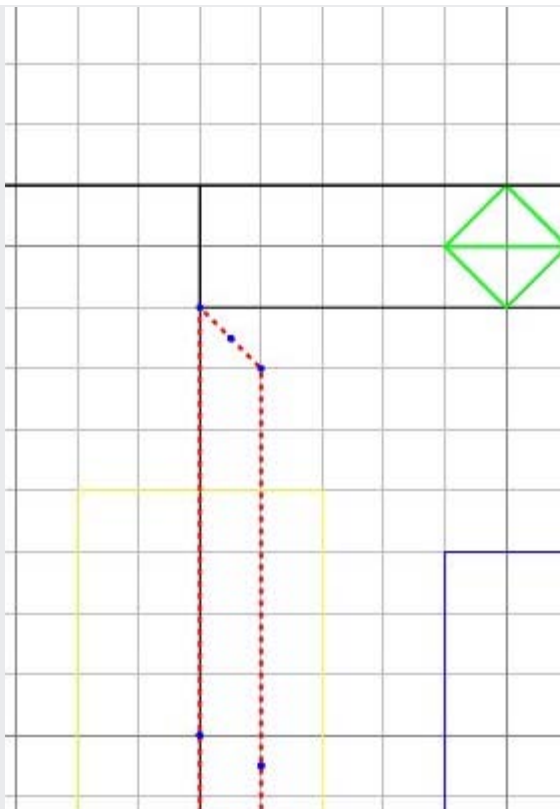




Stretch the brush down to the floor.

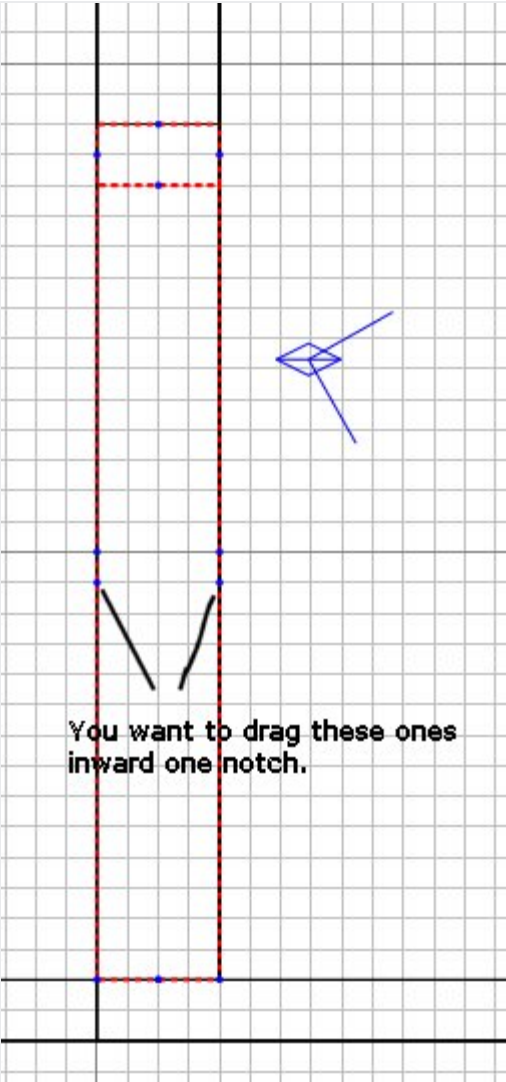


Zoom in for close up work, and angle the top part of the brush using the Edge tool.



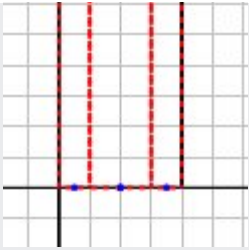
Press **3**.

Ctrl+tab again, and drag in the **lower** of the side blue dots one notch, as indicated, for both sides:

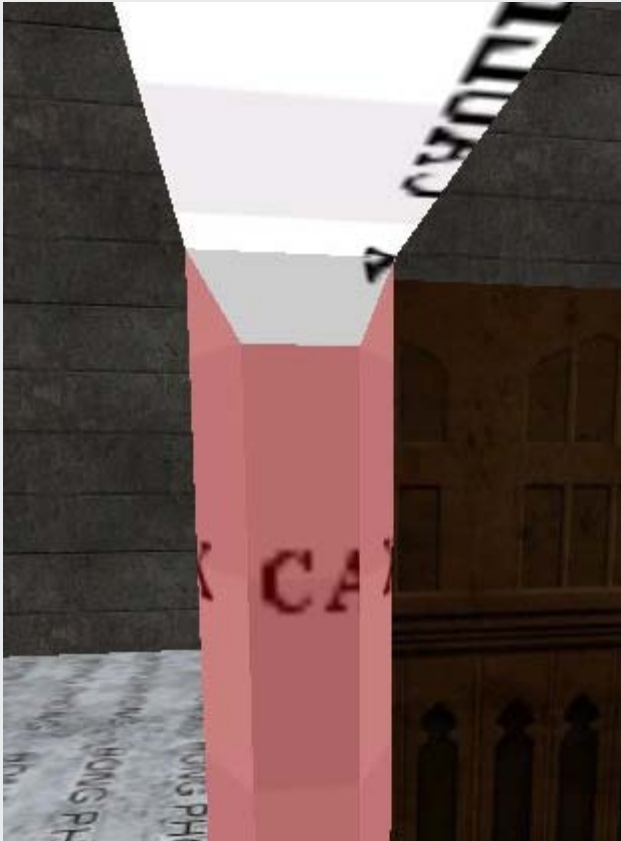


After you have dragged the two blue dots, one after the other, you get a nicely bevelled door frame side piece:





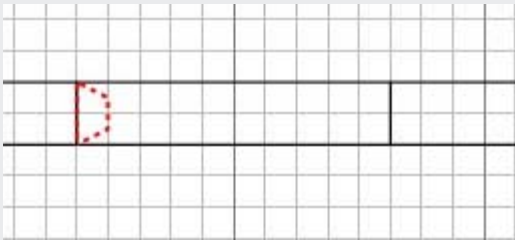
You'll see this clearly in the 3D view. Press ESC, then select the 3 main faces of the new brush, not the tiny upper one.



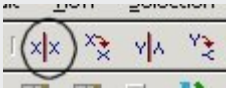
Click on the wood\_c01 texture in the textures window and press ESC.



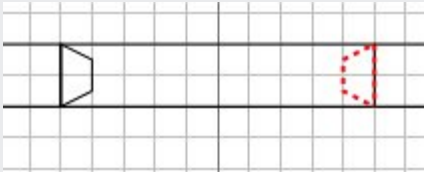
Get the overhead view via ctrl+tab if needed. Press **4**. Select the new brush.



Duplicate it, and click the "x-axis flip" button



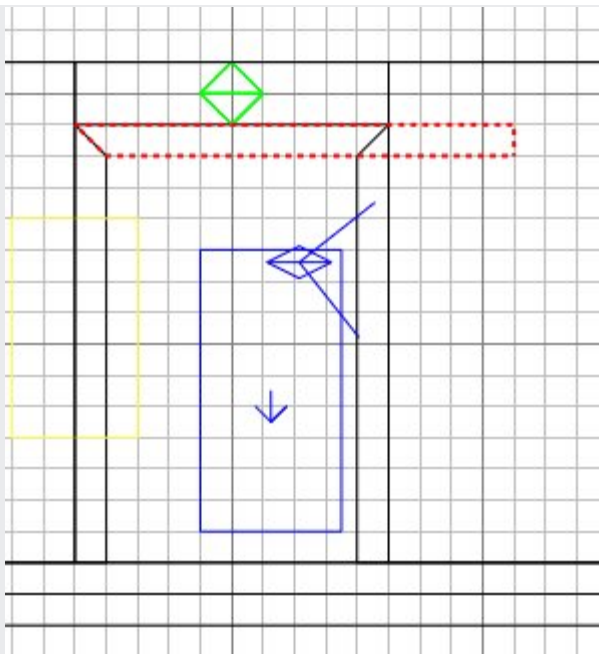
then move the brush to the opposite wall edge.



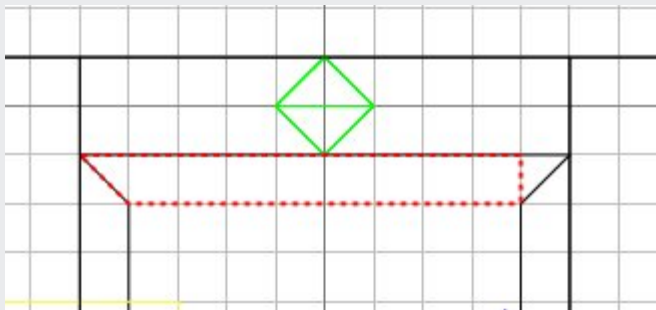
Ctrl+tab. Duplicate the brush. Click the "y-axis rotate" button 3 times and move the brush into place as shown.



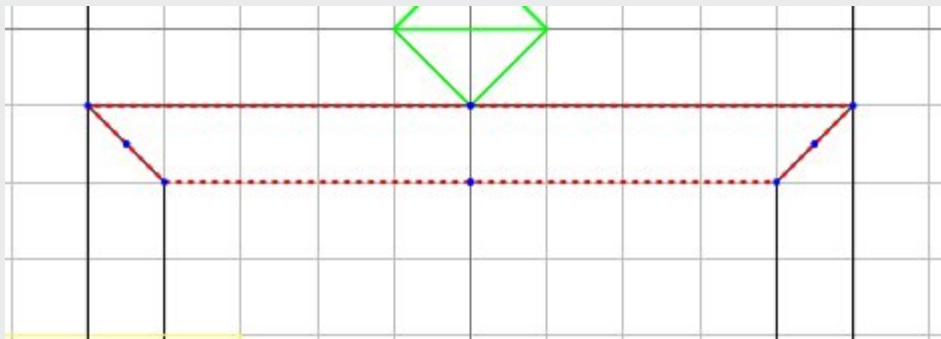
and move



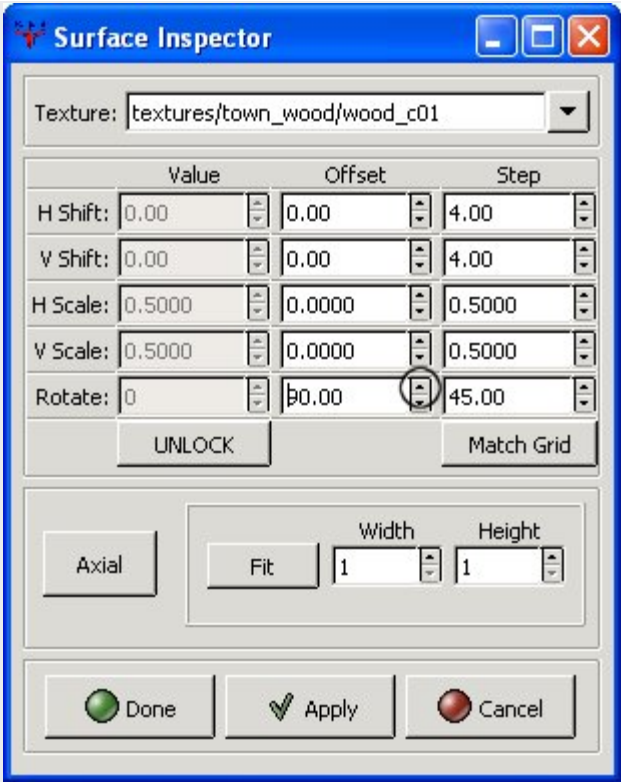
Shrink it to size.



Then use the Edge tool to get its edge to line up at 45 degrees.



Press ESC, and select all 3 visible faces of this brush. Press **S**. Click the up arrow to rotate the textures through 90 degrees. Click Done. Press ESC.

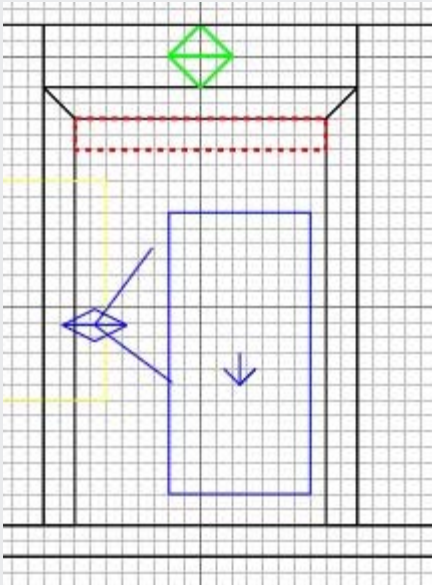


You've made the door frame.

### Making a door

[\[Top\]](#)

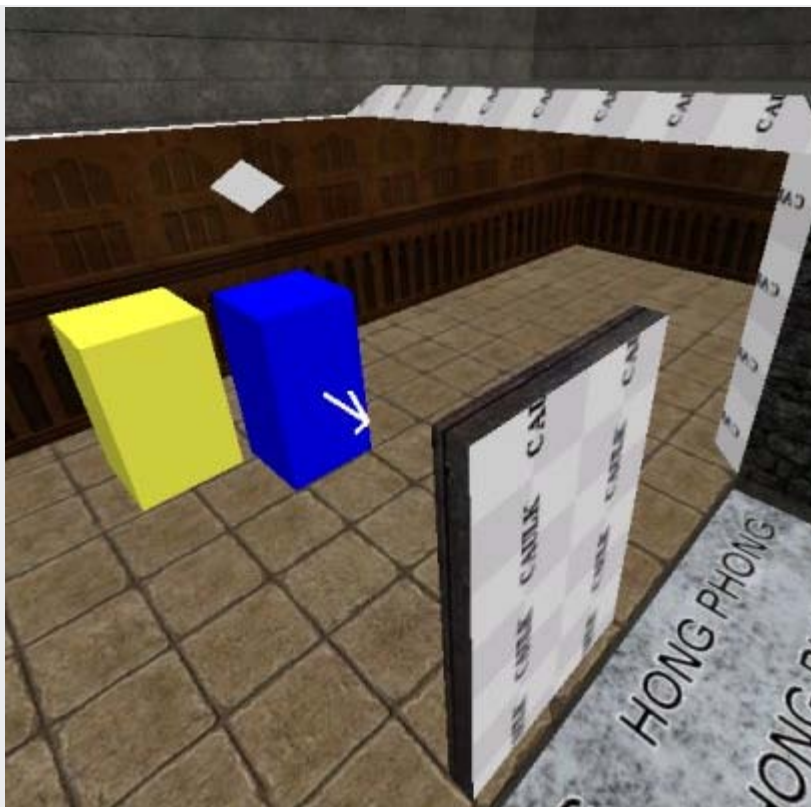
Get the overhead view. Press **3**. Create a brush as shown, and caulk it. Ctrl+tab, and drag the brush into the door opening.



Resize it to fill the doorway. Press ESC. Hide the door frames and walls that surround the door frames, so we can see our door properly.

Select the 3 door edge faces, and apply the wood\_c01 texture. Press ESC. Reselect the top face, press S, and rotate the texture through 90 degrees. Press ESC.





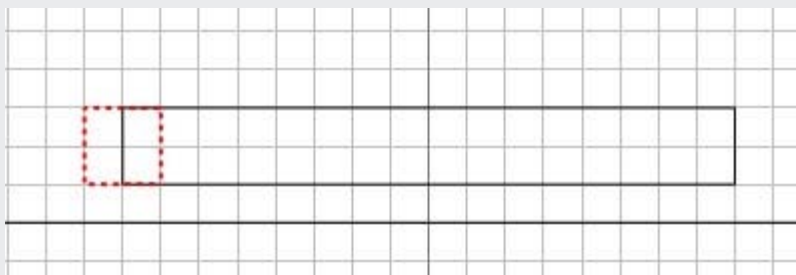
Select both of the main door faces, and click Textures/doors and click the door\_c01b texture. The door will look a muddle; don't worry. There are lots of door textures, but most of them are scattered about, rather than in the doors folder :(

Press **S**. Click "Fit". Click Done. Press ESC.

We have our door (non-functioning at the mo). Note that the door texture is mirrored on the reverse face, which is handy as it will keep the handle/hinges in the right place. Textures are always reversed on opposite edges.

Now to make the door work.

Get the overhead view, and draw a brush as shown (yes it overlaps). It will define the hinged side.



Get a side view. Adjust the brush to be the same height as the door.

Click Textures/common and click the **Origin** texture (orange check pattern).

Now select the door, so that it and the origin brush are both selected.

In the 2D window, right click and select "func" then "func\_door\_rotating".

Press N. Enter "type" as a key and "4" as a value and press return. Close the entities window.

We have an operational door, but there is one more thing we need to do, otherwise the compile will fail.

Press ESC and shift+H to reveal all the hidden brushes.

Select the 3 door frame brushes, then in the 2D window right click and select "Make Detail". This is because entities, such as an origin brush, cannot fall within a structural brush (such as all the walls made so far) but it's ok to fall within a detail brush (to be explained later).

You're all done. Save the work, compile and test. Don't worry, it isn't always this long-winded. Once you've made something like a door, you make it into a prefab (explained later) and then you can just plonk copies of it into place as and when you want.



To select the door entity, use shift+alt+click on any component brush, and the whole assembly gets selected. To select a component brush of an entity made up of several brushes, just use the regular shift+click.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 7

### Topics

#### Making a destructible window

[Making an opening for the window](#)

[Making a window](#)

[Back to main menu](#)

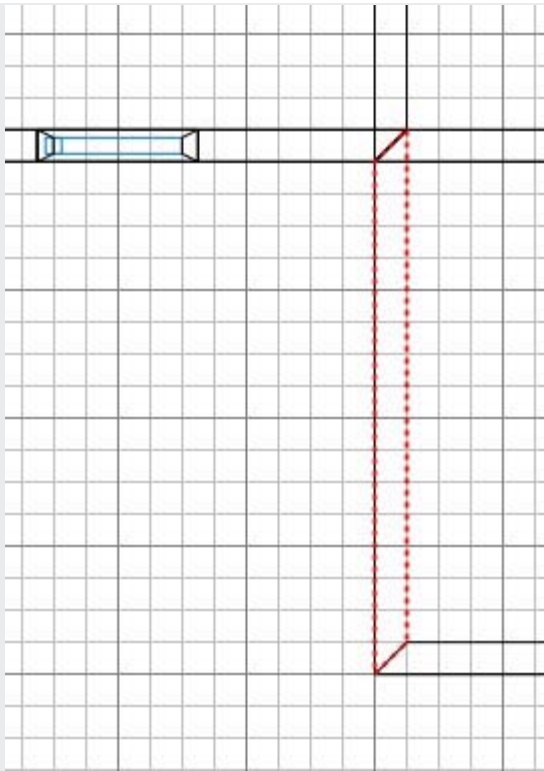
#### Making an opening for the window

[\[Top\]](#)

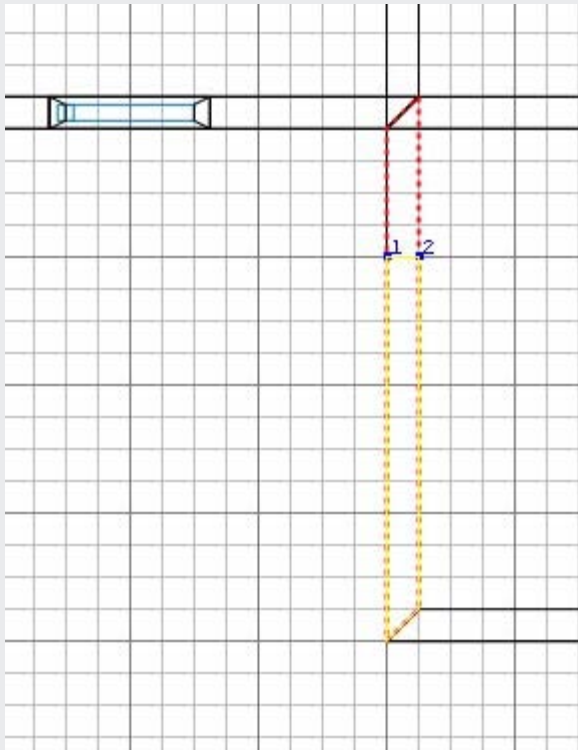
From this point on I am assuming you know how to create, resize and edit brushes with the clipper tool and edge tool (in the 2D), and that you know how to select and texture multiple faces (in the 3D). Also that by default you will apply caulk to any new brush unless directed otherwise.

Run Radiant. Open the tutorial map. Select the environment ceiling and Hide it. Select the roof over where we will put the window (see the picture below) and Hide it too.

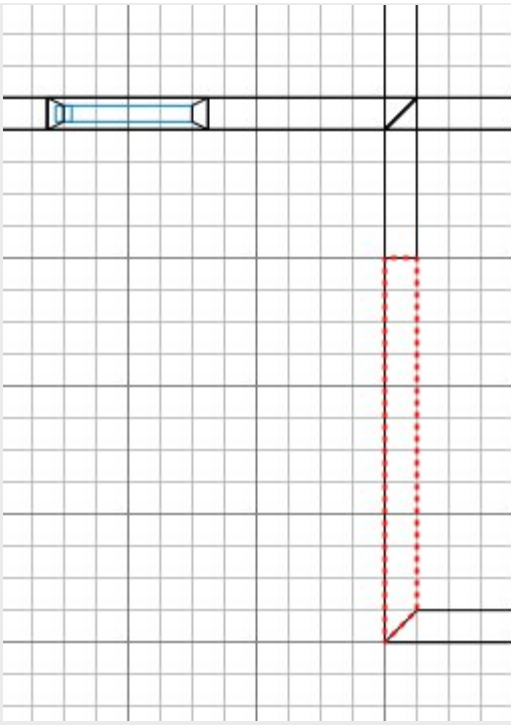
Press **5** for the grid scale. In the 2D, select the wall that we will put a window in.



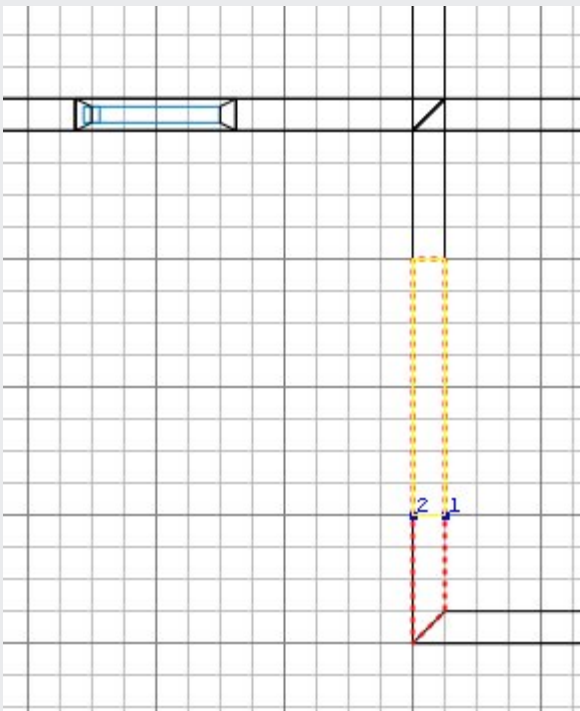
Press **X** (for the clipper) and click at spots 1 and 2:



Press shift+return (to cut the brush) then shift+click on the smaller chunk to deselect it.

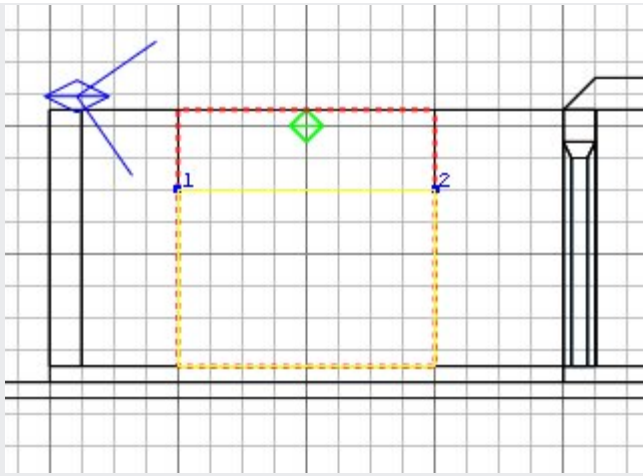


The clipper tool remains active, so click at 1 and 2...



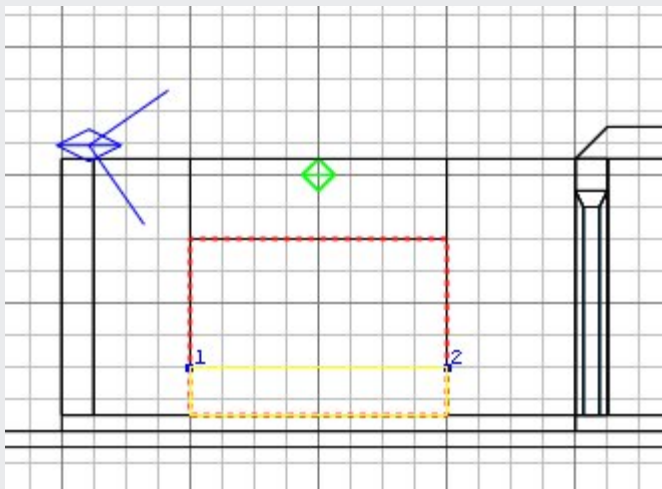
...and press shift+return again. Shift+click on the smaller chunk to deselect it.

Ctrl+tab (twice is clearer) to get a side view. Click on 1 and 2...



...and press shift+return again. Shift+click on the smaller chunk to deselect it.

Then click at 1 and 2 as shown below to complete the cutting out.



If the smaller chunk is not yellow, press ctrl+return to make it yellow.

Press return to eliminate the chunk that forms the window space.

Press ESC twice to turn everything off. (The green lines shown below are indicators I added, explained next.)





So we have the window opening.

Really we should now do one of two things for best performance: either put in a window frame, or cut up the side walls to prevent textures getting drawn behind an obscuring brush (as indicated by the green lines).

To move along more quickly we'll do the quick and dirty - which is especially ok if the current map area that the players are in has plenty of FPS slack.

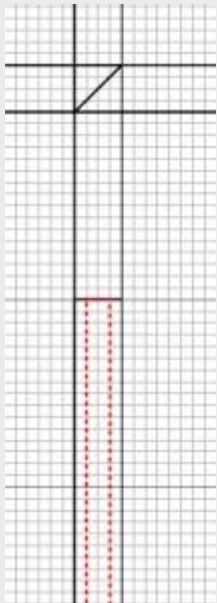
Select all the caulked window ledge faces and give them a wooden or other texture, say like plaster. I'm using town\_c61a which has a texture that doesn't need aligning. If you choose a wooden texture, you'll probably need to rotate the alignment for 2 of them, like you did in the door frame.

Press ESC.

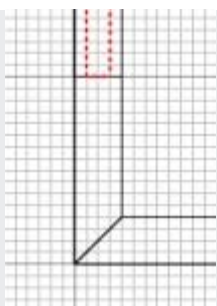
## Making a window

[\[Top\]](#)

Ctrl+tab to get the overhead view, then press **3** for the small grid scale. Draw a brush as shown:

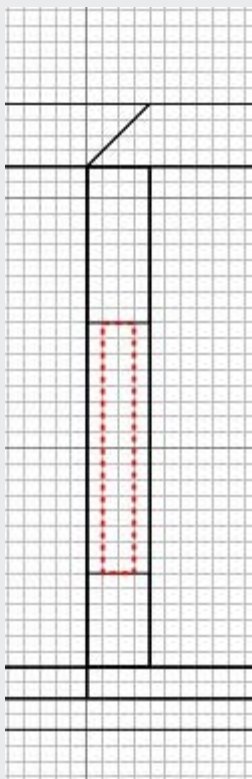






Reminder: caulk it.

Ctrl+tab, and move the window brush into place on the ledge, and resize it to fill the whole opening.



Press ESC. Select **one** of the two visible window faces. Apply Textures/sfx tramglass2.

Press ESC. Then select the window brush, and in the 2D, right-click and select func\func\_explosive.

Press N - close the entity window and press N again!

Tick the **Useshader** box. This means when the window is destroyed, the game engine will make the shards out of the same texture as the destroyed item. If you don't, the game engine will use a default texture.

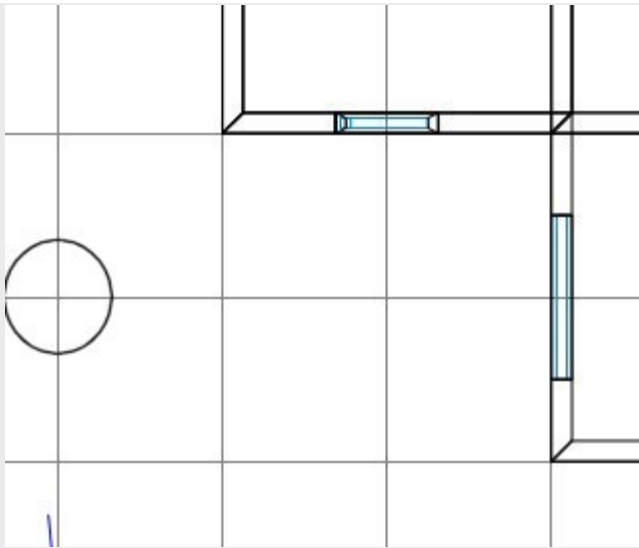
Enter "health" as a key and say "10" as the value, then press return. The bigger the health value, the tougher the glass. You could even use a value of 1 to guarantee the slightest damage will destroy the glass.

Enter "type" as a key and "glass" as the value, then press return.

Close the entities window and press ESC.

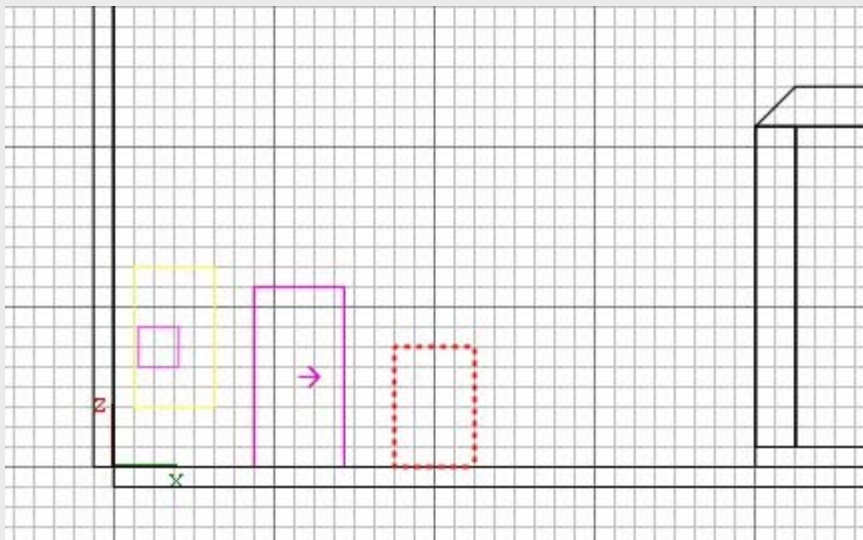
Press ctrl+tab twice to get the overhead view. Press **8** to get a big grid scale.

Just for fun, right click on the grid intersection where circled in the picture.



Select misc\misc\_MG42.

Ctrl+tab to see the side view. Press **4** and drag the red box (which represents a fixed MG42) up until it stands on the floor.



Press ESC. Save the work, compile it and go break that window soldier!



When testing you often want to restart the map quickly, say so you can rebreak your glass. Bring up the console and type:  
\map\_restart

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 8

### Topics

#### Creating the initial script

[Creating the basic script](#)

[Back to main menu](#)

#### Creating the basic script

[\[Top\]](#)

Every map comes with a .script file, which is a text file found in the maps folder.

The purpose of the script is to tell the game engine some basic operational details about the map, like what the respawn times are, and to power the interesting bits of the map, like making a tank move, a balloon fire rockets, or getting constructible things to go all white/wavy and then turn into the constructed thing.

Here is a .script file for the tutorial - [click here](#) - unzip it and put the .script file into your maps (yes, maps) folder. You'd think it would go in the scripts folder wouldn't you. One of ET's quirks.

Associate the script suffix with Wordpad. Rename it to <yourmap>.script if you are not using the name "tutorial".

Below is a brief explanation of its contents. Don't worry, it's straightforward.

Scripts contain a number of separate sections, with each section handling a particular aspect of something that features in the map. A section is given a name, and then its contents are marked out inside a pair of curly brackets, like these { }. The naming of the sections is up to you, but the names chosen sometimes have to match the names used in the map - we'll come to that later on.

The first and only section that *must* appear in the script file, is called "game\_manager". It contains information about the fundamental elements of your map, as you will see below.

The **game\_manager** section must contain a procedure named **spawn**, which is the procedure executed in a section when the map starts. Each section can have a spawn procedure if you want something to happen for it when the game starts, but it isn't mandatory.

For example, the spawn procedure for a tank might make the tank start the game disabled.

Here are the contents of the script file for the tutorial map, followed by explanations. Most of it is obvious.

```
game_manager
```

```
{
```

```

spawn
{
    wm_axis_respawntime 10
    wm_allied_respawntime 10
    wm_set_round_timelimit 30
    // Stopwatch mode defending team (0=Axis, 1=Allies)
    wm_set_defending_team 0
    // Winner on expiration of round timer (0=Axis, 1=Allies, -1=Nobody)
    wm_setwinner 0
    wait 500
    setautospawn "Axis Spawn" 0
    setautospawn "Allied Spawn" 1
}
}

```

wm_axis_respawntime 10	Sets the Axis respawntime to 10 seconds. This can be changed in the script later on if wanted, say on reaching a certain objective.
wm_allied_respawntime 10	As above, for Allies.
wm_set_round_timelimit 30	Sets how many minutes the map is to last.
wm_set_defending_team 0	For Stopwatch games, sets the initial defending team. <b>Wherever a team is referenced in the script, 0 means Axis and 1 means Allied</b>
wm_setwinner 0	If the map timer expires, the winning team is declared to be team 0, ie the Axis.
wait 500	Wait 500 milliseconds, to allow the other game components to start up, so that they can be safely referenced
setautospawn "Axis Spawn" 0	Tells the game where the Axis players should spawn by default. In the tutorial map, the yellow team_wolf_objective box for the Axis team has a description (visible if you select the box and press N) called "Axis Spawn". This is what is being referenced in the script.
setautospawn "Allied Spawn" 1	Likewise for the Allies.

T

he handy thing about having at least this much script in place is that it can set the respawntime to 10 secs, making your testing easier.

As you will have guessed, lines that have `//` in them are treated as comments from that point to the end of the line.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 9

### Topics

#### Creating the mission text to be shown as the map loads

[Making the .arena file](#)

[Back to main menu](#)

### Making the .arena file

[\[Top\]](#)

Every map comes with a .arena file, which is a text file found in the scripts folder.

The main purpose of the contents of the file is to supply the text to display as the map loads.

Here is a .arena file for the tutorial - [click here](#) - unzip it and put the .arena file into your scripts folder.

Associate the arena suffix with Wordpad. Rename it to <yourmap>.arena if you are not using the name "tutorial".

Below is a brief explanation of its contents.

```
{
    map "tutorial"
    longname "^2The ^7Tutorial"
    type "wolfmp wolfsw"
    timelimit 30
    axisRespawnTime 10
    alliedRespawnTime 10
    lmsbriefing "Fight to the death."
    briefing "Version 1.0.0**A scratch force of Allied students desperately struggle with the Axis
mapping documents.**They must overcome the defences and emerge
victorious.**^7www.tibetclan.com"
    axiswintext "Not used"
    alliedwintext "Not used"
```

```
mapposition_x 570
mapposition_y 660
}
```

map : the map name, "tutorial" in this instance.

longname : the text displayed as the map name. In this example I've stuck the word "The" at the start and applied some colours in the usual way.

type : wolf multiplayer and stopwatch values.

timelimit, axisRespawnTime and alliedRespawnTime are redundant; enter values that roughly match the real values in the script, but it doesn't matter.

lmsbriefing : For Last Man Standing. No-one plays it. Forget it.

briefing : The text that appears on the right panel while the map loads, unless overridden by campaign or server text. "\*\*\*" means new paragraph. "\*" means new line. Colours can be applied in the usual way.

axiswintext "Not used"

alliedwintext "Not used"

mapposition\_x and mapposition\_y : tell ET where to put the drawing pin on the loading map. When making your map, keep adjusting these values by 10s and 20s until the pin gets to where you want. So each time you are about to test, tweak these numbers until you get it right.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 10

### Topics

#### Adding ambient sounds

[Adding ambient sounds](#)

[Back to main menu](#)

#### Adding ambient sounds

[\[Top\]](#)

These are the bird tweets, disturbed gravel, war noises, dripping water, howling wind etc that all help to make a convincing environment.

They make a big difference so you should always consider what sort of ambient noise might be applicable at various places in your map.

You're going to need PakScape sooner or later , so you might as well get it now.

Download pakscape : [here](#)

It doesn't need installing, just unzipping.

Plonk it in your ET folder or anywhere you keep utilities.

Use Windows Explorer to navigate to the etmain folder. Right click pak0.pk3 and associate PakScape with the pk3 file type, and open pak0.pk3.

You'll see a windows-style view of the insides of the pk3.

Open the sound folder, and the world folder within it.

Double click on any of the WAVs to marvel in the audio possibilities open to you. It is possible to make your own WAVs, but using some of these standard ones is a good start.

I will use war.wav for this demo.

Contrary to what you'd expect, ambient sounds are not ordinarily placed in Radiant (they can be, but this other way is better). You place them while you are wandering around your map in ET!

So, run ET and /devmap <yourmap> - then stay in Spec mode when it starts. If you haven't tested recently, you will see your map description text as the map loads, and you now have a respawn time of 10 secs, which is better for testing.

Glide your viewpoint to somewhere high up around the middle of the map, and bring up the console.



Enter **\editspeakers** to enter speaker editing mode.

Then enter **\dumpspeaker** to create a speaker in front of you.

Then enter **\modifyspeaker** to edit the speaker you've highlighted (it will be pink) by looking at it.

(You can use / instead of \ if you prefer.)

Exit the console, and enter these values in the boxes:

Noise	sound/world/war.wav	the wav to play
Looped	on	keep playing this sound indefinitely
broadcast	global	everyone should hear it
range	10000	a good range to make sure everyone is in range

Click OK.

Move your 3D view around so you can see where the speaker actually sits. For a broadcast global speaker roughly the middle is fine. If you wanted, say radio static, you'd put the speaker near the radio brush and make the range smaller.

Practice moving the speaker, by looking at it so it goes pink, then going back to the console and entering \modifyspeaker (just use the up arrow to get back recent console commands) and come back out of the console. This time drag on the coloured bars sticking out of the speaker and move the speaker to where you want it. Put this one near the map centre.

Click on OK to end the modifying. Go back to the console and enter \editspeakers to end the editing session.

Join a team and run around a bit to see how the sound comes across - is it too loud/quiet, in the right place, etc?

When you add speakers to the map in this way, ET creates and modifies a file called <yourmap>.sps which it puts in the \etmain\sound\maps folder. When the time comes, we would need to include this file in the PK3 you'll build to distribute your creation.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 11

### Topics

#### Planting a tree

[Introduction to models](#)

[Making some ceiling space](#)

[Putting a tree model into place](#)

[Making a tree prefab](#)

[Back to main menu](#)

### Introduction to models

[\[Top\]](#)

What's a model? It's a predefined object that you can plonk straight into your map, without having to make or paint any brushes. Just like putting a ready-made MG42 into the map, we can put in any ready-made models we want, things like trucks, tanks, trees, vases...

For Radiant to see a model to give you the option of picking it, it has to be within your ET folder structure, rather than just in a PK3. When Radiant installs, it creates the models folder within etmain, and the mapobjects folder within models. Inside there are a load of folders each of which contain model files, ending in .MD3.

Models are great ways to include some graphical detail without having to do any work. :)

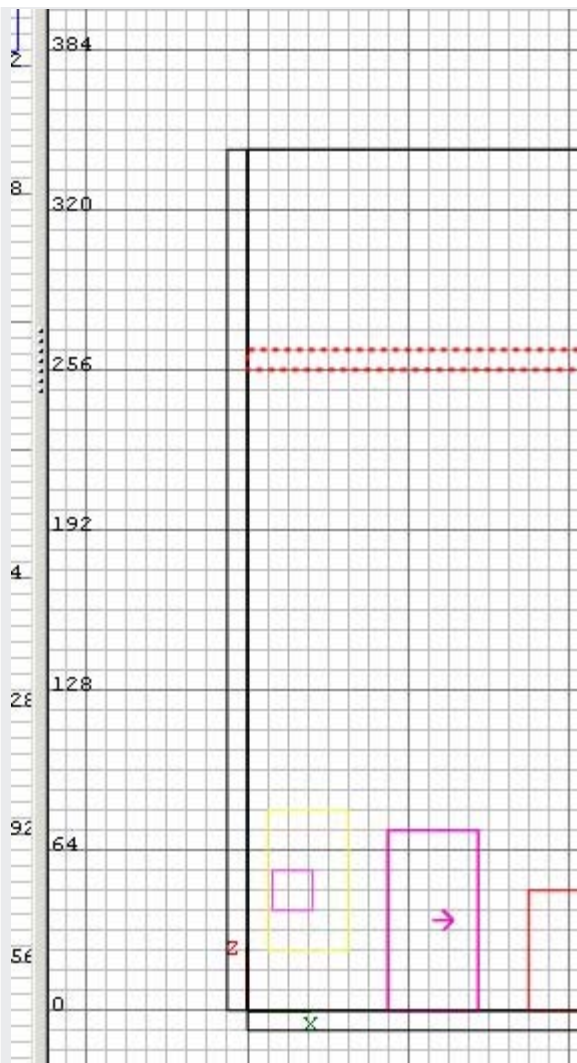
Apart from a bit. Models are drawn by the game, but do not affect player or missile movement. So if you put a tree model into place and leave it at that, the players would be able to run clean through it. You'll see how you can stop that further down in this lesson.

### Making some ceiling space

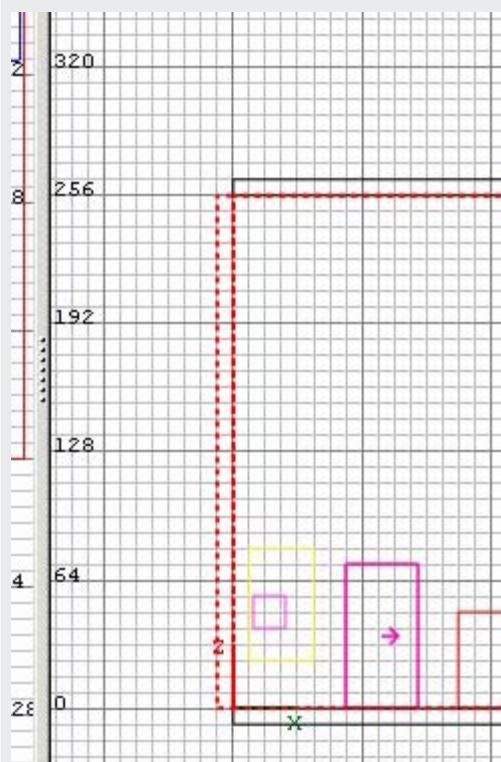
[\[Top\]](#)

Trees are tall. In our little space we need to raise the ceiling. This should also prevent the "grenades-through-the-sky" syndrome. If you find you can still manage to get a grenade through the sky, you could lift the ceiling higher again until you can't.

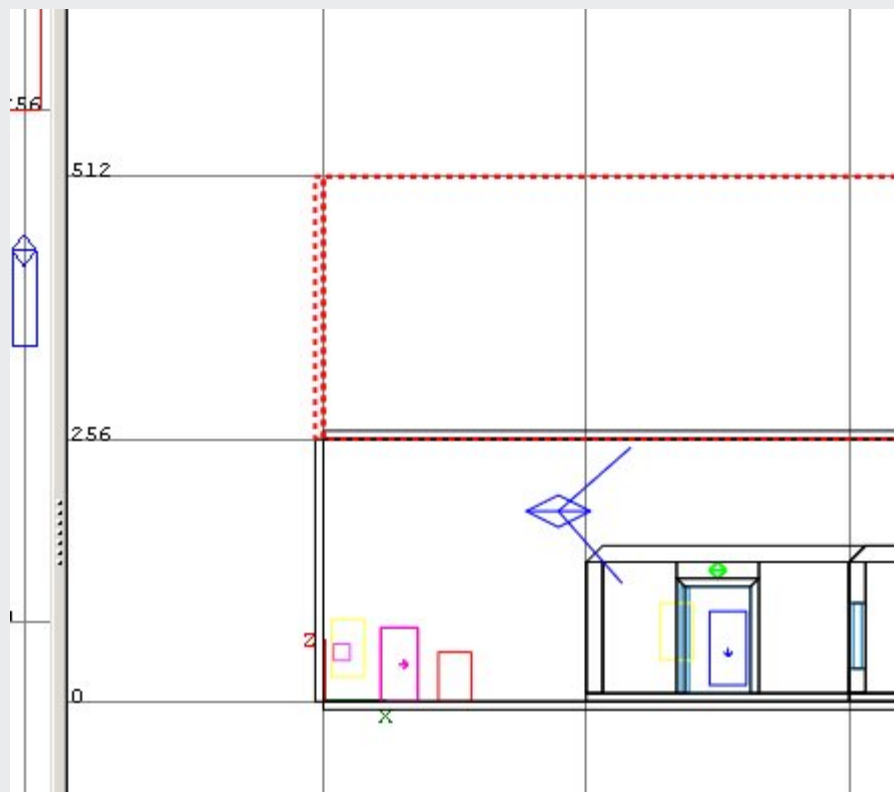
Run Radiant and open the map. Ctrl+tab to see the side view. Select the sky brush, and move it down until it sits on the 256 Z line. We're doing this to get the brush dimensions to fit nicely into a very large grid size.



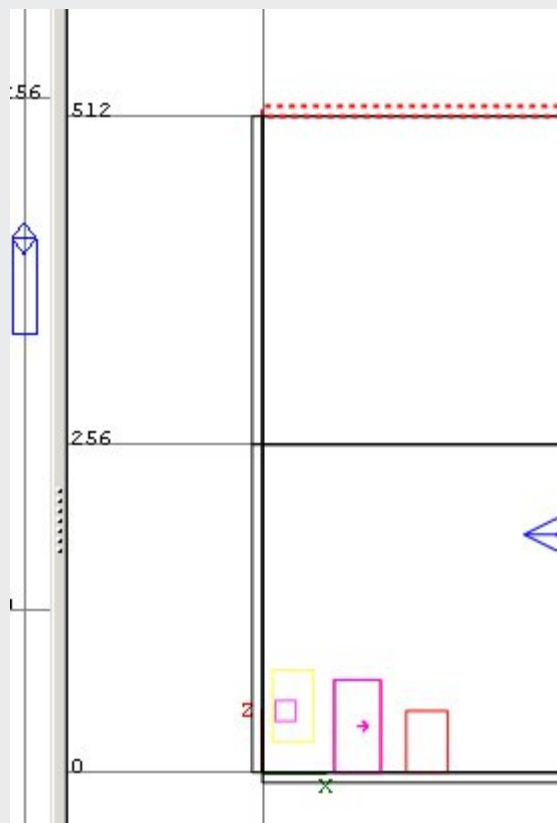
Press ESC. Select the 4 walls, and decrease their height to meet the ceiling.



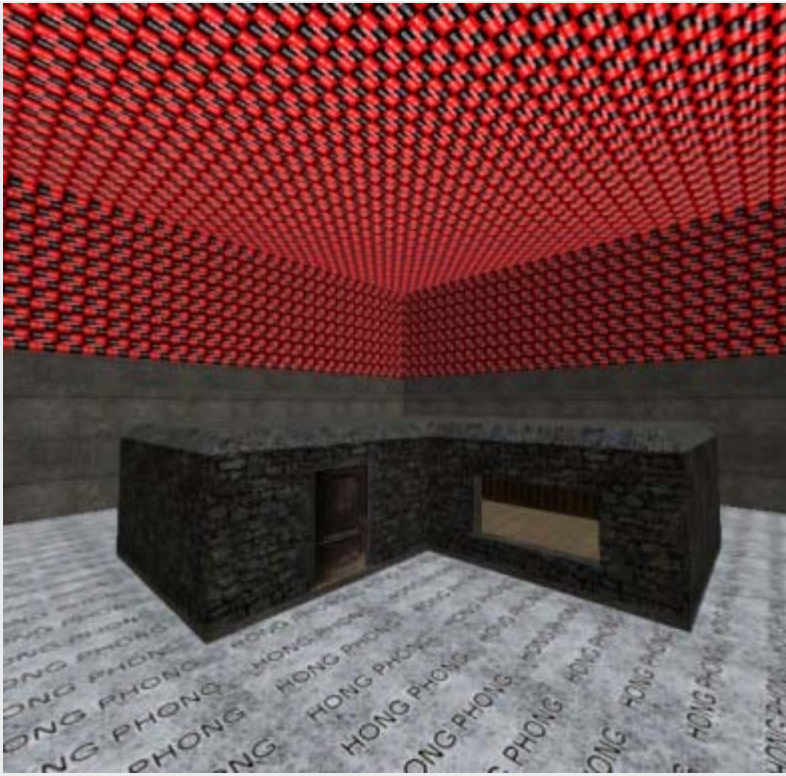
Press **9** for a nice big grid size. Duplicate the four walls. Move them onto the top of the other walls.



Press **ESC**. Select the sky again, and move it up one notch.



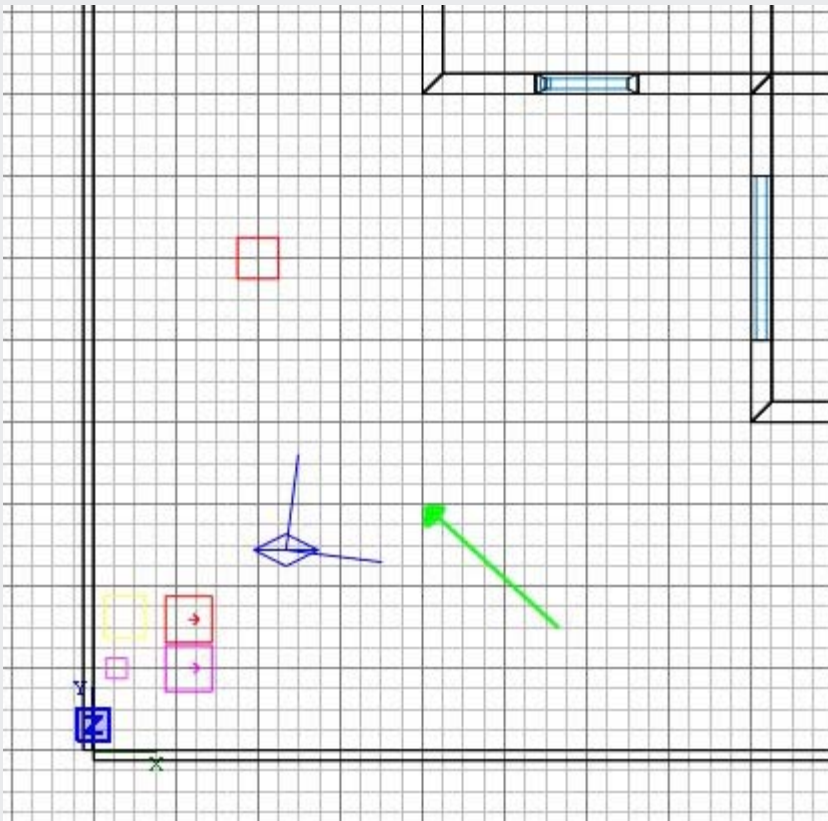
Press **ESC**. Now select all the inner wall faces of the new high walls we just built, and finally select the sky face too. You should have 5 faces selected, and the sky one must be the last. Why did we select the sky face? Because it's a handy way of getting the sky texture visible in the textures window. Click it now and it will apply sky texture to all the selected faces. Press **ESC**.



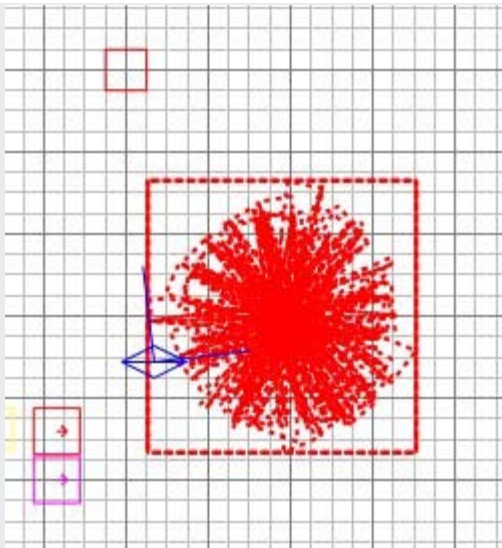
## Putting a tree model into place

[\[Top\]](#)

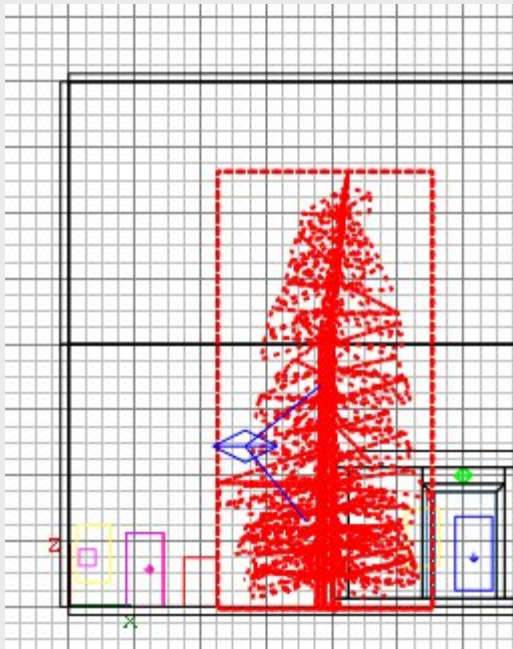
Get the overhead view. Press **5** to get a reasonable grid scale. Right click where shown here:



Click misc/misc\_model and a window will open. Double-click mapobjects then trees\_sd, then tree\_a.md3.



Get a side view, and move the tree down to the ground **by click/dragging somewhere within the actual tree shape, rather than just somewhere within the box it comes in.**

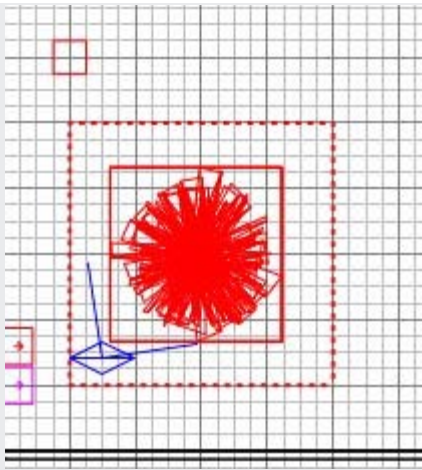


Maybe it's a little too big. Press N and enter a key of "modelscale" and a value of "0.8" and press return.

Ok the tree is about the right size, now we must add a "clip" brush, which is a brush that isn't drawn, but will act as the solid tree trunk to block movement, and also to give a wooden bullet ricochet noise if the clip brush is hit.

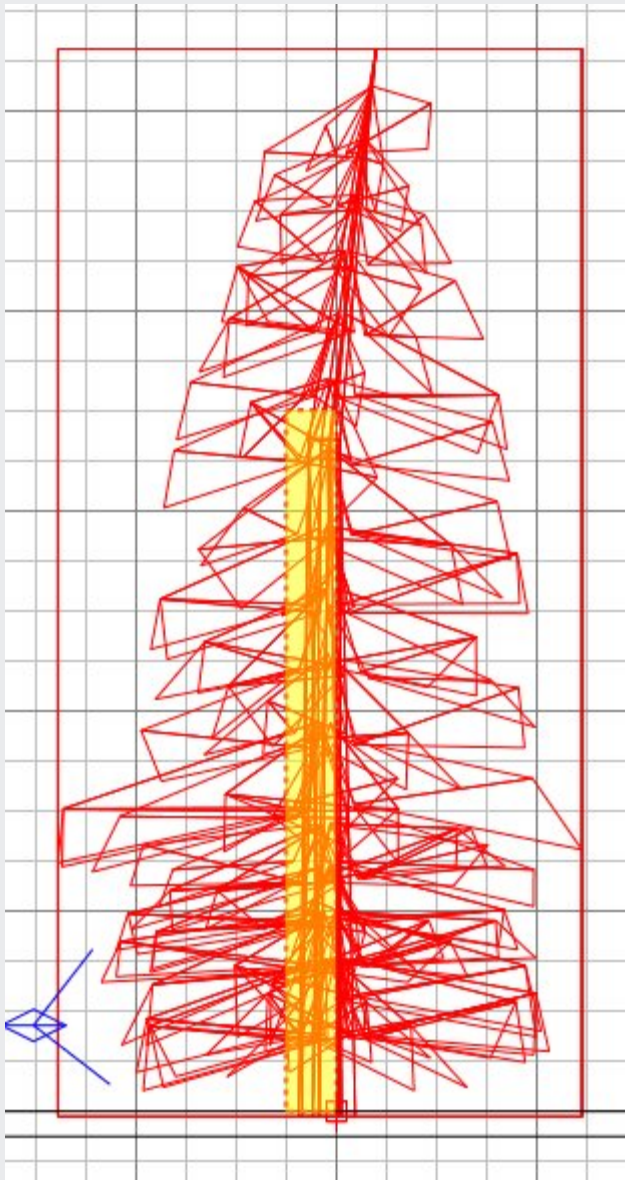
Press ESC. Get the overhead view and draw a brush as shown.





Then select Region/Set Tall Brush from the menu at the top of the screen. This restricts what Radiant draws to whatever is inside the box you drew. The box is deleted as it has fulfilled its task of indicating the area you are interested in.

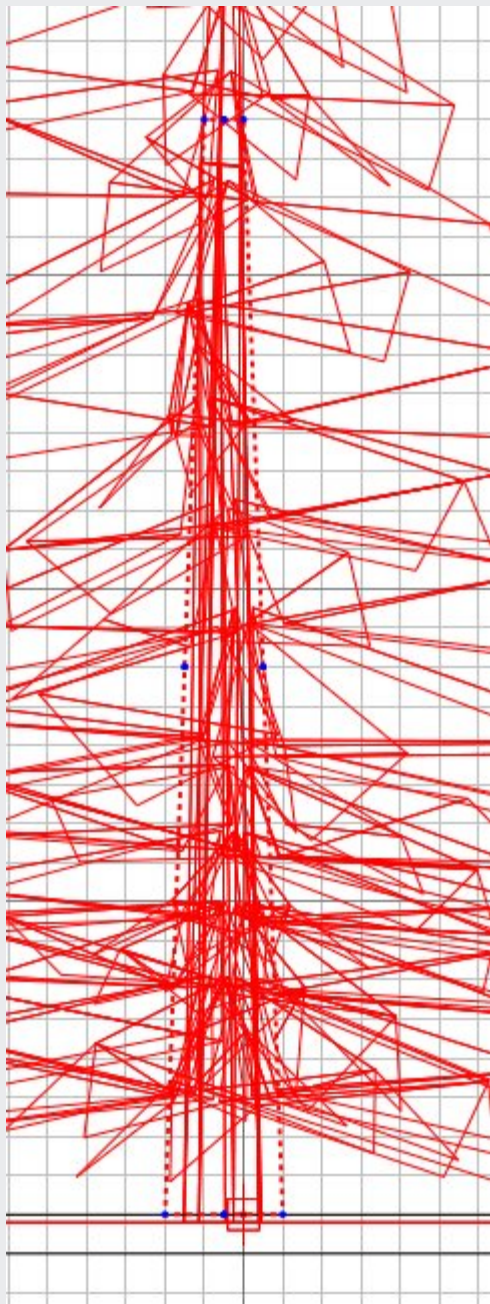
Get a side view and draw a brush as shown. I've coloured it yellow so you can see the red dashed outline against the red model.



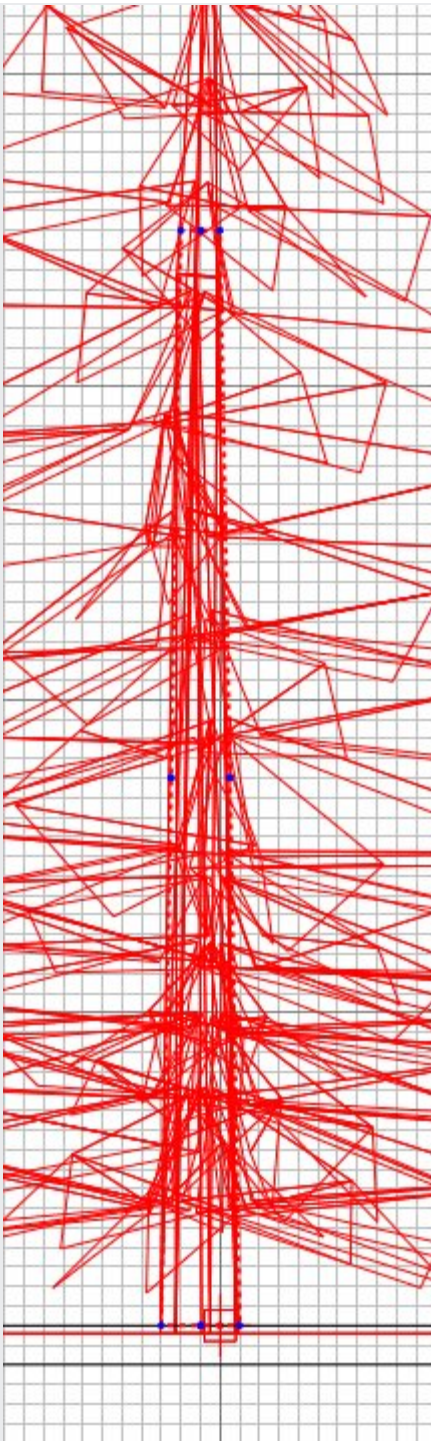
Get the next side view, and shrink the brush to about the right size by pressing **4** to get a better grid scale..



Once the brush is about right, but a bit lumpy and straight upright, use the Edge tool and grab the corner blue dots to make the brush hug the tree trunk up to about the point where you no longer need clipping, ie up high in the leaves.



Keep using ctrl+tab to check the 2 side views as you refine the clip brush. Use **3** if you want to get the fit really snug. You can see in the 3D how you are doing.



Press E when you are satisfied with it. Then click Textures/common and click on the Clip Weapon Wood (green/mauve check) texture. Finally right-click in the 2D and select Make Detail. This deselects the brush too.

Ctrl+tab to get the overhead view. Press **7** to return to a sensible grid scale. Click Region/Off to see the whole map again. Save your work.

## Making a tree prefab

[\[Top\]](#)

Happily you won't have to do that for every tree.

Make a folder called "prefabs" in your maps folder.

Select the tree model and its clip, either by clicking them both (fiddly) or by drawing a box around them and

"selecting complete tall".

Click File/**Save Selected...** then double-click the prefabs folder, give a file name of say "snowy\_tree\_1" and save it as a .map file type. Press ESC.



You can save any collection of brushes as a prefab. This is hugely helpful, as it means you can import prefabs as you need, without having to reinvent every little detail.

To include a prefab into your map, you click File/**Import...** and navigate to the prefab you want. It will appear within your map, at the co-ordinates that it was saved at, and already selected ready for you to move into place. When moving a model/clip like this, be sure to put your cursor within the clip area, or you may distort the clip by accident.

If you change the modelscale of your tree later on, be sure to change the size of the clip brush too. More on that later.

To make a little clump of trees it is best to make a few prefabs using different trees and different scales, and then import a variety so they don't all look the same.

Compile and test to make sure your tree is ok. Notice how the wall and ceiling sky textures are seamlessly merged by the game engine. Neat. And your tree model casts a plausible shadow. Very neat.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 12

### Topics

#### Making stuff you can shoot up

[Fencing](#)

[Back to main menu](#)

### Fencing

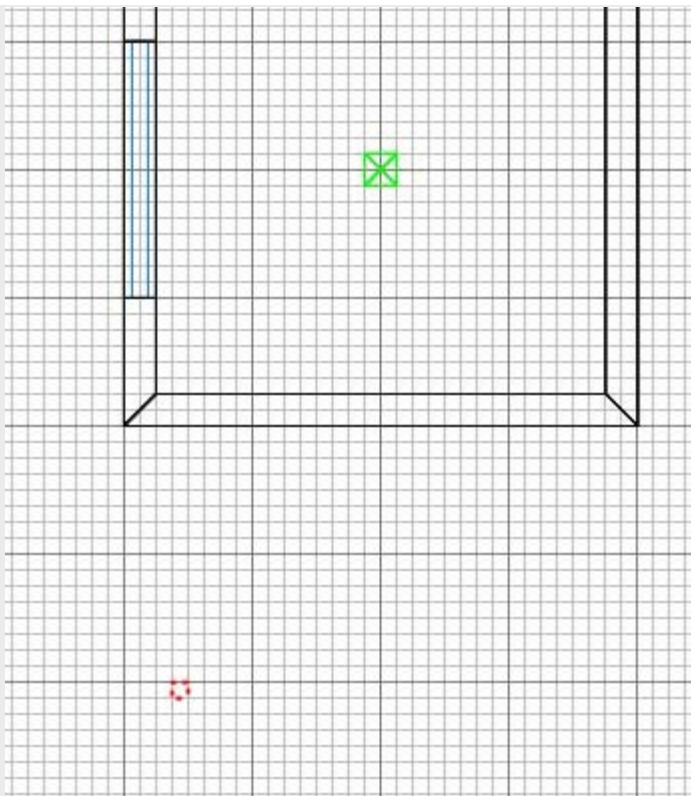
[\[Top\]](#)

Anything you can make out of brushes, you can make destructible. You can make a single brush destructible, or you can group brushes together as a single destructible entity.

At the simpler level, it is very easy to make things destructible by any and all attacks, from knife up to artillery. It is more involved to make say a wall destructible only by dynamite.

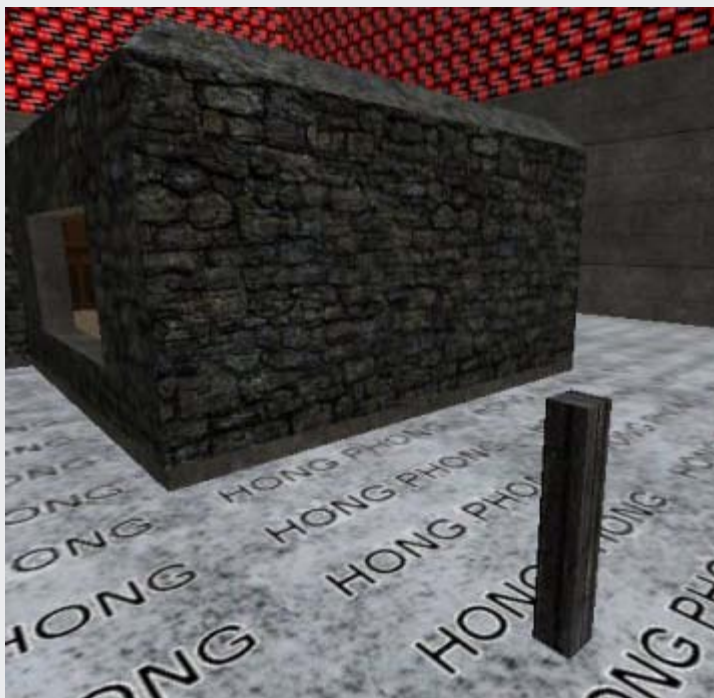
We're going to start at the simple end and make some fences you can shoot to bits with bullets.

Run Radiant, open the map, and make a fence post by drawing a box as shown.



Ctrl+tab so we can control the height of the fence post and ensure it stands on the ground. Caulk it of course too. Press ESC.

Apply some wood texture to the 5 visible faces.



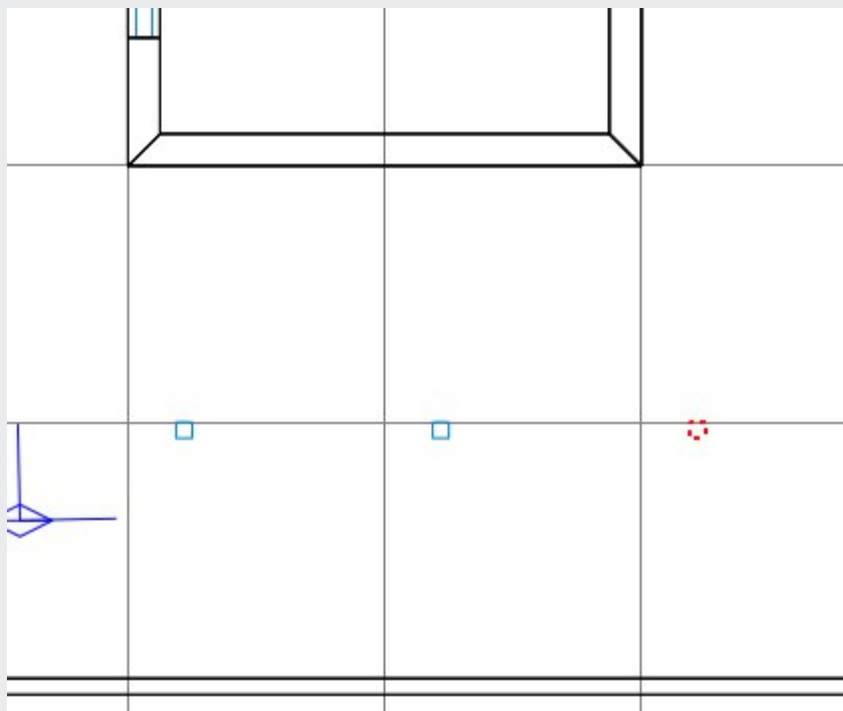
Select the fence brush, then in 2D, right-click and select Make Detail.

Select it again, and again right-click, but this time choose func/func\_explosive. Press N. Close the window and press N again.

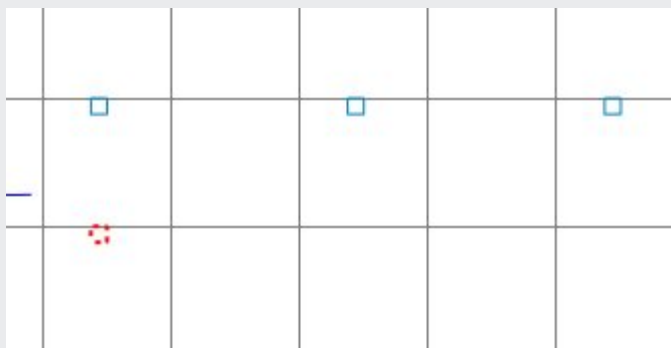
Tick UseShader box. Enter key "health" and value "100" and press return. A bullet does around 15 damage. Grenades do 200, and panzers do 400.

Enter key "type" and value "wood" and press return. Close entities window.

Press 8 and duplicate the fence post, then move it into line as shown below. Duplicate it again to make a third one in line.



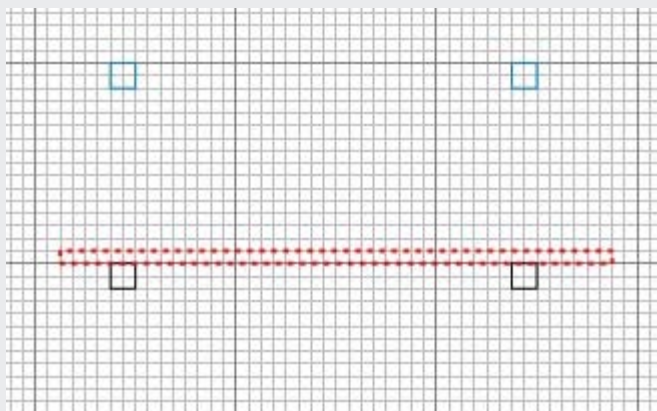
Press 7, and duplicate to make another fence post which we'll do something different with.



Right click in 2D and select Ungroup Entity. This returns the brush to an ordinary thing instead of an explosive one.

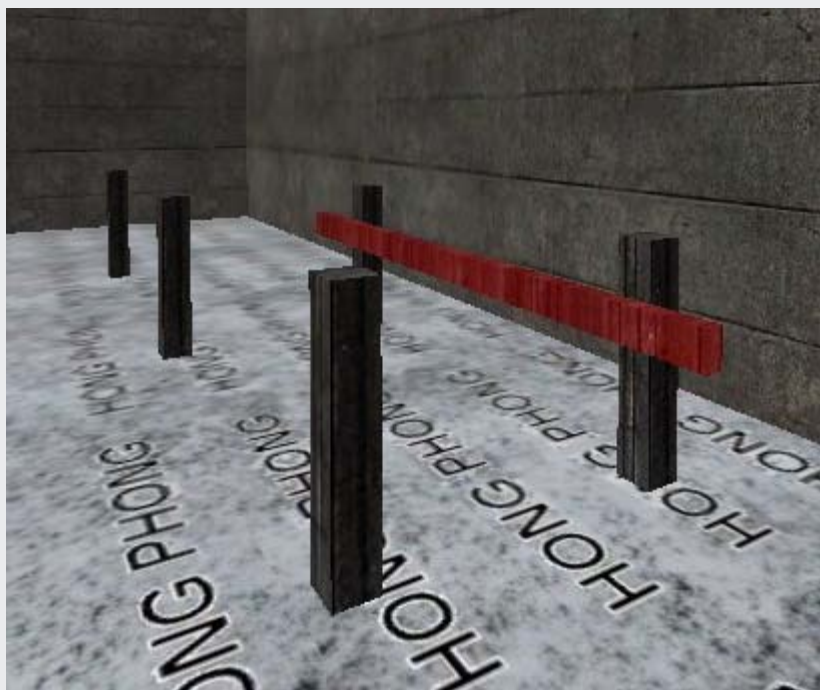
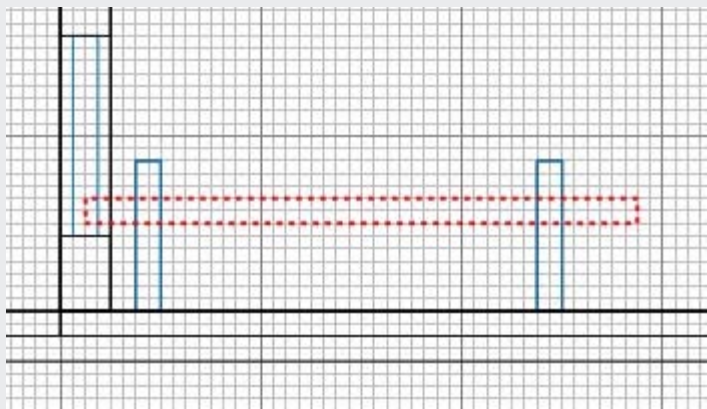
Duplicate the post. Press ESC.

Use grid scale **3** to create the linking fence plank as shown (this time allow wood texture to remain on this - do not caulk).





Ctrl+tab to see how tall the brush is, and make it fence bar size. Right-click and Make Detail the bar. Select the brush again.



Then select the upright posts so that you have all 3 brushes selected.





Then in 2D, right click and select func/func\_explosive. Again Press N. Tick UseShader. Put the cursor on the word "wood" in the value box and press return. This is a quick way to set the same value as used earlier. Also put "health" in the key and say "150" in the value and press return. Press ESC.

We now have 3 standalone posts that can be destroyed individually - and a fence that when destroyed, all 3 components go up together.

This is to illustrate that you can make a single explosive entity out of any collection of brushes (but not of models) and that you mustn't create something that might have its upright posts destroyed, but leave the crossbar intact floating in the air! This why you'll often notice that crossbars can be destroyed, but their posts cannot (eg Fuedump near the tank start).

Save, compile and go shoot some planks :)

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 13

### Topics

#### Making barbed wire

[Making the basic structure](#)

[Making the barbs hurt](#)

[Completing the structure](#)

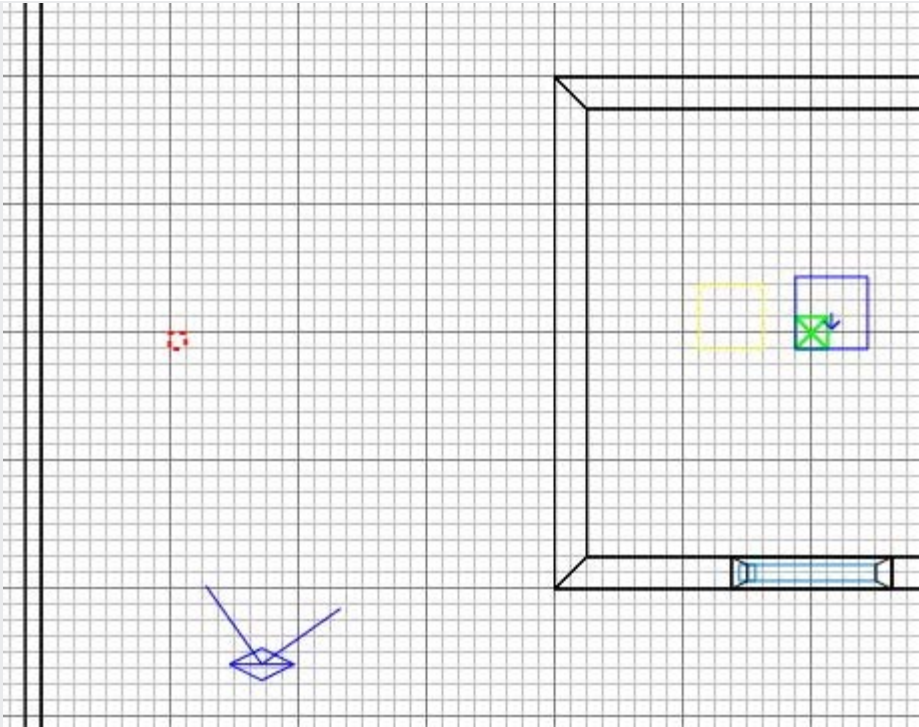
[Back to main menu](#)

#### Making the basic structure

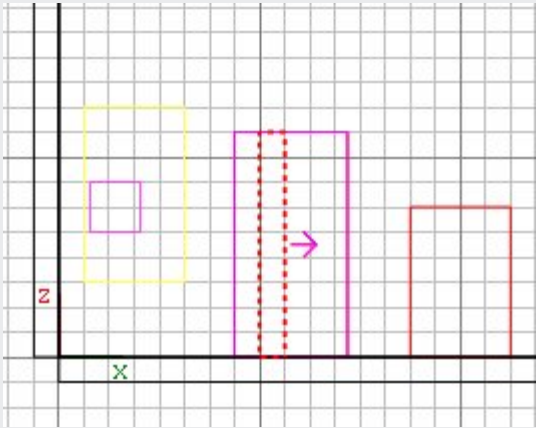
[\[Top\]](#)

We're going to make some barbed wire, because it demonstrates a number of new features and techniques and reinforces some we've touched on before. You'll use these a lot.

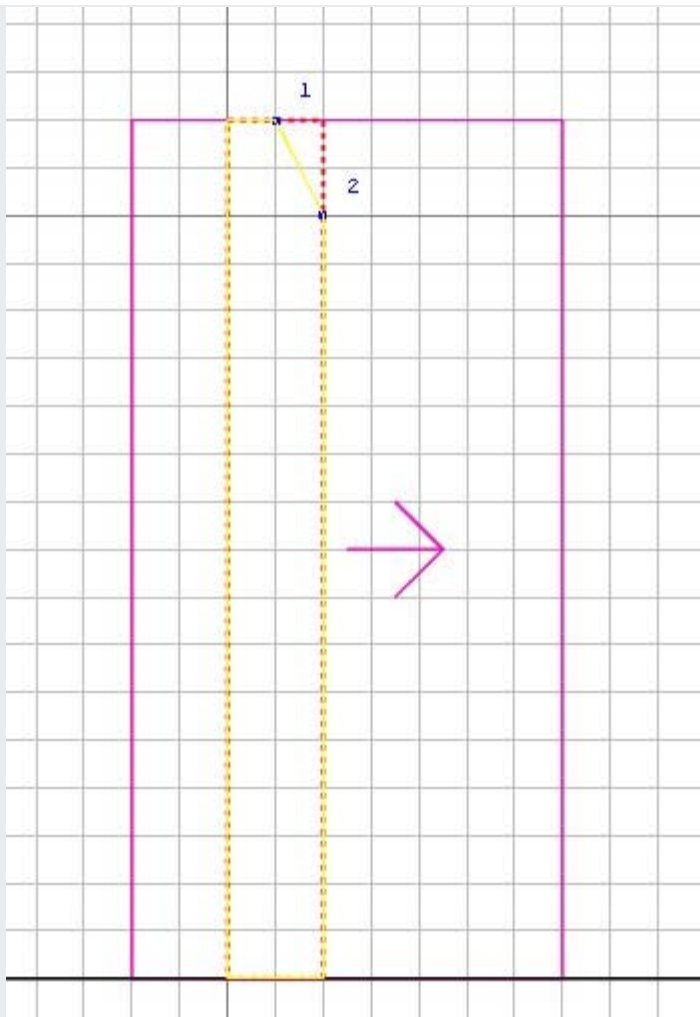
Run Radiant, open the map, and select one of the solitary explosive fence posts. Duplicate it, right-click in the 2D and Ungroup Entity so that it returns to being a normal brush. Drag the brush over to roughly where shown below. This will be the first post of the barbed wire.



Ctrl+tab and make the post a bit higher, say the height of a player start box:



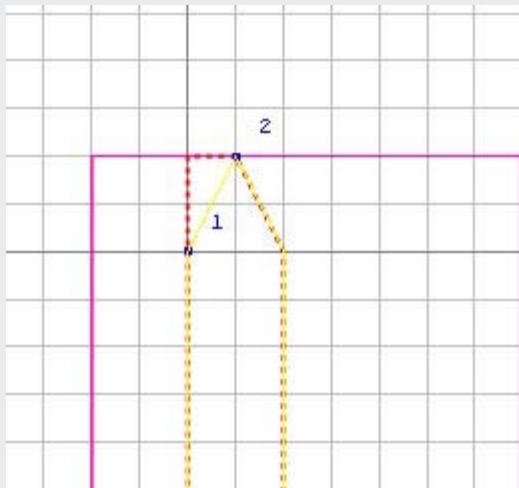
Press **3** and zoom in close. Turn on the clipper tool (X) and click where marked:



If the big chunk is not yellow, press ctrl+return so that it is.

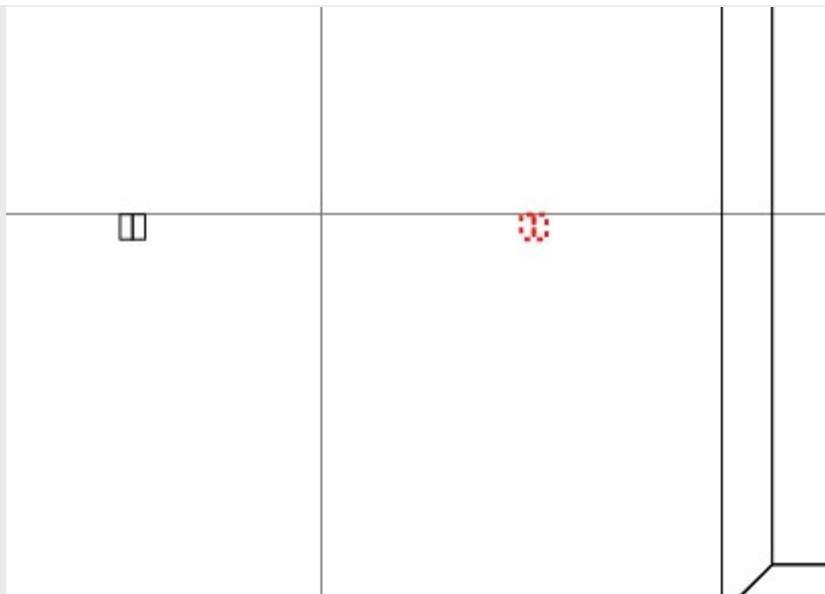
Press return to chop off the small chunk.

Click again as shown, make sure the big part is yellow and press return to chop off the other small chunk.



Press X to turn clipping off.

Ctrl+tab twice to get overhead view, zoom out again, press **8** and duplicate the post. Move it a bit further away as shown.

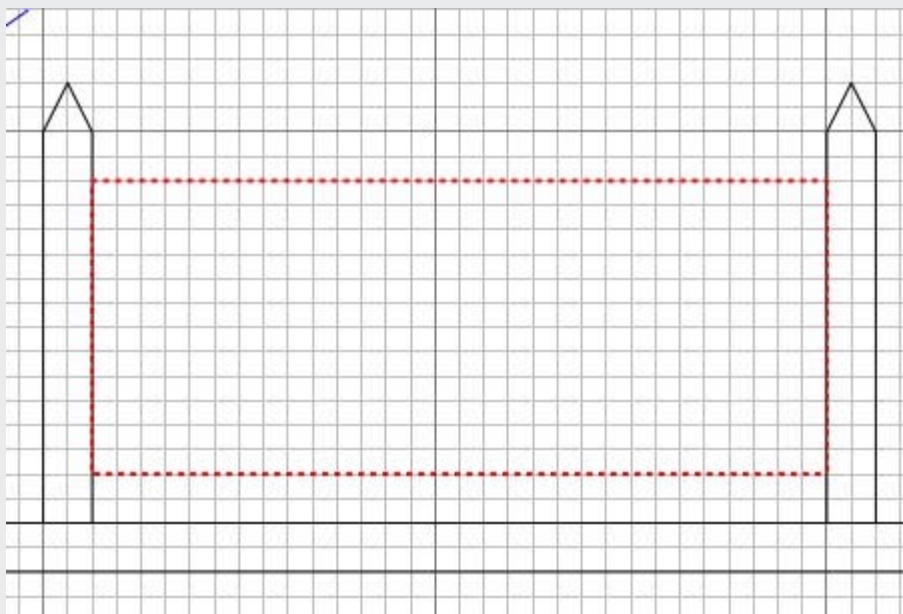


Press ESC, then **3** to return to a small grid scale. We're going to string some barbed wire between the posts.

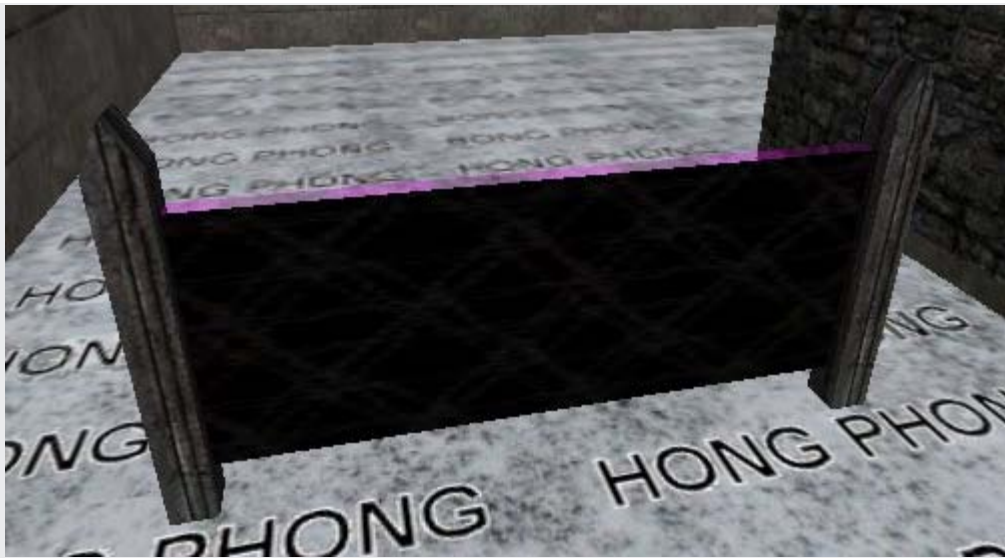
Draw a brush as shown. Click Textures\common and choose the NoDraw (pink/mauve check) texture for the whole brush. As the name suggests, faces with NoDraw texture are not drawn.



Ctrl+tab so you can set the height and position like this (press alt+2 or click View\Filter\Entities to prevent the display of entities (incl models) so that what you are working on is easier to see. You can filter all sorts of things out of the display under the Filter menu).



Right click in 2D and Make Detail. Then select the face that is nearest the Axis start box and click Textures\alpha barb\_wire. Then press S, set the width and height boxes to 2 and press **Fit**. We get the barbed wire texture compressed a bit to look better on the brush.



Ok so we have the barbed wire texture in place. But the barbwire face on the nodraw brush won't stop the player from running through it. We'll need to put a clip brush in there too.

Select the barbed wire brush and duplicate it.

The **common** textures should still be available to you in the textures window. Click on the **Clip** (pink/red check) texture.

Then drag the clip brush back so it occupies exactly the same space as the barbed wire brush.

The clip brush, like the one for the tree, will not be drawn. But unlike the Clip Weapon Wood, the Clip only stops player movement; it doesn't stop bullets or projectiles.

Press ESC.

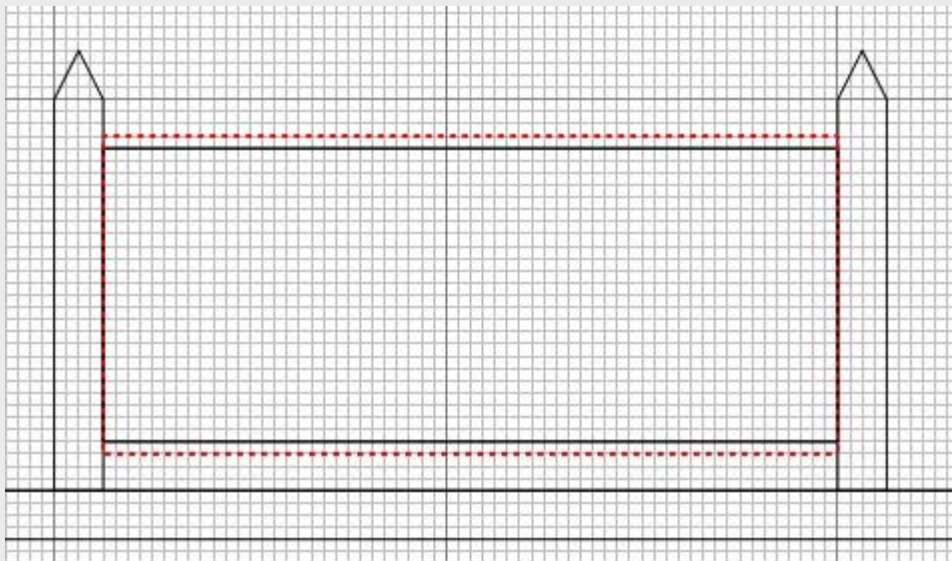
## Making the barbs hurt

[\[Top\]](#)

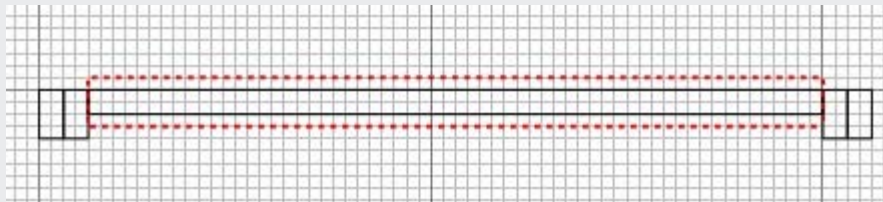
Ok so we have barbed wire that players can't run through - now we need to make it hurt those that try :)

Select either the clip or barbed wire brush. Duplicate it and click the **Trigger** (grey/pale yellow check) texture. Move the trigger brush to be in the same place as the others.

Press **2** for some fine adjustment, zoom in if needed, and make the trigger brush slightly larger than the other brushes.



Get an overhead view and again expand the trigger brush a little.



It's not enough to make a brush into a trigger brush - you have to then turn it into a trigger entity. Right click in the 2D and select trigger\trigger\_hurt.

Press N (if you don't see tick boxes, close the window and press N again).

Tick **Slow** - so the hurt gets applied once per second. Don't tick Silent, as we want the player to know it's hurting :)

Enter "dmg" as a key and say "15" as a value and press return. Close the window.

Press ESC.

**If you filtered entities, turn the filter off again now (alt+2), so that you will be able to see the trigger brush. Otherwise Radiant will filter the entity once you deselect it.**

Save your work.



Trigger brushes have many useful purposes, of which hurting players is just one. When a player enters the volume of the trigger brush, the trigger is activated. It might be to advise the player that he is near a constructible or a destructible, or that if he waves his pliers he could repair the tank or that if he presses Use he can pull a lever; or it might cause any scripted event. For example I have a trigger in the gap between the bank and the houses in Breakout. When a player passes through it, it triggers the playing of a dog barking WAV file - a handy warning to others that someone is approaching...

## Completing the structure

[\[Top\]](#)

Press **4**. Select the whole new structure and tilt it by clicking "Selection\rotate\arbitrary rotation..." and entering 30 in the X box. Click OK.

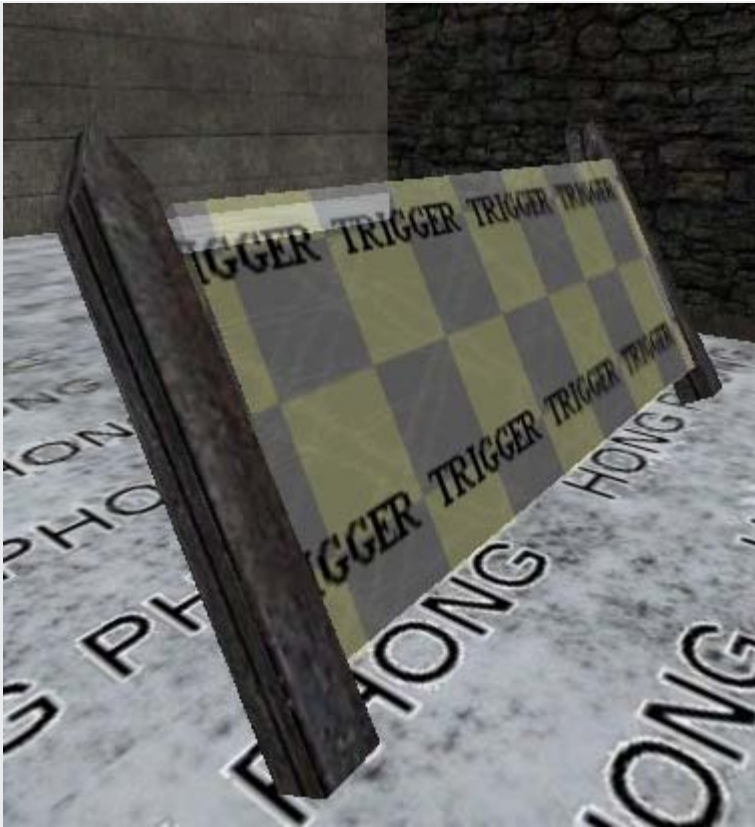
You'll notice that some of the the wooden textures have not tilted - we'll fix that.

Press ESC. Select both upright fence posts, and apply the wooden texture again (we want the wooden texture on the bottom of the posts this time, and they were caulk before this). Press ESC.

Select the **four** faces on each fence post that need properly aligning, so you have **eight** faces selected.

Press S. Change the Rotate Step value from 45 to 15. Then click the Rotate Offset downarrow twice to apply a 30 degree tilt. If you've done this right, you're fence post textures now line up properly. Press ESC.

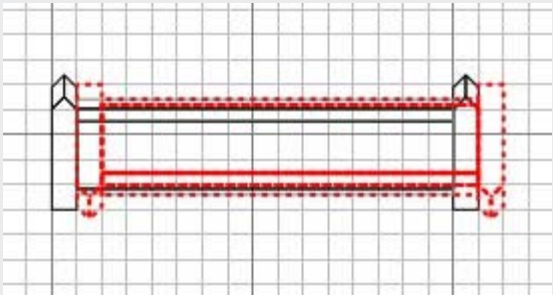




Check you have the overhead view. Select the whole structure, and duplicate it. Click the Y-axis Flip button on the menu bar.



Move the brushes next to the first set.



The wooden textures on the uprights are wonky again. Press ESC. Select the 8 affected faces. Press S. Enter 60 into the Rotate Offset box and click done. Press ESC.



Select the whole structure, and click File\Save Selected..., navigate to your prefabs folder, and save it as prefab\_barb as a map type. Then you can import it as and when wanted into another map.

Press ESC.

Compile and experiment with running into your barbed wire!

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 14

### Topics

#### Making a ladder

[Making something to climb up](#)

[Making the ladder brush](#)

[Back to main menu](#)

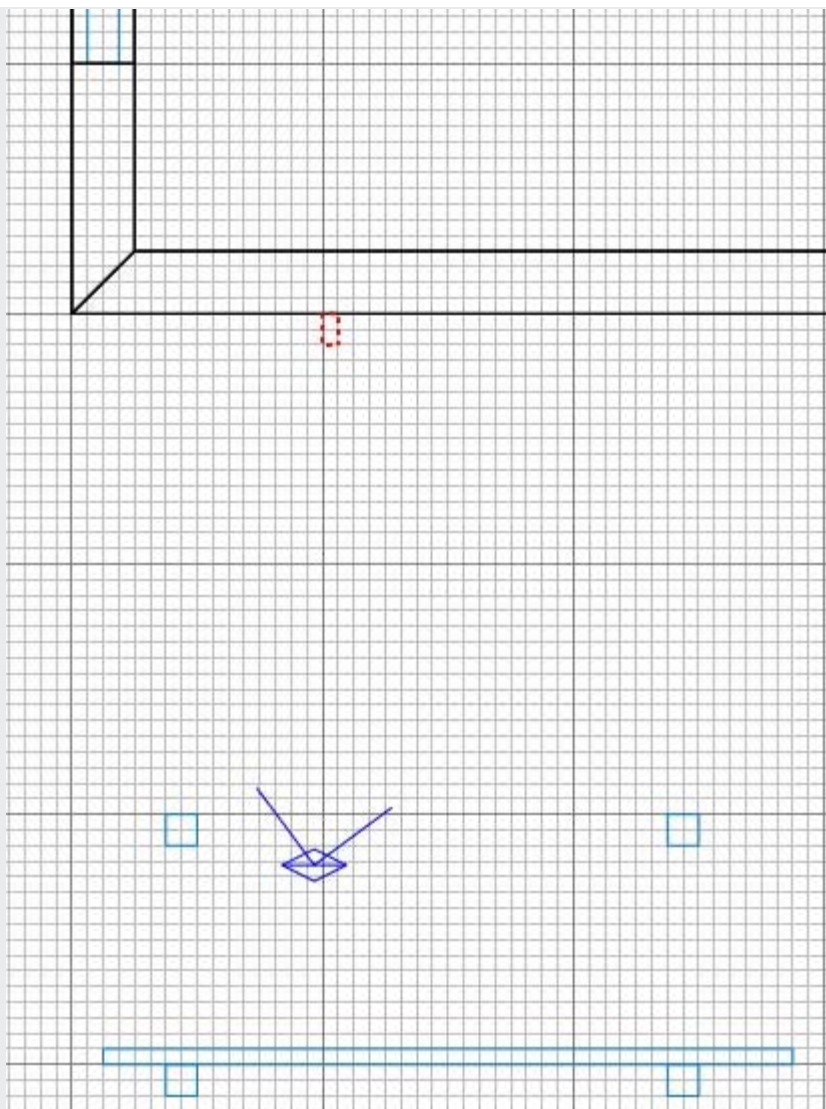
#### Making something to climb up

[\[Top\]](#)

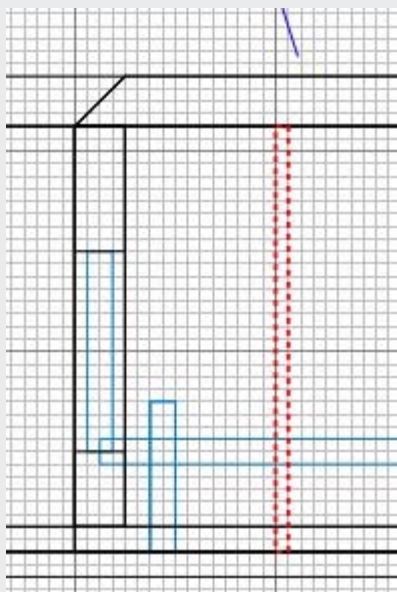
We'll make a ladder so we can get onto our building roof. Then we'll put a constructible MG42 on it in the subsequent lesson.

You can make any object/wall something a player can climb up. For simplicity we'll just stick to a ladder shape for now.

Run Radiant, open the map, and press **3**. Create a brush where shown.



Ctrl+tab to get the side view, and make the first leg of the ladder the right height.

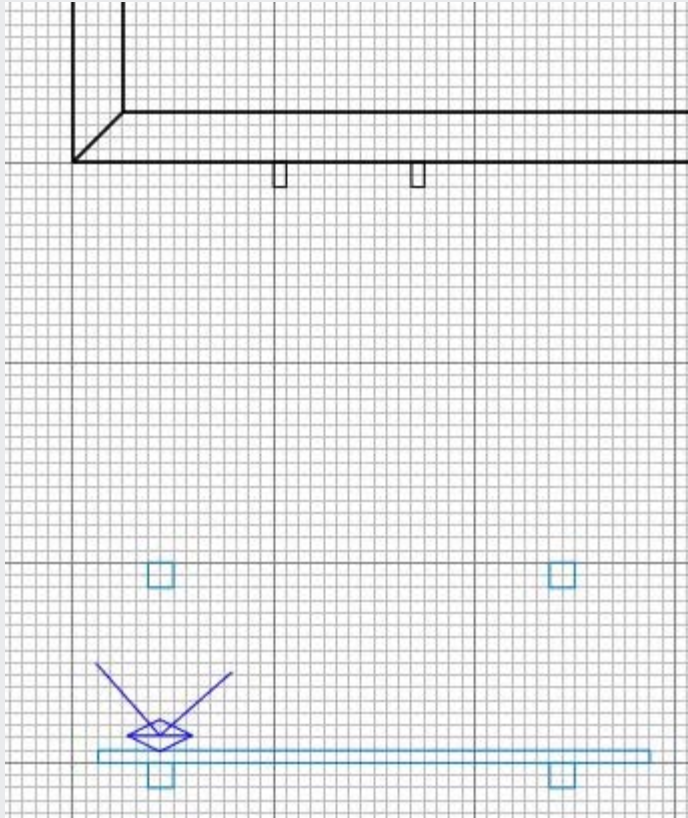


Tip: Sometimes there can be a lot of brushes around in the 2D window and you can't easily see which one is actually the floor you want to place a new brush on. In the 3D window just select the floor for a moment so you can see which brush lights up in the 2D window, and thus reveal where the "floor" is. The deselect the floor and continue to move/re-size the brush you are working on.

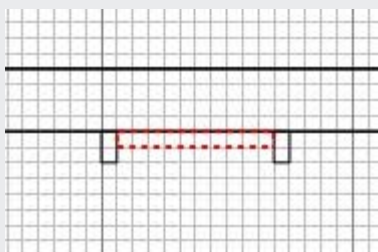
Apply Textures/metal\_misc ametal\_m03 to the whole brush. Ordinarily you would caulk the faces that won't be visible, but for the moment we won't, because we can create a prefab ladder which might be used in any free-standing context. Once the prefab is created, you would import it as needed and then caulk the unseen faces that suit where you've positioned it.

Set the brush to Make Detail.

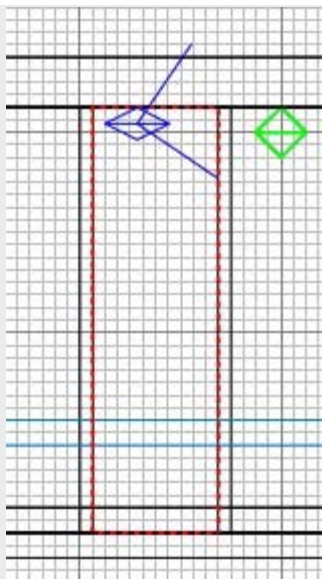
Top down view again, select and duplicate the ladder leg, and position it as shown and press ESC. (You may want to hide the sky if you keep selecting it by accident).



Draw the brush between them (**and caulk it**) that will serve as the ladder rungs.



Side view and position the rungs brush properly between the ladder legs.



Make the brush Detail.

Now select the rungs face that the would-be climber would go up and apply Textures/alpha ladder texture. Press S and enter 1 in the width box and 3 in the height and click **Fit**. Then click Done and press ESC. We use a texture that looks like ladder rungs instead of individual ladder rung brushes because:

- It's quicker for us to make
- It's quicker for ET to draw: one face instead of many faces on many brushes

If you want fancy rungs and you know there is no pressure on the FPS and you can be bothered, you can of course make individual rungs, bricks, whatever, to indicate a climbable structure to the players. Make sure you make them Detail !



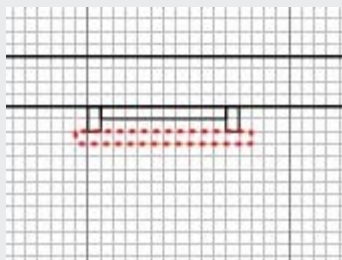
## Making the ladder brush

[\[Top\]](#)



Ok so we have something that resembles a ladder. Now to make it climbable.

Top down view, select the rungs brush and duplicate it. Resize and reposition it as shown.



Then apply Textures/common ladder texture (lavender/mauve check) to the whole brush. We now have a functioning ladder. To make it more useful, select the ladder legs and rungs brushes too so we have all 4 brushes selected.

Right-click in 2D and select func\_group. This has no effect on ET but is useful in Radiant. It means we can select the group of brushes with one shift+alt+click. Handy when importing it and you need to move it.

You could File/Save Selected... if you want to save this in your prefabs folder as prefab\_ladder.map.

Press ESC.



Save, compile and make sure you can scramble up your ladder :)

[Next lesson](#)





# ET Mapping Tutorial

## Lesson 15

### Topics

#### Making a constructible MG42

[Making the constructible MG42 nest](#)

[Including the script](#)

[Back to main menu](#)

#### Making the constructible MG42 nest

[\[Top\]](#)

Generally speaking, making all the components for a constructible item is too much hassle to do from scratch each time. So I use the Blue Peter method: "Here's one I made earlier".

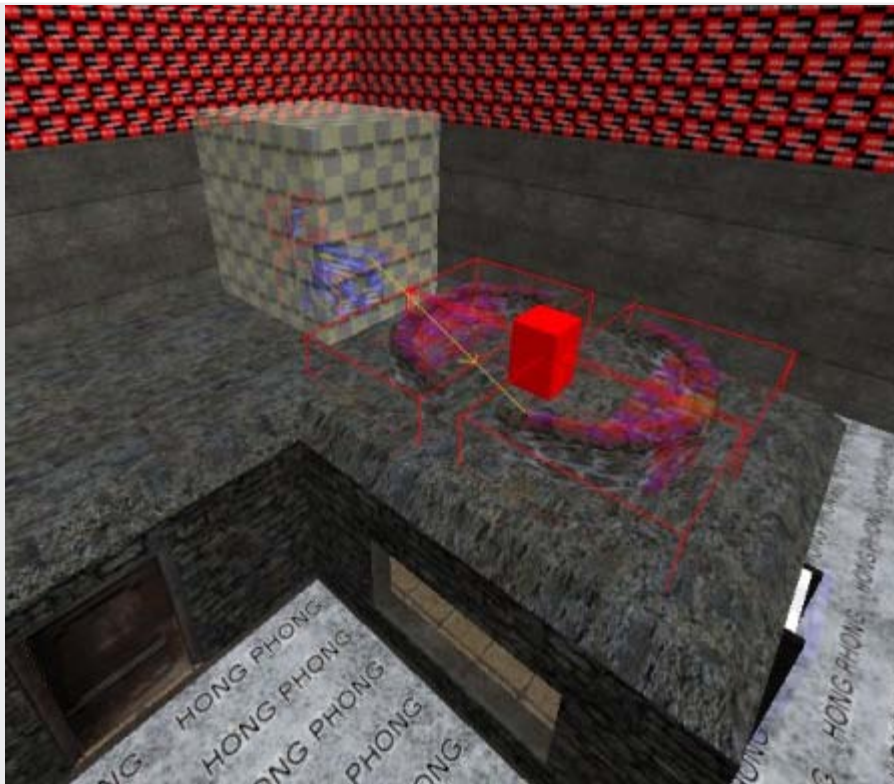
By that I mean for things like constructible MG42s, Command Posts and Health & Ammo Cabinets, you just use a prefab made before, tweak it a bit to get the placement and names right, and away you go. Saves tons of time.

So I will provide you a link to the [MG42 map elements](#) for you to import, which includes the script chunk you need to make it work. The rest of the lesson will be devoted to explaining what the components are and what their settings mean, so you can reproduce them in your map as you need them.

Download the zip file, and unzip the script file to your maps folder, and the map file to your prefabs folder. The script file is complete with the game\_manager section too, so it can replace the previous tutorial.script file.

We'll deal with the import of the MG42 into the tutorial map first.

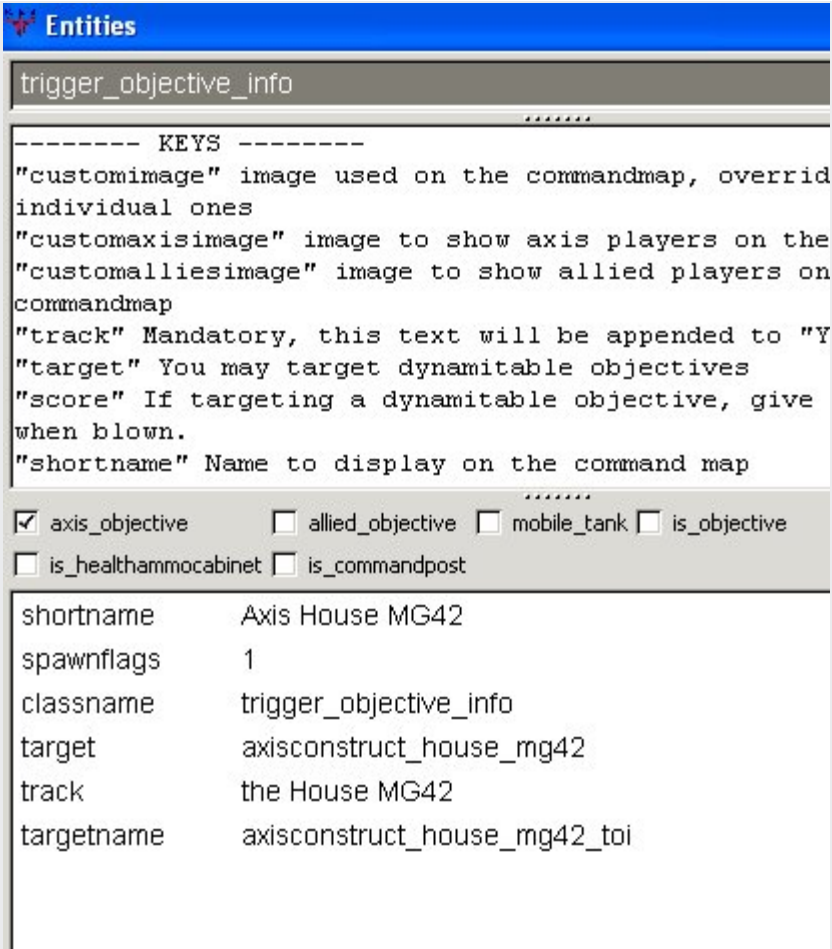
Run Radiant, open the map, and import the prefab\_axis\_constructible\_mg42 - I saved this in the right place so that when you import it, it will already be placed on the roof of the house. If your house is not the same dimensions, you'll need to drag the whole selection into place yourself.



Let's go through the components so you understand what you've got.

Shift+alt+click the large trigger cube. This will select the large cube and a small cube within it. The one within is the origin brush, which tells ET where the "middle" of the trigger is, regardless of the odd shape that a trigger brush can be.

Press N to see the details of the trigger.



This trigger is not a trigger\_hurt, it is a trigger\_objective\_toi (Trigger Objective Info), which means it is used to determine when a friendly engineer will see the pliers hint.

This is an axis constructible MG42 we are making, so the **axis\_objective** box is ticked.

The **shortname** is what gets displayed on the Command map.

The **spawnflags** is a number which represents the set of tick box settings you've chosen.

The **target** is the targetname of the entity to be constructed. When the pliers-waving is done, this is the name of the procedure in the script that will get executed.

The **track** is the text that gets shown when a player enters the trigger volume, as in "You are near...".

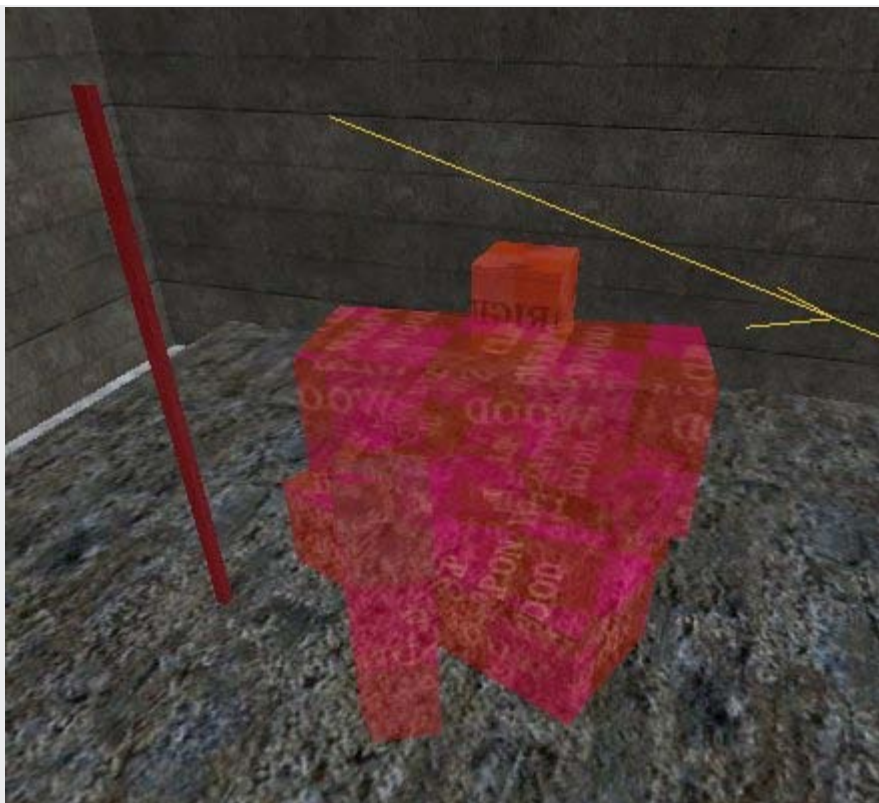
The **targetname** is the name given to this toi.

You will see that **axisconstruct\_house\_mg42** gets used as the common naming thread to join all the components together.

Close the entity window and press H to hide the trigger. By successively selecting components and hiding them, you can be sure you have seen everything.

Press shift+M to filter out the models, making it easier to select the clip brushes of the crates.

Shift+alt+click the crate brushes. If this doesn't select the brushes, keep doing it until it does. You can see they will stop bullets with a wooden sound.



Press shift+M again to see the models again, and press N.

Entities

script\_mover

----- KEYS -----

"modelscale" Scale multiplier (defaults to 1, uniformly)  
"modelscale\_vec" Set scale per-axis. Overrid if you have both, the "modelscale" is ignored  
"model2" optional md3 to draw over the solid  
"scriptname" name used for scripting purposes  
"health" optionally make this entity damagabl  
"tagent" target entity a mounted gun will be  
"gun" variable gun type, only other option ot  
"browning"

☐ triggerspawn
☒ solid
☐ explosivesdamageonly
☐ resurectable

☐ compass
☐ allied
☐ axis
☐ mounted\_gun

spawnflags

2

classname

script\_mover

targetname

axisconstruct\_house\_mg42\_clip

scriptname

axisconstruct\_house\_mg42\_clip

A **script\_mover** entity has been used here, but in this instance this is a bit misleading, because the crates don't move. The choice of a script\_mover entity was simply a convenience and other entity types would have done. This example for first taken from Goldrush and there's been no point in changing it.

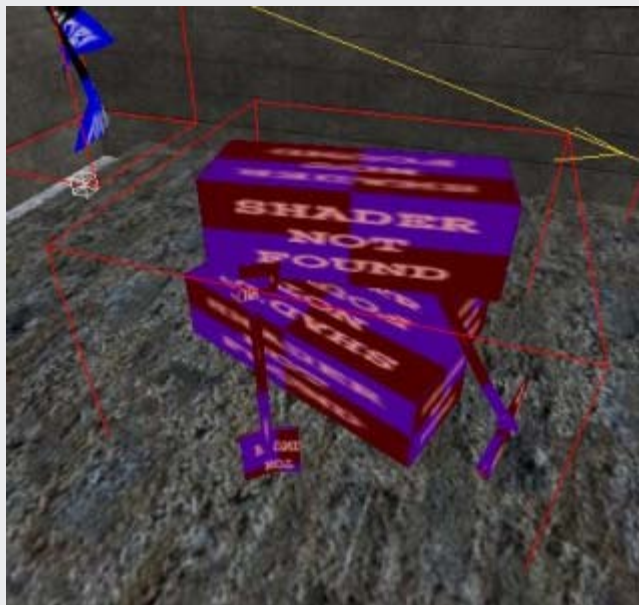
What matters is that this is just a clip brush so that the players don't run through the crate models - remember models don't impede player or missile movement.

The **solid** box is ticked because this is a script\_mover which by default also doesn't impede player movement.

The **targetname** and **scriptname** are both axisconstruct\_house\_mg42\_clip. Convention has it that if an entity is going to have a script (ie a procedure appear in the script file) then you make the procedure name (scriptname) the same as the entity name (targetname). But just because an entity has a scriptname, it does not mean you necessarily have to have a script procedure for it. In other words, no harm done by having a scriptname.

Close the entity window and press H to hide the clip.

Now shift+alt+click the crates model and press N.



Entities

misc\_gamemodel

```

----- KEYS -----
"model" arbitrary .md3 file to display
"modelscale" scale multiplier (defaults to 1x, and scales
uniformly)
"modelscale_vec" scale multiplier (defaults to 1 1 1, sca
each axis as requested)
"skin" .skin file used to define shaders for model
"trunk" diameter of solid core (used for trace visibility
collision (not ai pathing))
"trunkheight" height of trunk
"frames" number of animation frames


```

☐ orient\_lod ☐ start\_animate

targetname	axisconstruct_house_mg42_materials
model	models/mapobjects/cmarker/cmarker_crates.md3
origin	656 626 152
classname	misc_gamemodel
angle	270
skin	models/mapobjects/cmarker/axis_crates.skin




This is the model of the crate boxes. A **skin** is a texture, is applied to the crates model to make them axis crates.



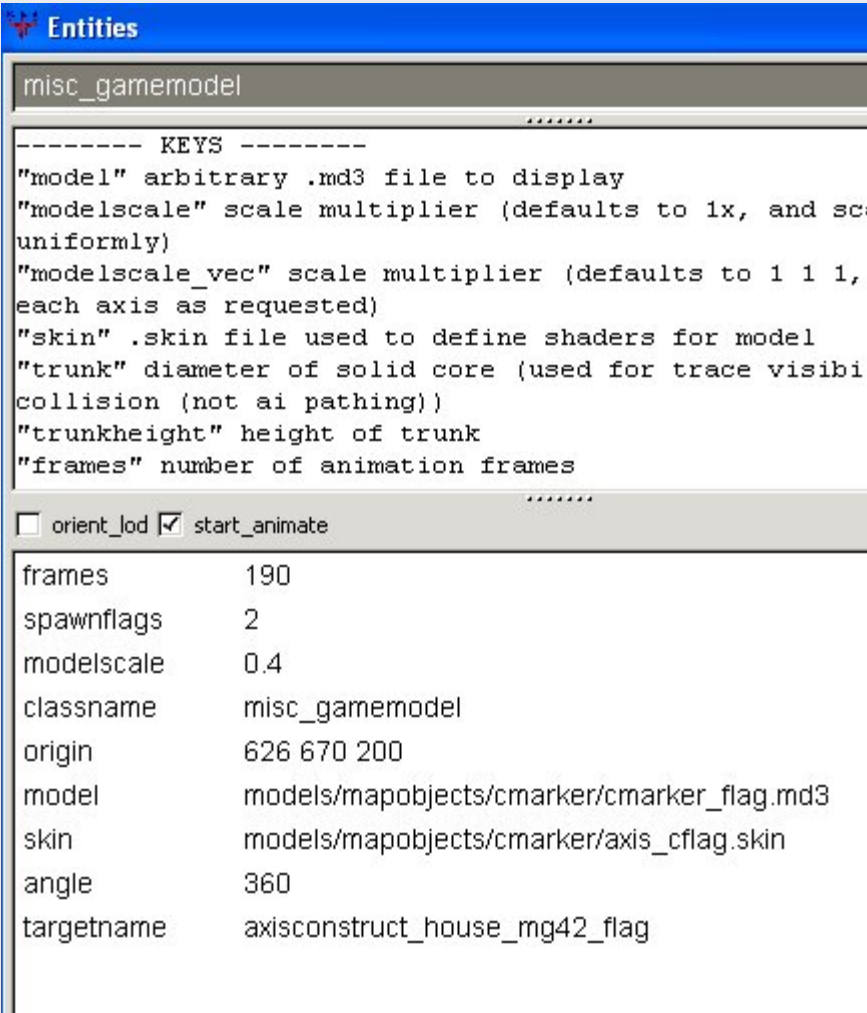
Models are not rotated in the map in the same way as brushes. That is, if you try to rotate a model it won't do anything. To adjust the angle of a model, you use the "angle" key and a value in degrees. If angle is not specified, 0 (East) is assumed. These boxes have been rotated 270 degrees.

The **classname** is misc\_gamemodel rather than misc\_model.

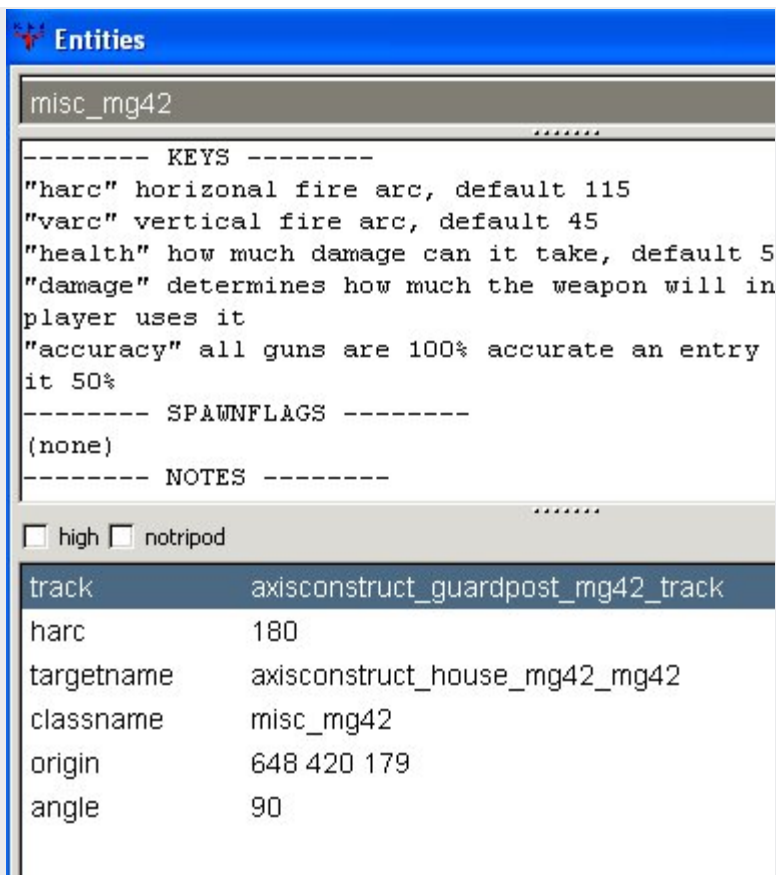


Use misc\_model for any sundry model, no matter how big or small, that will just sit there and do nothing, like trees.  
Use misc\_gamemodel if the model needs to be referred to in the script. There is an exception to this which will be covered later.

Close the window and press H.  
Shift+alt+click the flag. Press N.



This model is animated: the flag flaps in the breeze. Hence the **start\_animate** and **frames** settings.  
The 360 degree angle seems pointless, but hey, that's how it is in Goldrush so I've left it alone.  
Close the window and press H.  
Ok, that's dealt with the crates side of the MG42. Now select the red box which is the MG42, and press N.

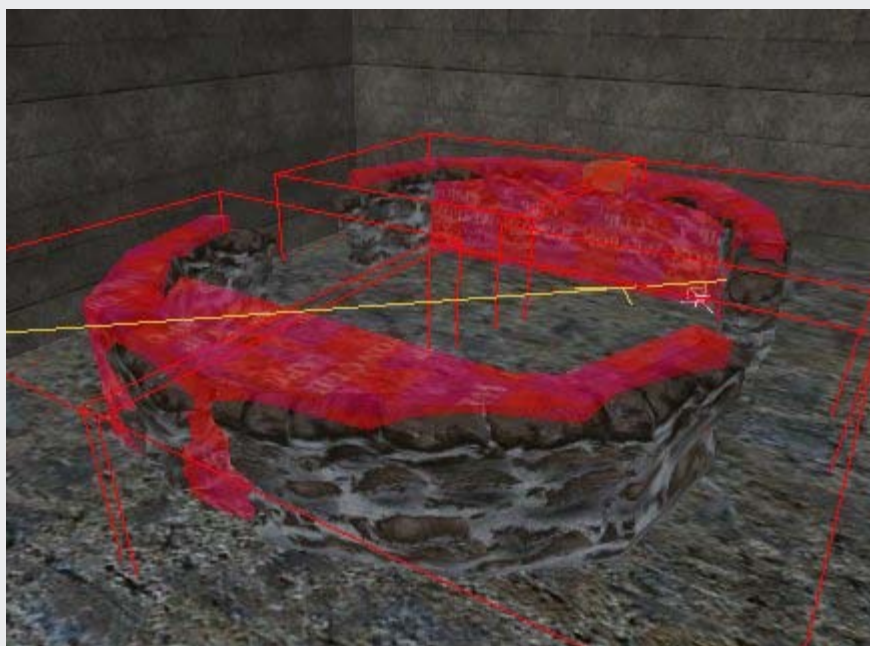


I have given the MG42 a horizontal arc of 180 degrees, which allows it to spin completely round. Normally the default arc is ok so **harc** is not specified

I wanted the built MG42 to face north, so I gave it an **angle** of 90 degrees.

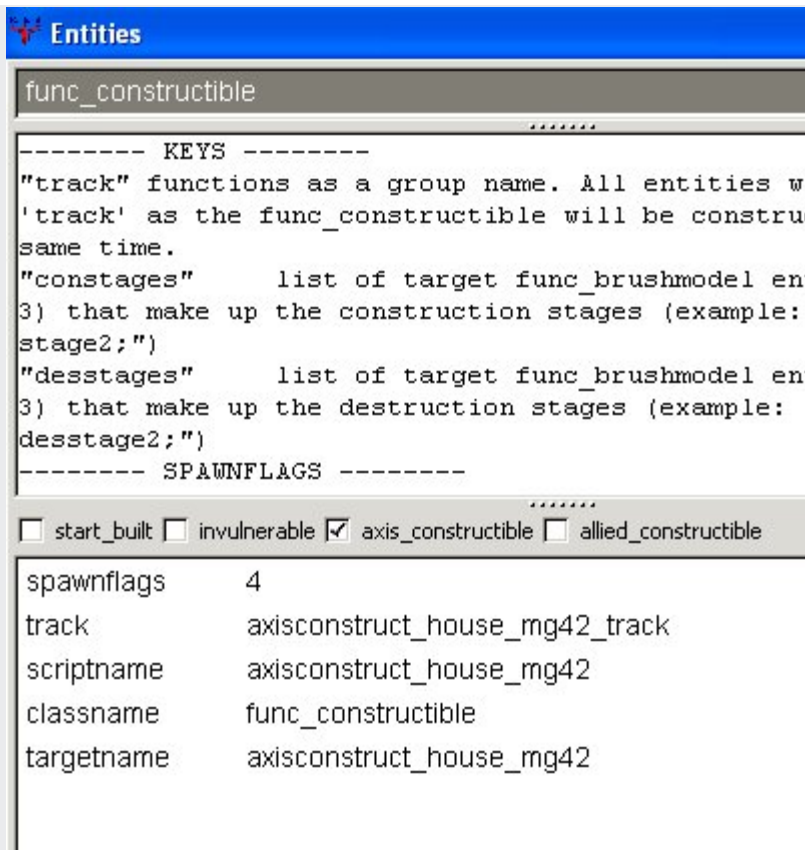
Close the window and press H.

Shift+alt+click the clip brush that envelopes the sandbag models.



Press N.



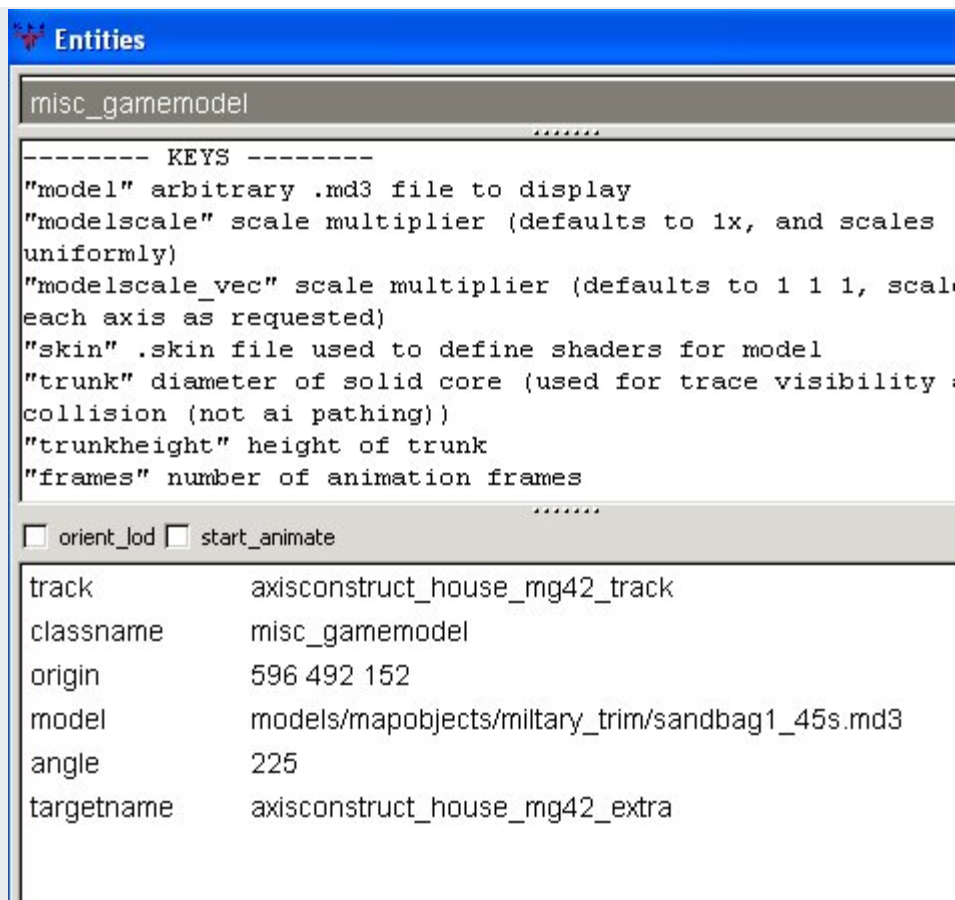


The clip brush is a **func\_constructible**, ie the subject of a toi trigger.

The MG42 + sandbags are to be built by axis, so the **axis\_constructible** box is ticked.

The **track** in this instance is used by ET to tell it to consider anything else with the same track value to be part of the constructible. So when the clip gets built, the models with the same track value undergo the construction with it.

Close the window and press H. You are left with just 4 models. Shift+click any of them.



You know enough now from previous explanations to be able to interpret the settings shown.

Press shift+H to reveal everything again. Save your work and compile it. Don't test it yet.

## Including the script

[\[Top\]](#)

When your script gets large, you would just copy/paste the **axisconstruct\_house\_mg42** portion of the script supplied into your own script file. For now I have supplied the whole script file for convenience.

Open the script file in Wordpad or similar.

You will see the MG42 script procedure. I have commented what's going on throughout the procedure to help you start to understand scripting, but this is the gist of what happens:

- At game start, the various MG42 components are set to their initial required state, eg the MG42 wants to start invisible.
- The func\_constructible can be destroyed by a satchel or dynamite (class = 2). Class 3 would be dynamite only.
- Once an engineer has waved his pliers long enough, the **built final** procedure gets executed, to bring the MG42 into being.
- If the engineer started waving but doesn't finish in time, the **decayed final** procedure gets executed.
- Finally if the MG42 is destroyed, the **death** procedure gets executed.

With your script complete, you are ready to test the map: make sure you can construct it as Axis, and destroy it as an Allied cov ops.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 16

### Topics

#### Secure doors

[Making a door secure to your team](#)

[Back to main menu](#)

### Making a door secure to your team

[\[Top\]](#)

This is a nice easy task :)

Run Radiant and load up the map.

Select the door using shift+alt+click.

Press N.

Enter **allowteams** as a key and **axis,cvops** as the value and press return. The axis door is obviously the more common, but you can make allied secure doors by putting **allies** instead of axis. And if you don't put the cvops part in, then enemy cov ops in uniform cannot open the door.

Close the window and press ESC.

To indicate the secure nature of the door, we should put a new door texture on.

Select both faces of the door and apply Textures/doors door\_m01asml\_axis texture (if you are not sure which one this is, click on the texture you think it is, and check the name that comes up bottom right of the output window.)

Press S and **Fit** the textures (1 wide and 1 high).



Close the window, press ESC, save and compile.

You should find now that only Axis can open the door. You can't test for cov ops alone so you have to ensure your spelling of **axis,cvops** is accurate.

Easy lesson!

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 17

### Topics

#### Lighting

[Ambient light](#)

[Light entities](#)

[Put on the red light](#)

[Adding a corona](#)

[Light-emitting textures](#)

[Back to main menu](#)

### Ambient light

[\[Top\]](#)

Run Radiant and open your map.

You can give your map an overall minimum level of light, regardless of the location and lighting you've included. This is the ambient light setting. It is optional.

If you feel your creation is just a little too gloomy generally, you might consider setting the ambient light.

To set it, select a regular brush and press N to get at the worldspawn settings.

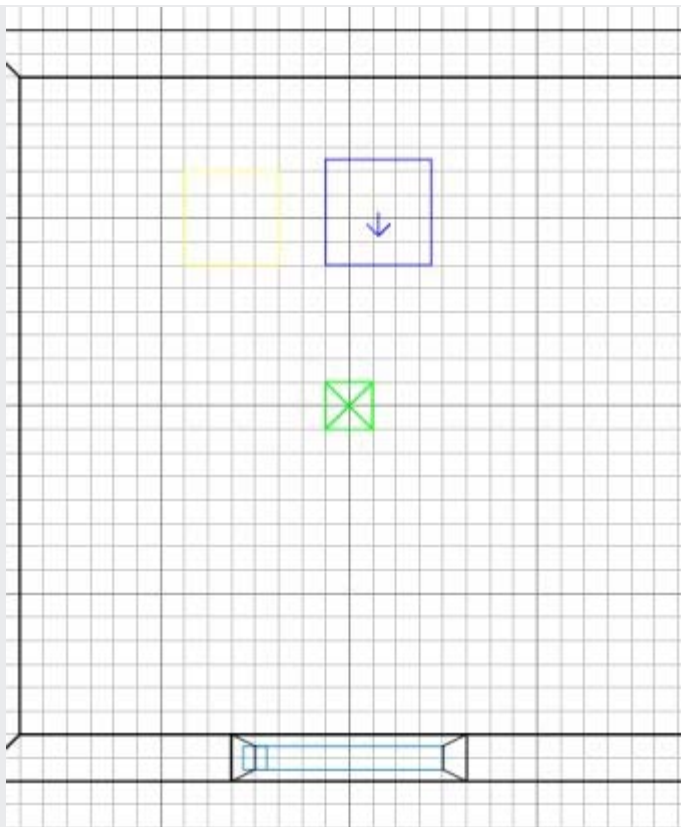
Enter "ambient" in the key and a number say "10" in the value and press return. Close the window. I wouldn't really go above 20 as it starts to wash things out and reduces the appearance of shadows. As a guide, Breakout has a setting of 20; 6flags has a setting of 10; and 2tanks has no ambient light.

### Light entities

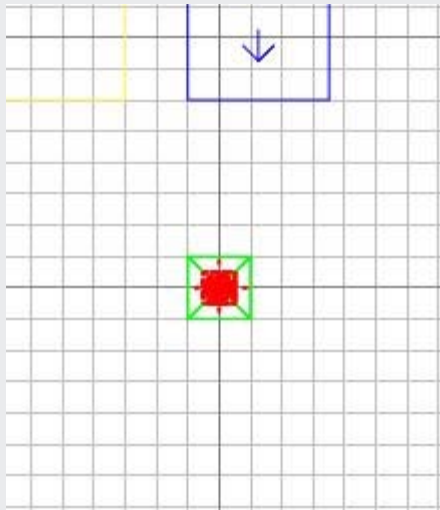
[\[Top\]](#)

We've already added 3 light entities to the tutorial map. We'll look at them in more detail and make them look better.

In 2D overhead view, zoom in to the room with the Allied start point in it. I have moved the start up a little so it doesn't confuse the picture with the light that is overhead.



Right click on the centre of the green light box and select misc/misc\_model. Then navigate to mapobjects\light and select cagelight.md3.



I've found that some light models either add hours to the compile time, or crash the game. These are my findings:

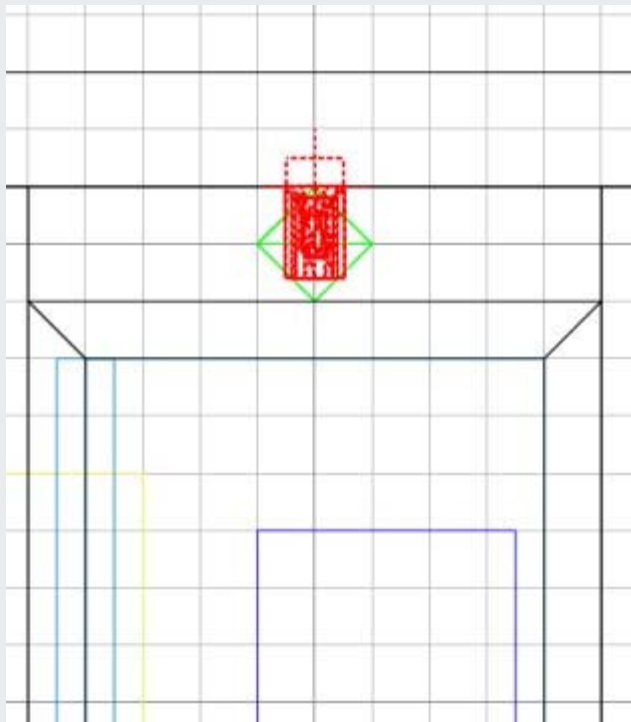
**These models are safe:**

cagelight  
cagelighta  
cagelighttr  
p\_nolight  
lantern

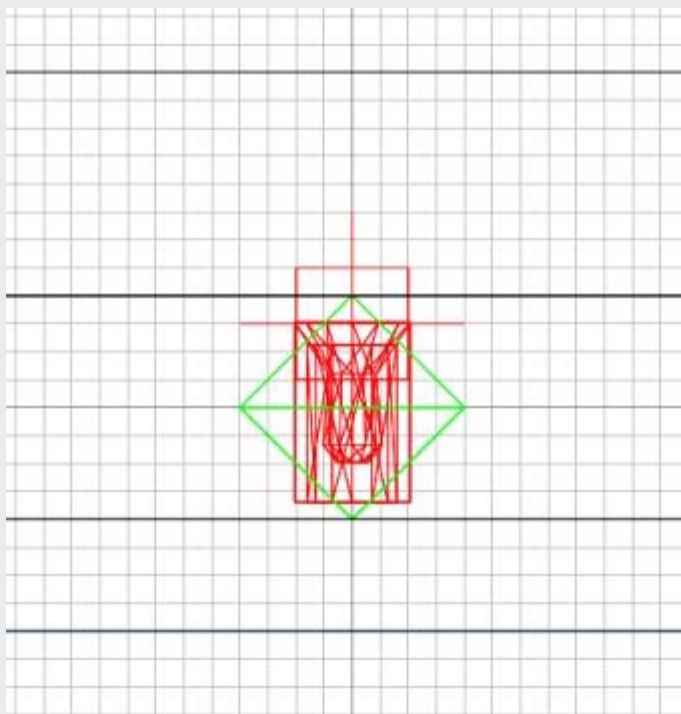
**These models should be avoided**

pendant10k  
sdsconce3

Get a side view and move the lantern up to the ceiling.

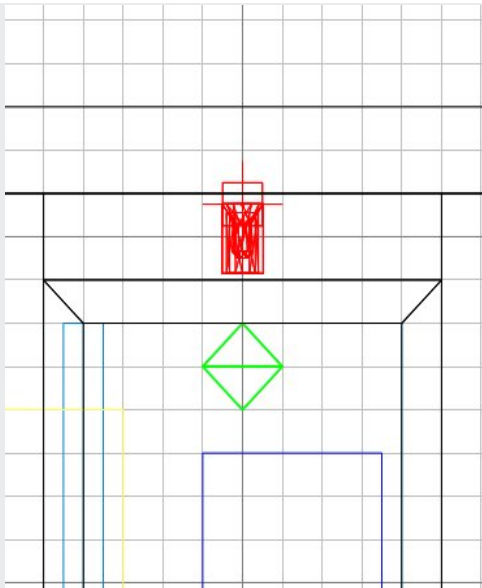


Press **2** and lower the lantern one notch. We have to bring the lantern ever-so-slightly out of the ceiling because you can't put a model into a structural brush. If the ceiling had been a detail brush you could have put the model partially into it. If you find you can see the gap between the model and the ceiling, make yourself a detail brush to run along the ceiling and close the gap between the ceiling and the light - make it look like electrical trunking for example.



Move the light entity down a bit. It usually is just below the model.





If you put the light too close to the model it will cast a shadow of the model onto the ceiling. Which might be ok, depends on what effect you want.

Compile and test that you see an illuminated cagelight model and that the light illuminates the room satisfactorily. (In Radiant the cagelight will appear unilluminated.)

Put on the red light

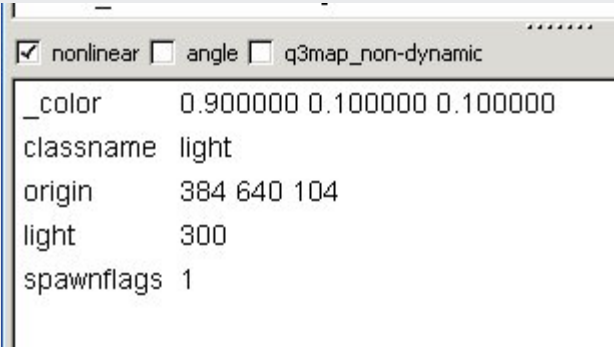
[\[Top\]](#)

Let's give the cagelight a red bulb. There is already a model for this.

Select the cagelight model and press N. Press the **model** button at the bottom right of the entities window.

Navigate to mapobjects/light and select cagelightr.md3.

Press ESC then select the light entity and press N. Enter the "\_color" key and "0.900000 0.0100000 0.100000" value as shown, to make this light red instead of white.



The numbers of the \_color value are the amount of RGB (Red Green Blue) in the light. Values are 0 to 1 to 6 decimal places.

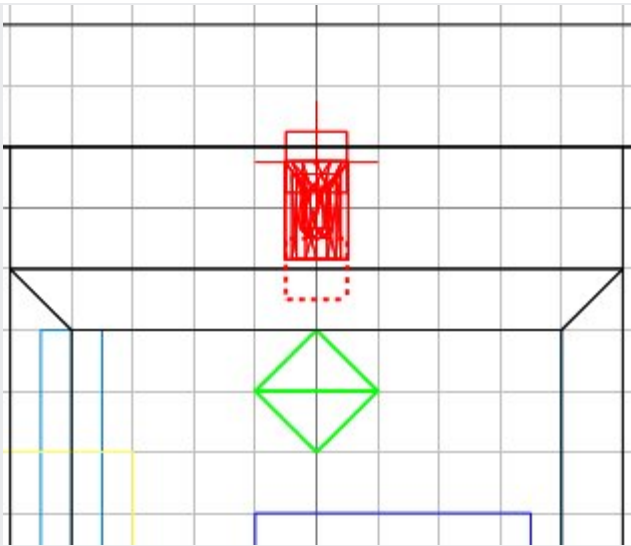
Close the entities window and press ESC.

Let's give this light a corona too. The corona is the glow that a player sees, which is bigger the more of the light source he can see.

Adding a corona

[\[Top\]](#)

Right click in the 2D view at the point shown, and select **Corona**.



Then press N. If you haven't clicked on any other entity keys in the entity window since setting the light color, the `_color` stuff will still be in the key and value boxes. If so, just press return to input them. This is a handy shortcut to setting the corona colour (although coronas are generally white because most light sources in a map will be white). If the key is not `_color` already, one trick is to select the light entity, press N and click on the `_color` row - this puts their values into the key/value boxes. Then close the window, press ESC and select the corona. Now when you press N you'll have the boxes pre-filled.

If your corona is still selected, press ESC.

Compile and test the red light and make sure the corona is placed realistically. Two things to note: one is that a coloured corona doesn't show up as much as a white one, and also that the coloured light has actually been shone through the closed door and has coloured the snow!

So this would be no good in a map - either you'd have to move the door or light, or make the light white, which is what we'll do:

Select the cagelight model, press N and click **model** and change the model back to `cagelight.md3`. Close the entity window and press ESC.

Select the corona, press N and click the `_color` row then click **del key/pair**, close the window and press ESC.

Select the light, press N and click the `_color` row then click **del key/pair**, close the window and press ESC.

Compile and test, and you'll see the effect is now fine.





## Light-emitting textures

[\[Top\]](#)

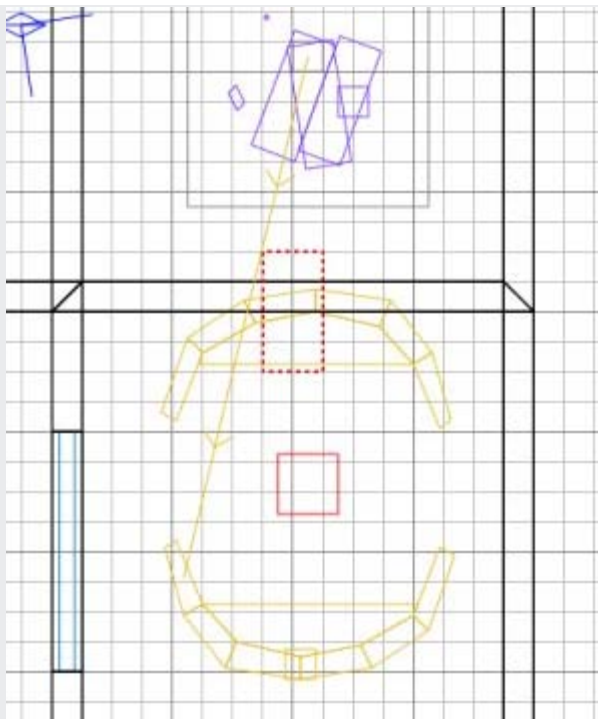
As you know, textures can give off an ambient light of their own, such as if you put a sky texture on a ceiling or wall face. This is done by creating a shader (a set of instructions) for the texture. We won't need to do this ourselves because there are plenty of pre-supplied light textures with the shaders already done.

Select both of the other lights in the room and delete them.

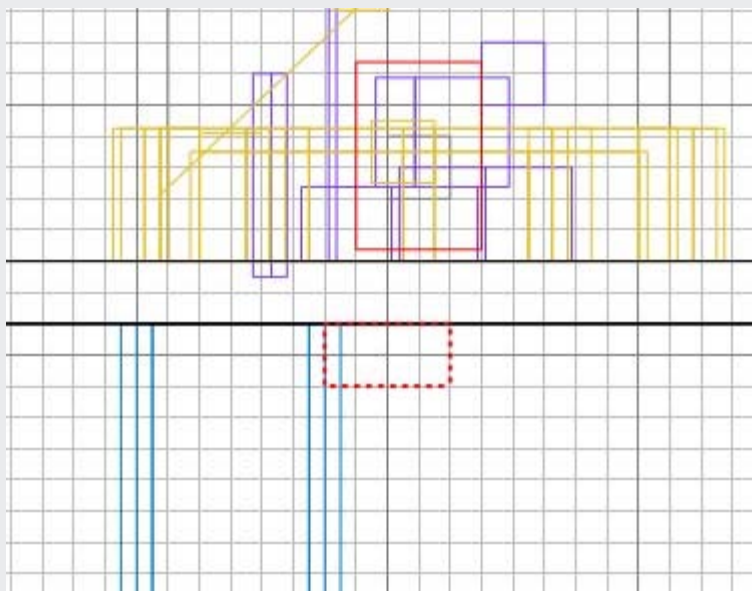
In 2D topdown view, create a brush which we will make into a ceiling light.

Press shift+M so that the models aren't shown. This makes it easier to see what we're doing.

Press **5** and draw a brush as shown. Caulk it and **make it detail**. Select the brush again.



Ctrl+tab so you can move the brush up to the ceiling. Press **4** so you can get it flush against the ceiling.



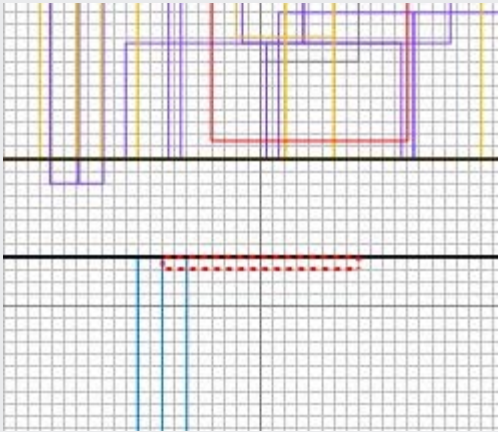
Apply a metal texture to all of the 4 side faces of the brush (shift+ctrl+click in 3D on one face and shift+alt+ctrl+click the other 3). Metal\_c07 will do, and it's already listed in the texture window.

Now shift+ctrl+click the bottom face, which will be our light texture.

Select Textures/lights and apply light\_xlight3\_4000.



The texture is misaligned on the face, so press S and click Fit and Done, and press ESC.  
Select the brush, and shrink it upwards so it's not such a chunky brush. Use grid scale 2 to achieve this.



Press ESC and press 5 to put the grid scale back to something sensible. Press shift+M to remove the models filter if you haven't already.  
Compile and test and you should see something like this:



[Next lesson](#)



# ET Mapping Tutorial

## Lesson 18

### Topics

#### Detail brushes vs structural brushes

[Detail brushes vs structural brushes](#)  
[Back to main menu](#)

#### Detail brushes vs structural brushes

[\[Top\]](#)

What's the point in having **Detail** brushes and **Structural** brushes?

Imagine two large room walls, one detail and one structural. This table shows the basic difference.

Brush type	Blocks player line of sight?	Blocks program line of sight?
Structural	Yes	Yes
Detail	Yes	No

What this means is that from a player's perspective, he can't distinguish "detail" brushes from "structural" ones - but the program can.

ET has to decide what elements of a map might need to be drawn for a player every time it refreshes the display. And it does this by judging what is in a player's line of sight, based on what **structural** brushes are in the way.

When Radiant builds the map, it breaks the whole volume of the map down into chunks (called portals), whose edges are made up of structural brushes. I'll use this picture to illustrate what this means in practice.

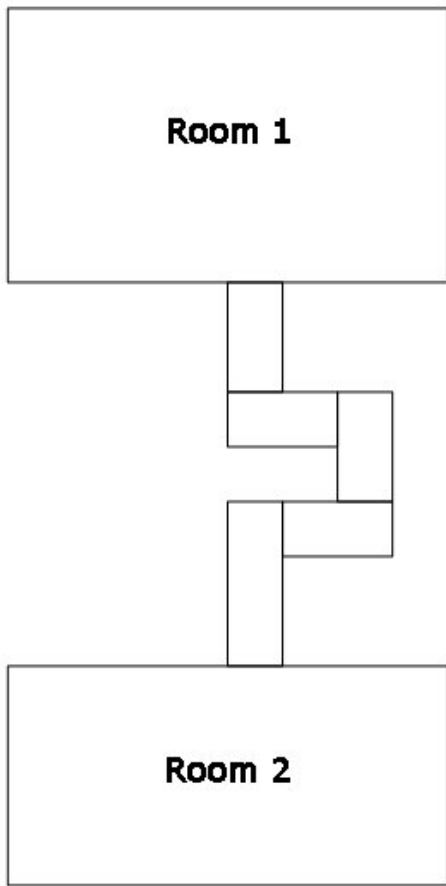
Suppose an area has been created that connects two rooms, as shown here. I'm sure you've often seen examples where connecting corridors or passageways go through seemingly pointless 90 degree bends, when a straight corridor would have made more sense. Monte Cassino is one.

The map author has done this to reduce the amount of work ET has to do when drawing the screen for a player.

A player in room 1 can't see anything in room 2 and vice versa. Indeed a player can't stand anywhere and see both rooms at once. This is how it seems to the player and how the map author intended.

But this is only true when the brushes are **structural**. If the brushes had been **detail** although it would all look the same to the player, ET would actually draw everything in the direction the player faces, regardless of all the intervening walls. The player would still see the same though, because the nearest wall would get drawn last and would obscure all the stuff behind it.






So how do you choose when to use structural and when to use detail? It would seem that if you made everything out of structural brushes there could not possibly be any wasteful drawing by ET and you'd always get the best FPS....

Well yes. And no. The problem arises when some of the brushes you make are small and irregular, say for crates that are scattered about at odd angles, likewise chairs, wall charts, steps of a flight of stairs, window ledges, cables, ducts, wonky fencing, planks stacked at different angles, etc etc. You would make all these **detail** brushes, because they reasonably wouldn't affect overall line of sight - if you're standing on a step, you'll see about the same when you stand on the next step - but more importantly if they were made of **structural** brushes Radiant would break up the surrounding volume into vast numbers of portals that would make your compile take forever and would clog up ET when it ran with vast portal tables telling it the 450,000 portals you can see from one step, and the 450,000 very similar portals you can see from the next step and so on.

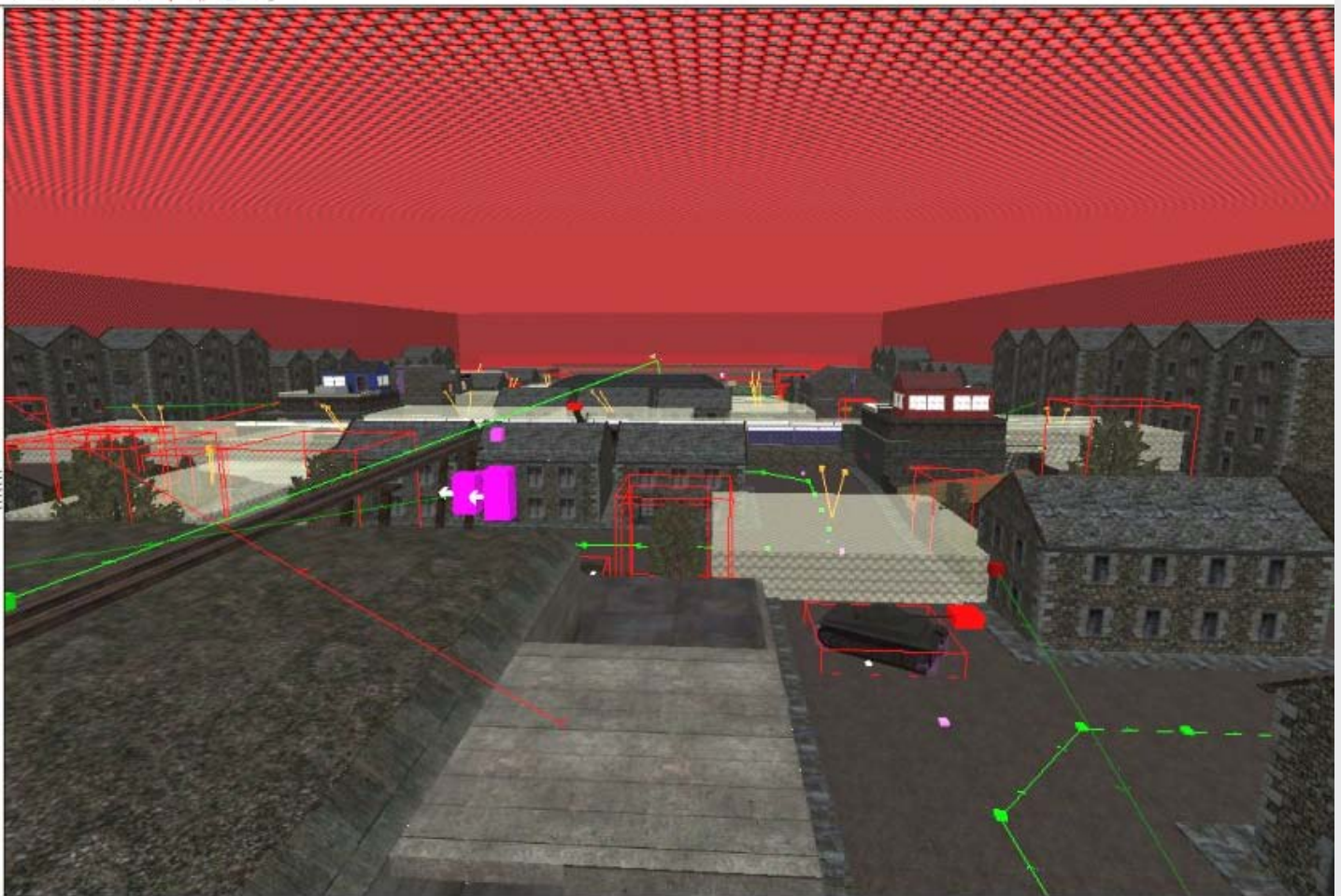
#### In summary:

Use **structural** brushes to define buildings and principal areas/rooms. Keep it in mind to try to separate main areas in this way so that ET won't have to draw more than it has to. For example, in 110 Factory all of the outside is detail brushwork, while the bunker and factory walls are structural. A player standing on the submarine will make ET draw everything in the direction he is looking at, but not the insides of the factory, nor the rear of the bunker (it will have to draw the inside front of the bunker because of the window opening).

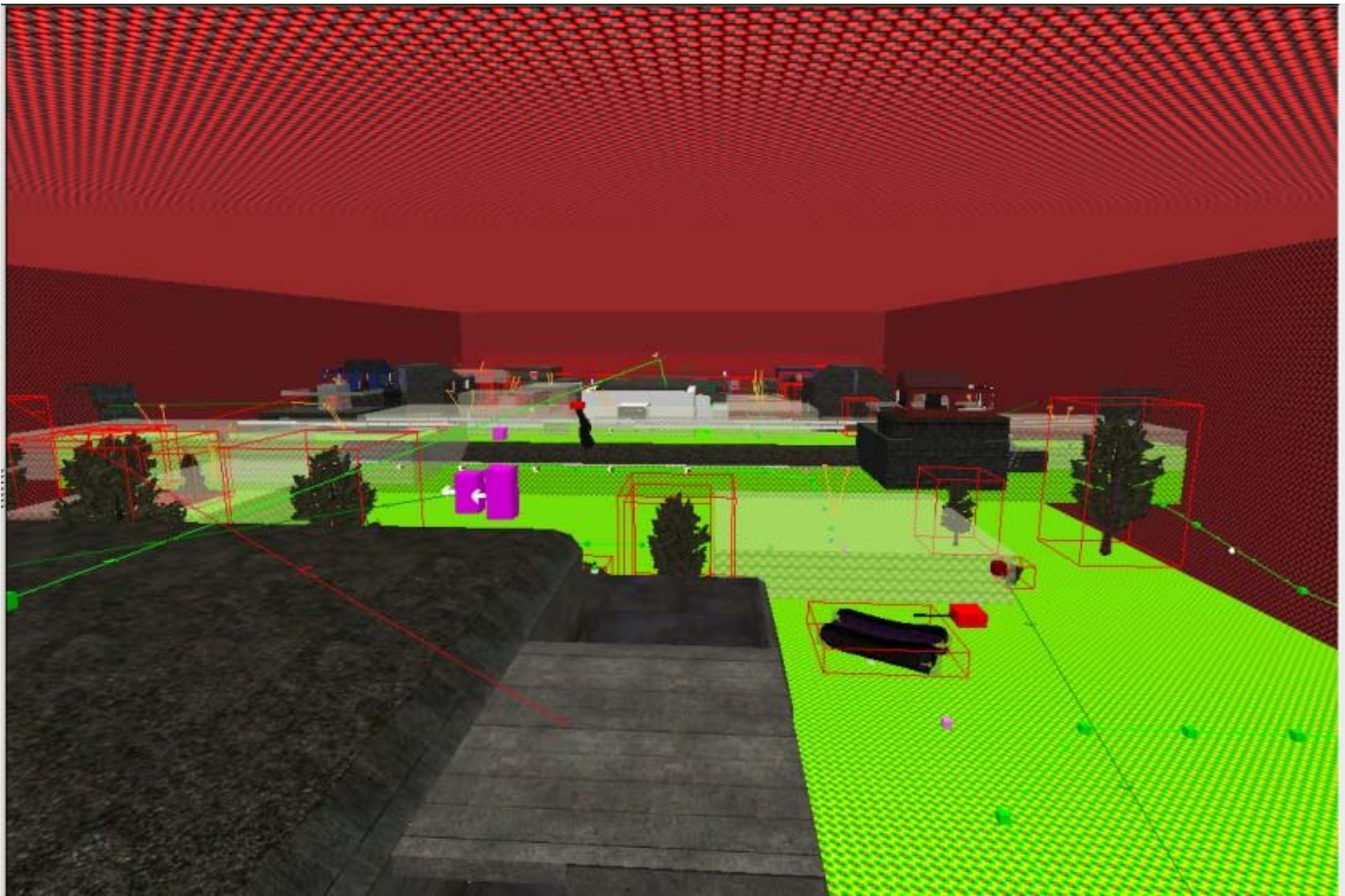
Use **detail** brushes to make small details, intervening walls where it really adds little to the display overhead, and any surface like steps, ramps, walkways etc where the view from one to the next differs very little. Also you really must make terrain with detail brushes when using GtkGenSurf, which we'll cover later.

 You will often forget to make brushes into detail as you make your map. A very handy tip is to press ctrl+D from time to time to toggle the filtering of detailed brushes. When you press it first, all your detail brushes will disappear. It can then be very obvious that you've got some structural brushes that ought to be detail. As you go around making them detail, they disappear too. When you're satisfied, ctrl+D to get the detail brushes displayed again.

These pictures are of a Radiant view of 2Tanks. The first shows all the brushes.



Then we press ctrl+D and filter out all the detail brushes.

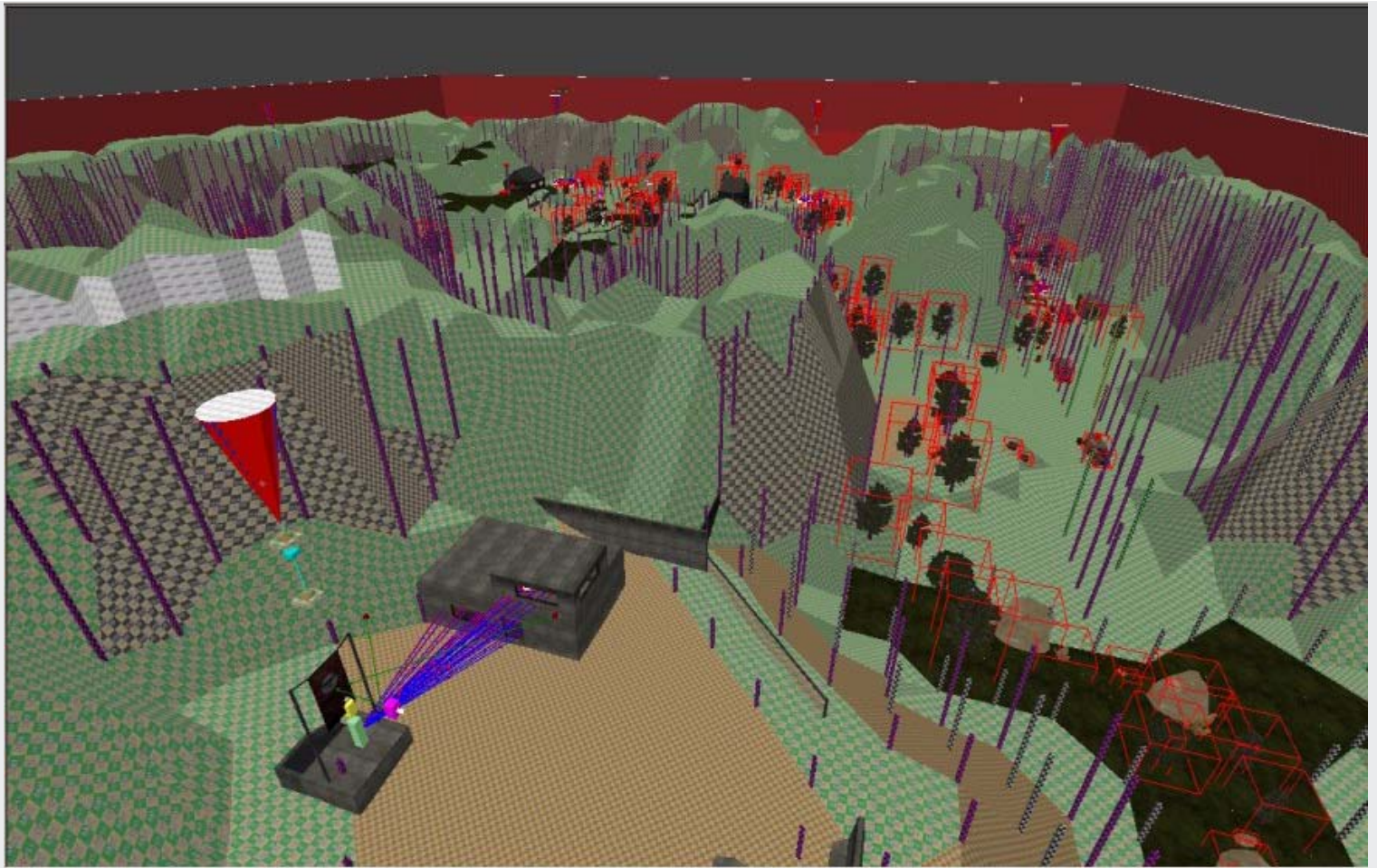


You'll notice virtually all of the roads, terrain and buildings disappear. They are all detail. The main areas the players can get inside are made of structural brushes, so for example ET won't have to draw the inside of the Fuel Dump nor the inside of the V-1 Base unless the player is in there.

There was no point in making roads and dummy buildings structural, because there are too many ways for the player to see over the top of them anyway, so they wouldn't really act as efficient blocks to line of sight. By keeping the dummy buildings simple the overhead in displaying them is small, and it cuts down hugely on unnecessary portals (and keeps my compile time sensible!).

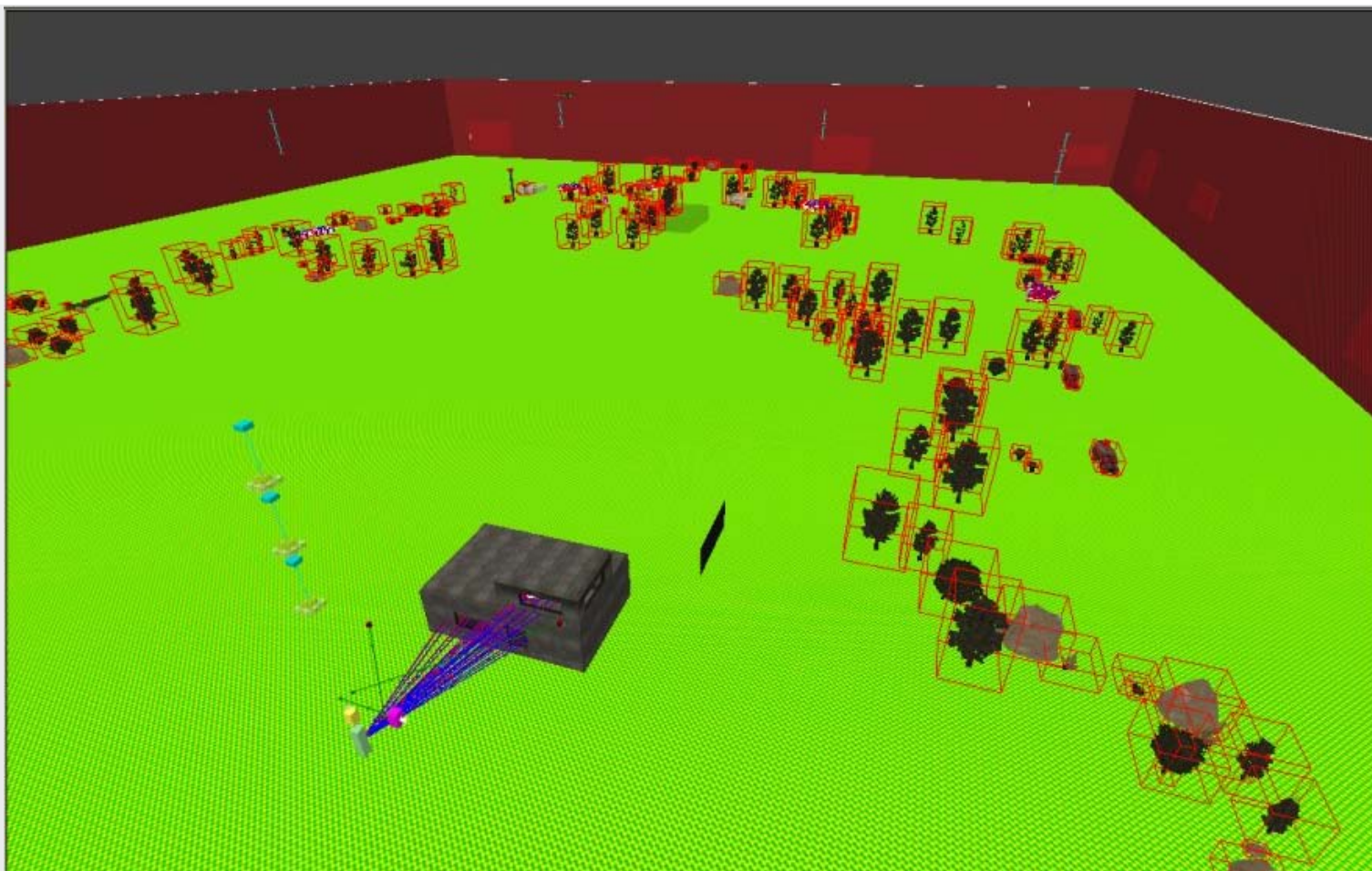
With 6Flags the difference is even more startling. Here's the normal view:





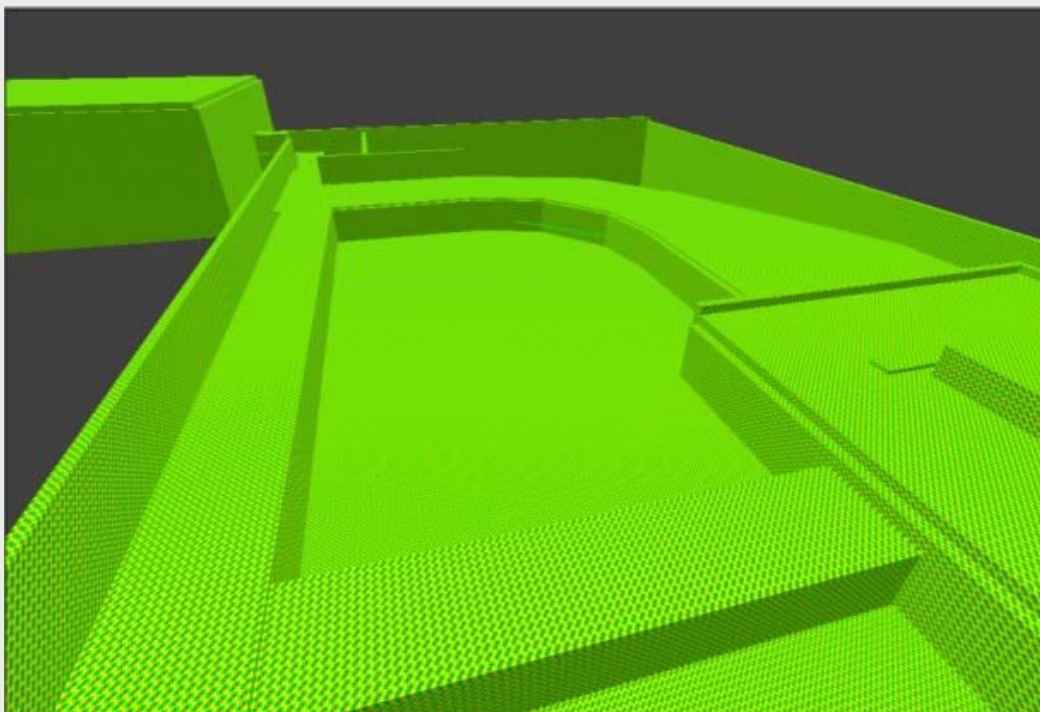
And now with the detail brushes hidden:





The open, rolling terrain, on which a player could get quite high, made it impractical to use structural brushes. It also meant I couldn't go overboard on including too many fancy buildings etc as they would start to be a drain on the FPS.

Finally here's an "outside" view of TankBuster, showing how structural caulk brushes surround the overall environment.



[Next lesson](#)



# ET Mapping Tutorial

## Lesson 19

### Topics

#### Health and ammo cabinets

[Health and ammo cabinets](#)

[Back to main menu](#)

#### Health and ammo cabinets

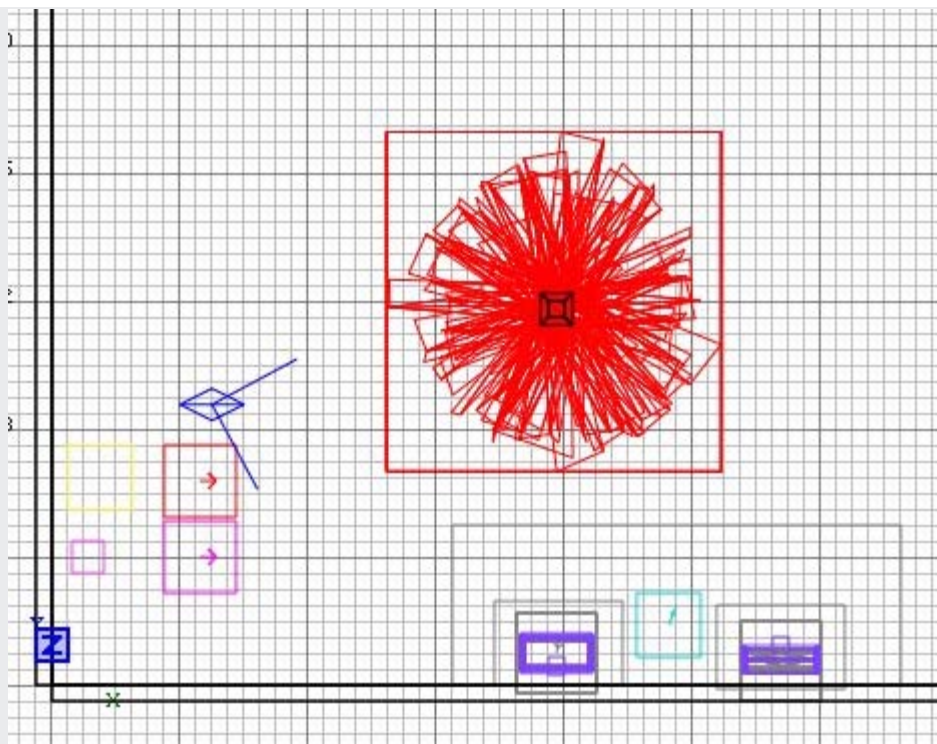
[\[Top\]](#)

This is another thing that you don't make by hand. [This one that I have zipped](#) was originally the health & ammo cabinets near the Allied start on Goldrush. I put the cabinets in line with each other instead of at right angles, put them at the same height as each other and created a prefab.

Download it, unzip the map and put it in your prefabs folder.

Run Radiant, open your map and import the prefab. It will appear near the 0,0,0 co-ordinate.

Drag it into place as shown.



That's it - no scripting needed. When a player stands near them they'll dish out health/ammo. The location will also appear on the command map.

If you want more than one set of cabinets, you can duplicate the whole thing but you will need to make some name changes: use shift+alt+click on each item in turn and for each one, press N and change "first" to "second" for any key/value in which it appears, then Hide the brush and do the next.



If you need to rotate a set of cabinets, or a command post, it's quite a lot of aggro. Better is to move the wall or find somewhere else to put it! But if you have to rotate the cabinets, rotate the set by multiples of 90 degrees to keep it simpler. The nuisance is that you don't change the orientation of models (which the cabinets do have) by rotating, so while the triggers and clips will rotate, the models don't :(

You must instead select each model and press N and enter its new angle. Save your work before attempting rotation on model/brush combinations! Especially true for trucks and tanks, and most of all for command posts.

You can control how much and how quickly the cabinets dispense things.

Shift+click on the big trigger and Hide it.

Shift+click on the ammo trigger (it's on the right as you look at them in 3D).

Press N. You can see the ammorate and ammototal. Healrate and Healtotal apply to the other trigger. These ones have the standard expected values so you normally won't need to change them. You can make other things dish out health/ammo by using these triggers, like the water in the dog kennel in 2tanks.

Save, compile and test your cabinet.

[Next lesson](#)





# ET Mapping Tutorial

## Lesson 20

### Topics

#### Command Posts

[Command Posts](#)

[Back to main menu](#)

### Command Posts

[\[Top\]](#)

Of all the constructibles, destructibles, health cabinets and whatnot, the one thing you really don't want to make from scratch is a Command Post.

Ok you can make a constructible wall by hand if you like, it's a little long-winded, but quite easily achievable without resorting to "here's one I made earlier."

But a Command Post has a number of fiddly, angled models and a variety of invisible brushes, and it really isn't realistic to make one by hand.

So here are the [prefabs](#) you need. :)

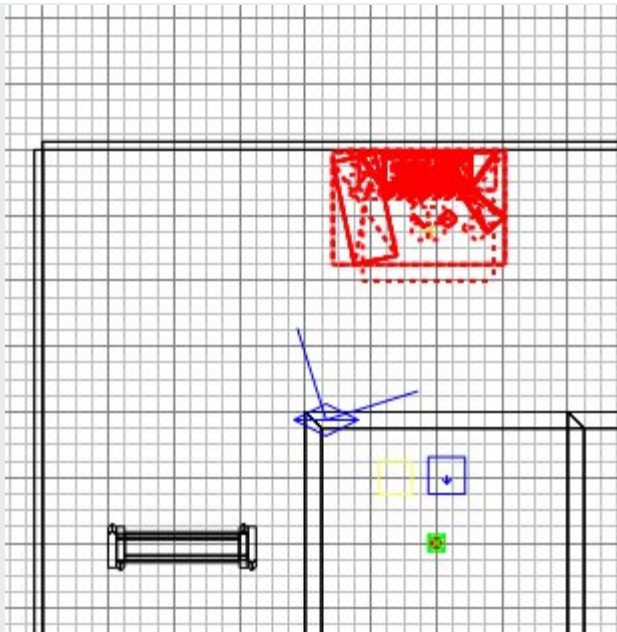
Unzip them and stick all 6 files into your prefabs folder. There are 3 pairs of .map and .script files, being a pair each for the Allies, Axis and Neutral Command Posts.

Run Radiant and open your map.

We'll put a Command Post against the north wall. The command post model is so complicated you really want to avoid having to change its angle. So I always find a suitable wall against which I can stick a command post without having to turn it around. You have been warned :)

Suppose we want to put the Axis CP there.

Go to File/Import... and navigate to prefab\_axis\_cp and import it. The model will appear below/left of (0,0,0). Drag it to the north wall. It happens to have the correct Z co-ord value of zero, because I tend to save prefabs with a Z co-ord of 0, and I try to keep much of my map floor surface around Z 0 for this reason. It makes building things easier. (I also try to keep major height differences to multiples of 128 or 256, because again it makes something easier: texturing. It cuts down on having to re-align a texture.)



Press ESC. Compile the map but don't yet go to ET to test it.

Now edit the `prefab_axis_cp.script`, and copy and paste the lot into the bottom of your `tutorial.script` file. (As your script file grows, try to position new procedures in the right place alphabetically, ie put "allies" procedures before "axis", and arrange all subsequent procedures by alphabetical procedure name. This makes it easier to find things later.)

Don't worry about trying to interpret the contents of the Command Post procedure, it works just fine and you don't need to know how the script works (until later in the tutorial.)

Save and exit from the `.script` file.

Run ET. We are not quite finished. We have to add the CP speaker. `/devmap` the map. Be an Axis Engineer.

Go and stand directly in front of the unmade CP. It probably seems a bit dark. CPs generally have extra light shone on them by an explicit light entity, as for some reason they otherwise show up dark. I don't know why. Don't worry about it now.

Bring up the console and type:

```
\editspeakers
\dumpspeaker
\modifyspeaker
```

Then exit the console. Make sure the speaker is just in front of the CP.



Enter these values into the boxes:

**Noise:** sound/world/radio\_axis.wav The "/" must not be a "\" this time!

**Targetname:** speaker\_axis\_cp

**Looped:** on Make sure you hear the axis CP sound. If not, you have mis-spelt the **noise**. Go back and get it right then proceed from here.

**Looped:** off

**Broadcast:** nopvs

**Volume:** 32

**Range:** 1250

And click OK.

Go to the console, up arrow 3 times and press return to cancel speaker editing mode.

Enter **\map\_restart**

Come out of the console.

Go to your unmade CP; there should be no CP noise. Make the CP. You should now get the noise and a working CP.

Make yourself an allied cov ops and prove you can blow up the CP and that everything looks normal.

**You are all done!**

The allied prefab works just the same way, with these changed values:

- **Noise:** sound/world/radio\_allies.wav
- **Targetname:** speaker\_allied\_cp

For the neutral CP you'll need 2 speakers, one for Axis and one for Allies. Put them beside each other.

Easy!!!!!! :)

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 21

### Topics

#### Curved walls and arches

[Curved walls](#)

[Making an arch](#)

[Back to main menu](#)

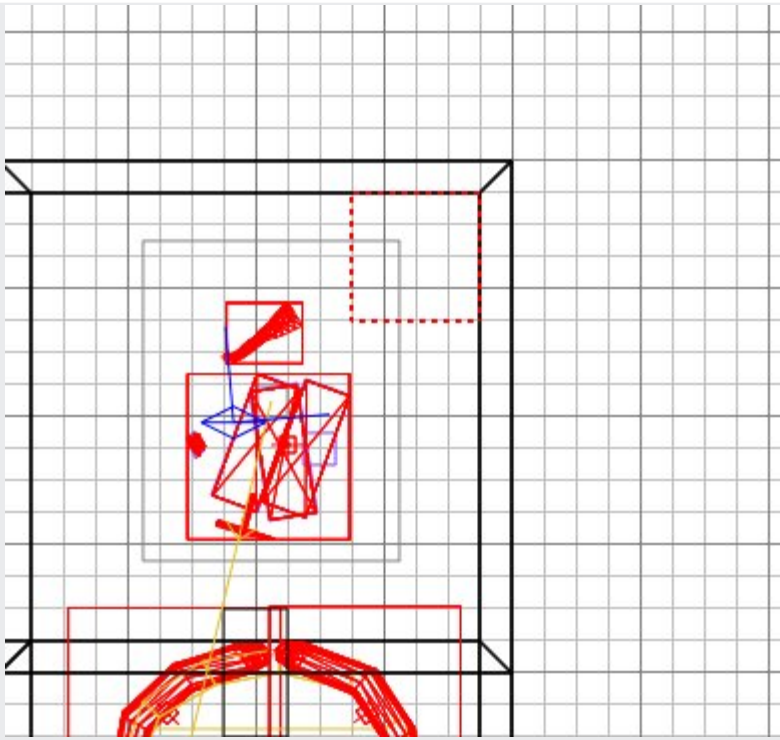
### Curved walls

[\[Top\]](#)

We can demonstrate how to use a simple curved surface by making one of the room corners rounded instead of right-angled. You'll be able to apply this technique to making bulges in a wall, or convex shapes like arcs of a column.

Run Radiant and open the map.

Select grid size **5 and** draw a brush as shown.

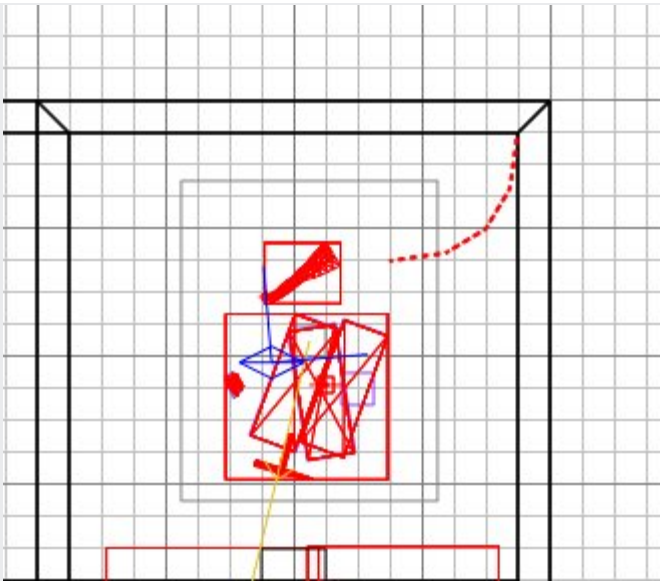


Give yourself the 3D view that will show you what's going on in the corner.

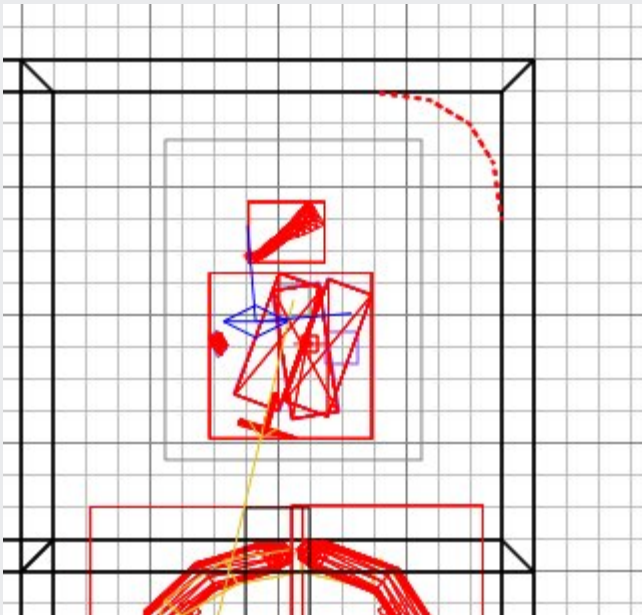


Depending on what other brush manipulations you've been doing, you may or may not see the brush in front of you. Let's assume you don't.

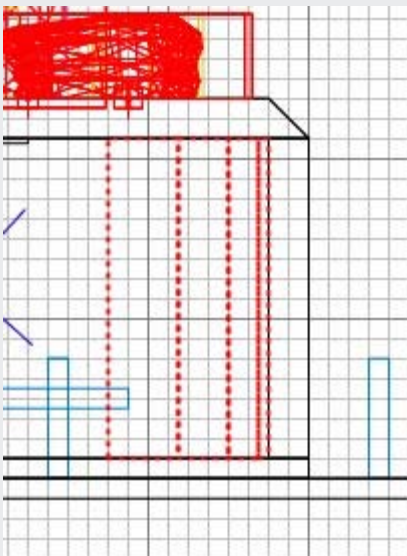
Click Curve\Bevel and see what you get in the 2D view. The brush has been converted into a curve called a patch. You still select patches just like a brush, ie shift+click.



Do a Z-axis rotate 3 times so that the curved line fills the gap between the walls.

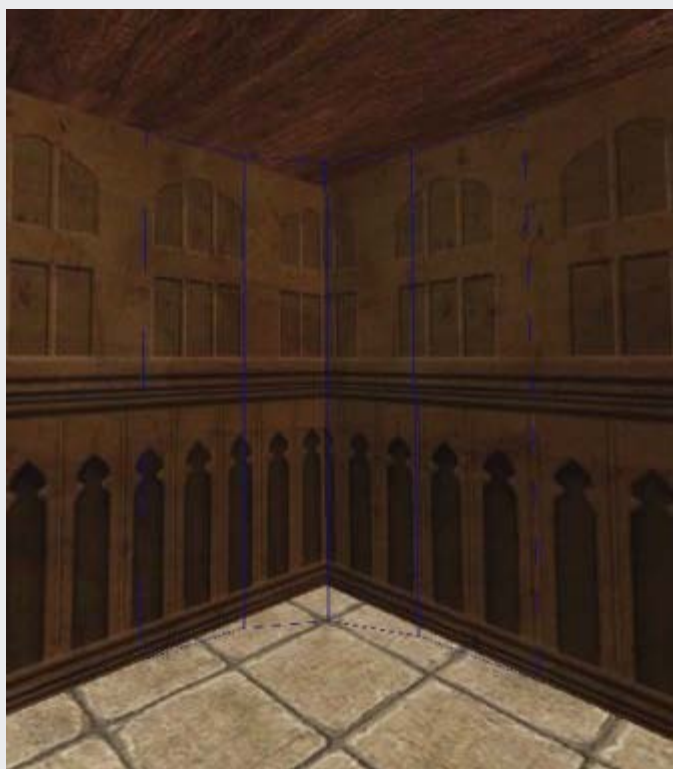


Ctrl+tab so you can position the curved section to fill up the gap from floor to ceiling. You'll need to change to grid **4** to make the fit.





In the 3D view you'll now see some faint lines of your curved surface, but no texture, so you can see through the new brush.



Click Curve\Matrix\Invert to flip the drawn face to this side.



We will give the new curve the matching wooden texture of the rest of the wall.

Press **shift**+S to bring up the Patch Properties window.



Shift+S for patches (the name given to this type of modified brush) is like S for the faces of regular brushes. **BUT**: if you accidentally press shift+S when you meant "S", or "S" when you meant shift+S, you will very likely crash your PC.

Scroll your textures window until you see the wood\_test texture and click it. Ok the texture gets applied but

it looks terrible.

Click **Natural**. This will look better, but it may be upside down. If so, click the up arrow **Rotate Step** four times.

Press ESC. It should now look like this:



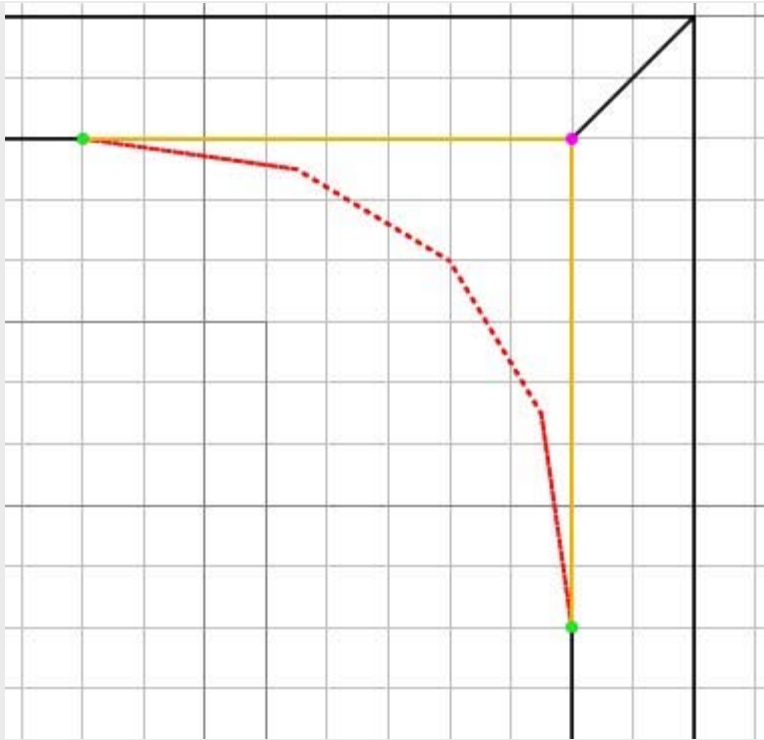
With a texture that has no discernible pattern, like concrete or metal, you won't see the join. With this panel effect, we do see the join. It probably won't be noticed by anyone: did you notice the bad join in any of the inner corners? They are probably there but not noticed.

We can either cover the join with some cosmetic furniture or trunking or similar; or we can adjust the positioning/stretch of the texture to try to get a better fit.

Select all the inner room wall faces, other than the curved bit, but including the faces around the window and door, and click the wood\_test texture again. This makes the texture run properly from one face to the next, but it also highlights what happens when brushes don't sit squarely at Z height intervals of 128. Because our wall brushes are up a notch, the textures are vertically wrong by a notch.

We correct that by pressing **S** and the up arrow V Shift 4 times and then click Done. Press ESC. So alright, the wall textures are all run smoothly, but the curved texture still doesn't quite fit.

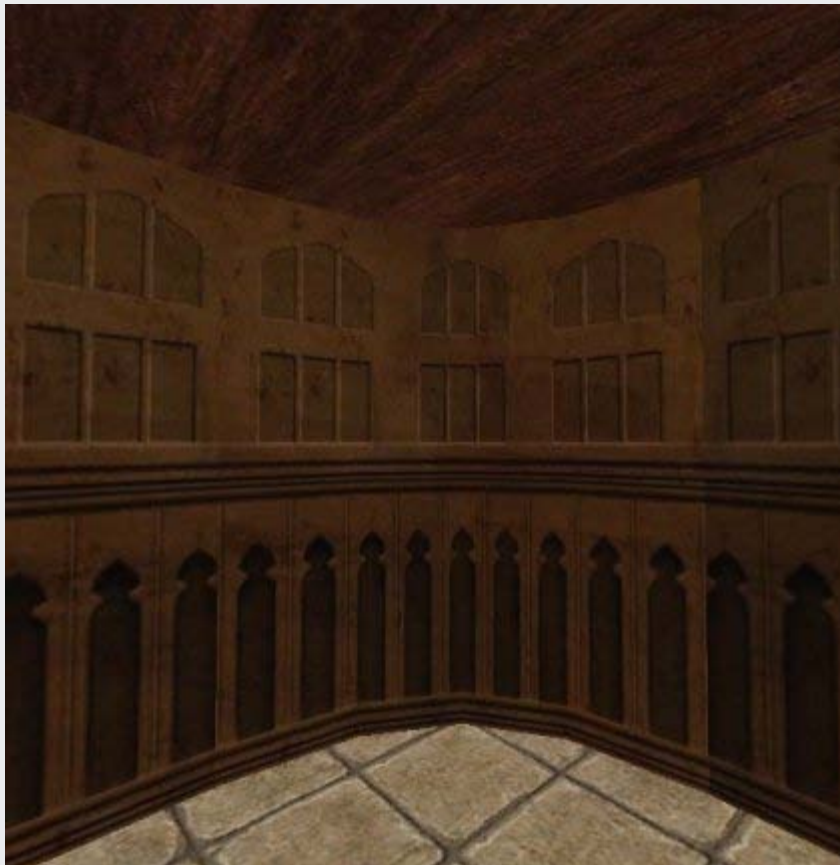
Select the curve patch, zoom in in 2D and press **V** (Vertex tool).



Click on the top left green dot and drag one notch to the left.

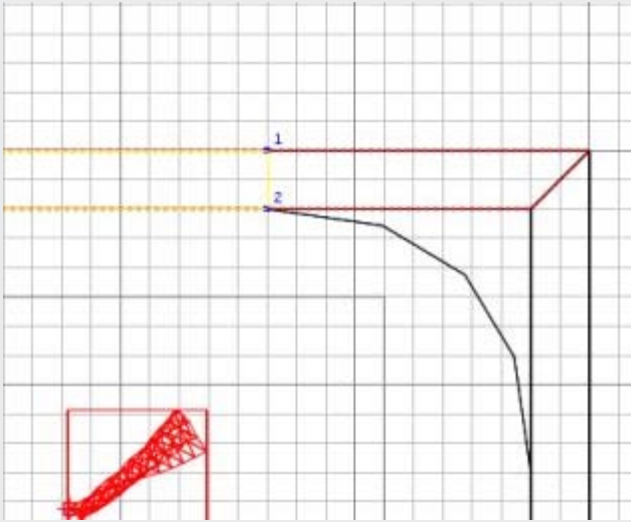
Click on the bottom right green dot and drag one notch downwards.

Press ESC twice. We've now got the curve probably as good as we're going to get it.

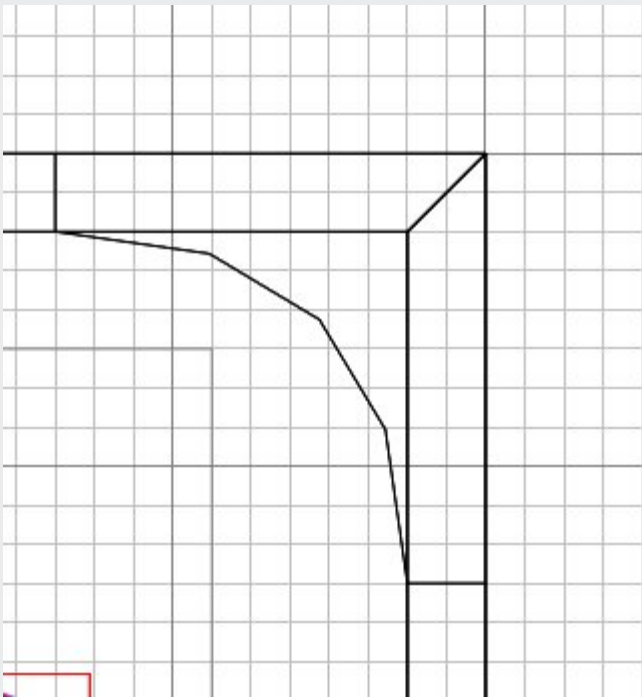


Finally we want to eliminate the wasteful face drawing that is behind our new patch.

Still in overhead 2D, select the north wall and cut it (X tool) where shown. Use Shift+Return to keep both parts.

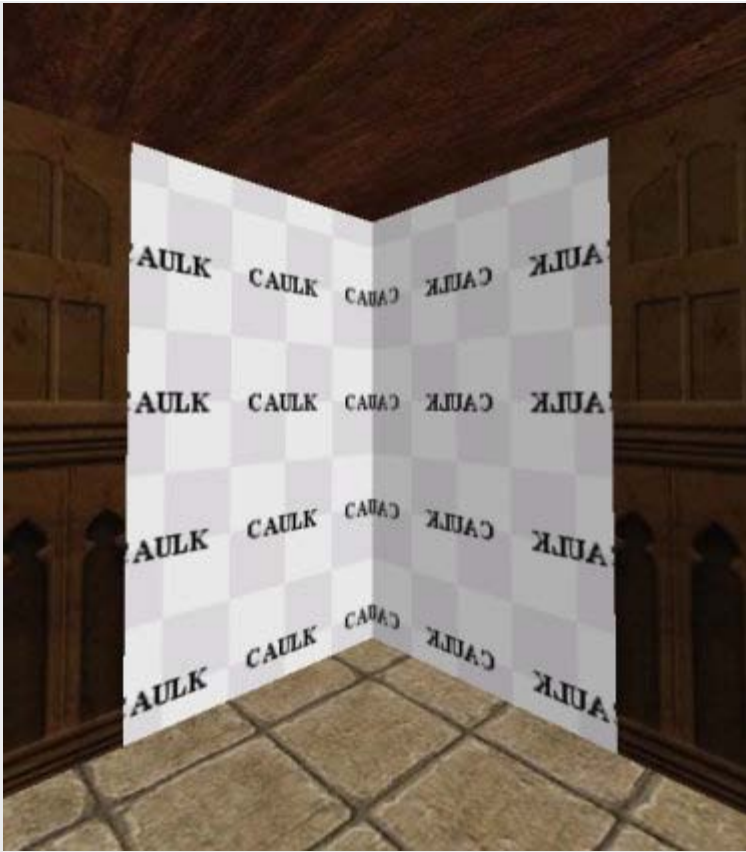


Press ESC twice and select the side wall, and cut it also where it meets the edge of the curve. Press ESC until everything is deselected.



Now select the patch and Hide it.

Select the wall faces that are obscured by the patch, and caulk them. Press ESC.



Then press shift+H to reveal the patch again.

Save, compile and go and see how it looks in ET.

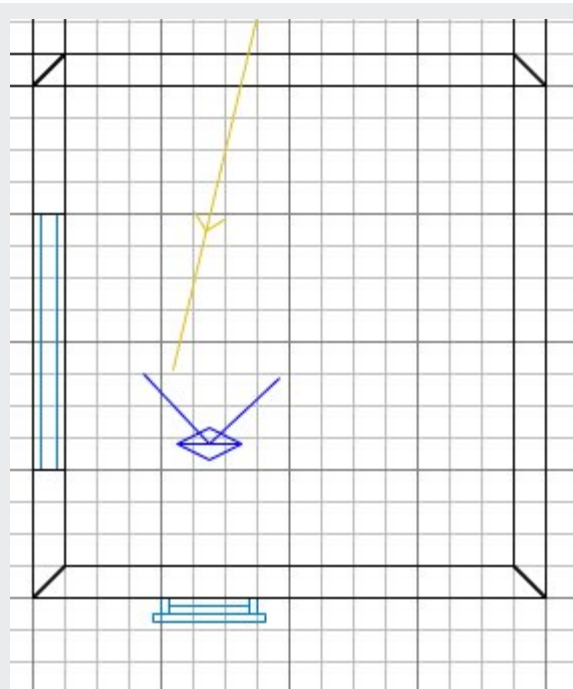
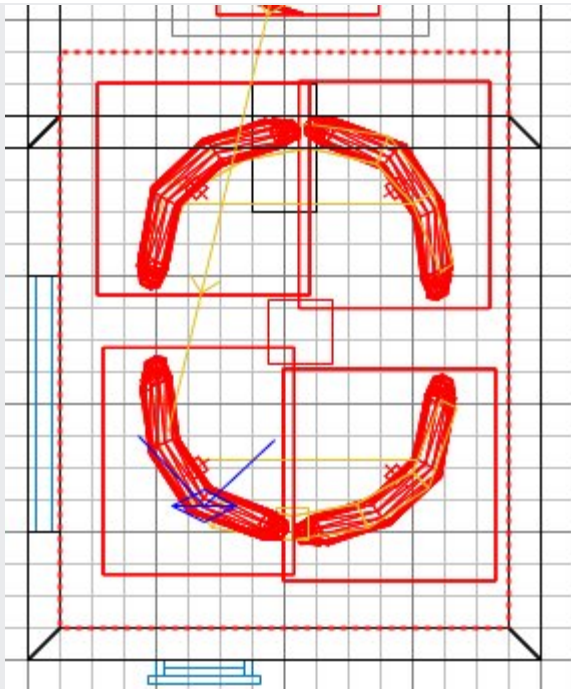
## Making an arch

[\[Top\]](#)

We'll add an archway inside the room. The same principle would apply to making archways and tunnels outside of course.

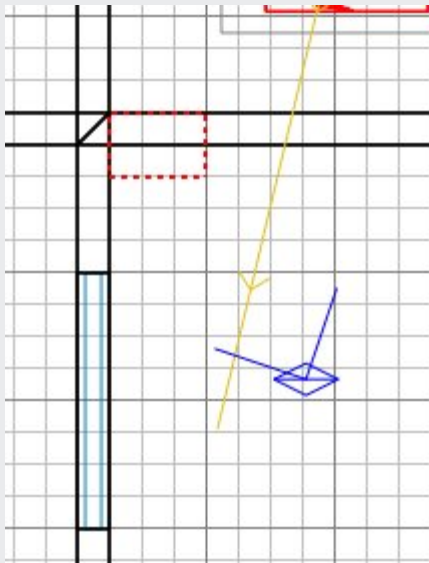
Draw a brush as shown, then Select Complete Tall and Hide them. This just makes it easier to see what we're doing in the room.





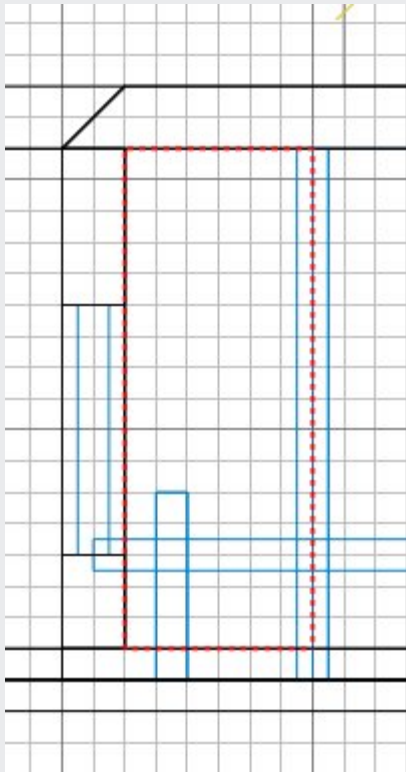
Let's put in a couple of uprights to be the arch supports.

Draw the brush as shown, caulk it and make it detail.

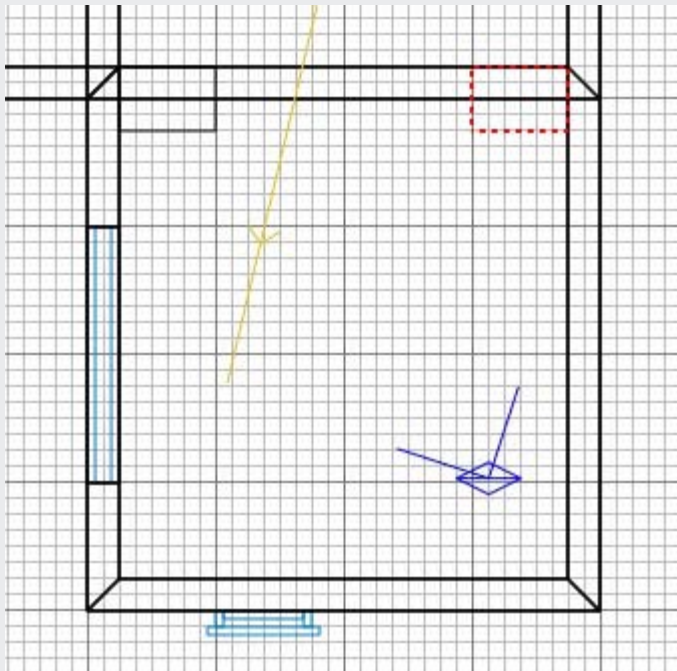


Get a side view and make sure it goes from floor to ceiling.





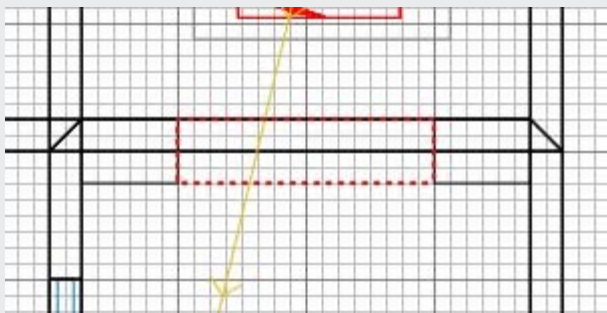
Select the 3 visible faces and give them the church\_c01dm texture. Press ESC. Get the 2D overhead view and then duplicate the brush, mirror it in the X-axis (x-axis flip button), and move it to the opposite wall.





Press ESC. Now we'll put in the connecting ceiling section prior to creating the archway.

Draw a connecting brush as shown and caulk it and make it detail.

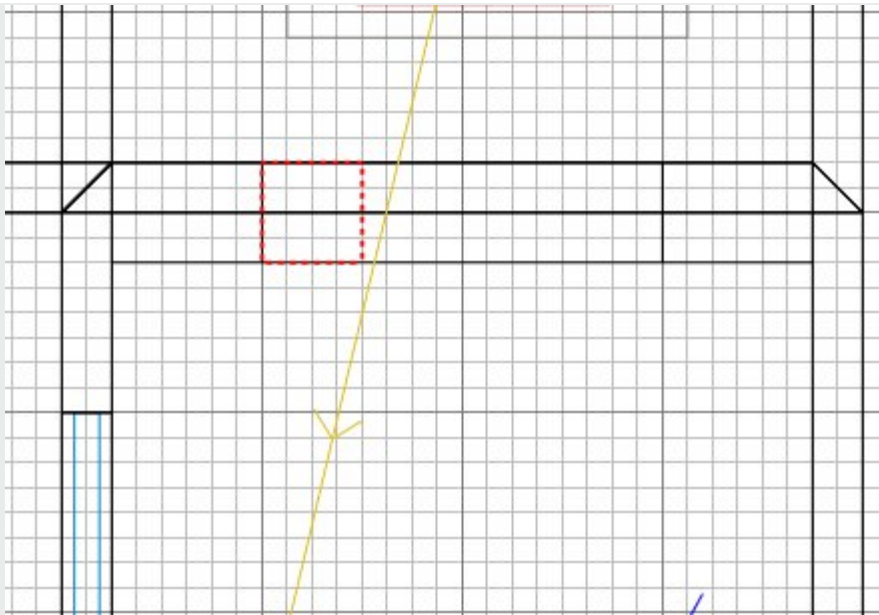


Get a side view and shrink the brush up to a narrow strip at the ceiling.



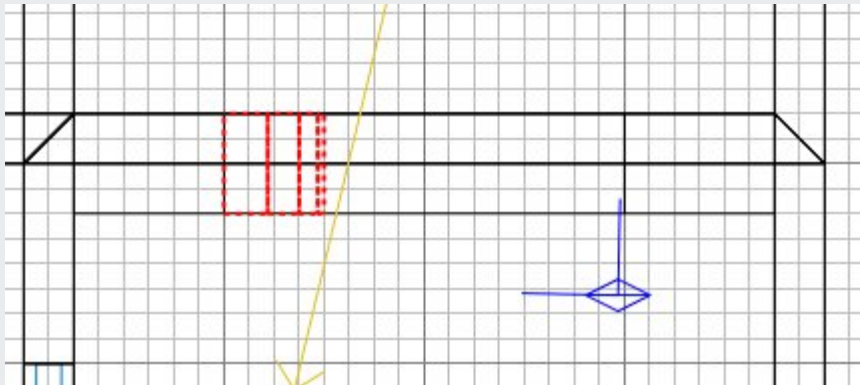
Apply the brick texture to the 3 visible faces of the new brush.

Overhead view again in 2D and draw a brush as shown.

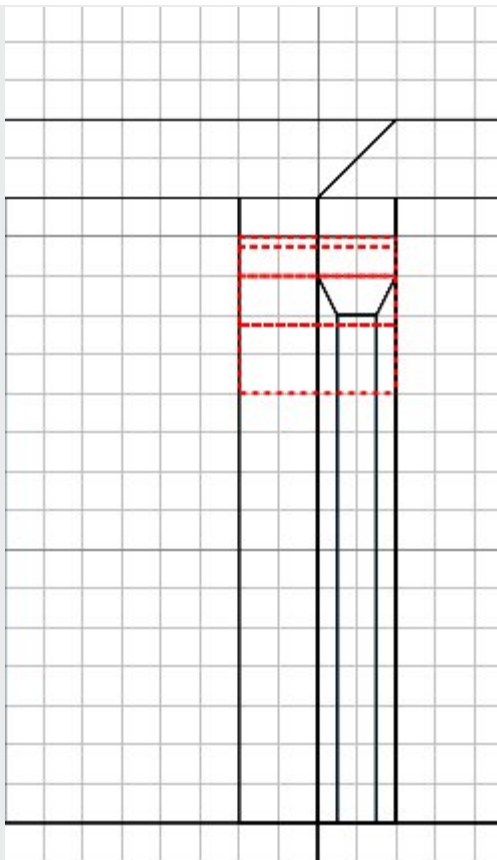


Curve/Bevel to turn it into a patch, then rotate it 90 degrees in the x-axis (x-axis rotate button).

Reize the brush until it's the same width as the columns.



It's the wrong way round for this side of the archway, so rotate it twice in the Z axis. Then ctrl+tab twice so we can move it vertically into place. It should be placed as shown here:



In the 3D view we can see the texture is on the inner face and we want it on the outer face:  
Curve\Matrix\Invert.

Now we can also see that the texture is oddly stretched. Press **shift**+S and click Natural, then rotate it 90 degrees by clicking the up arrow rotate step twice.

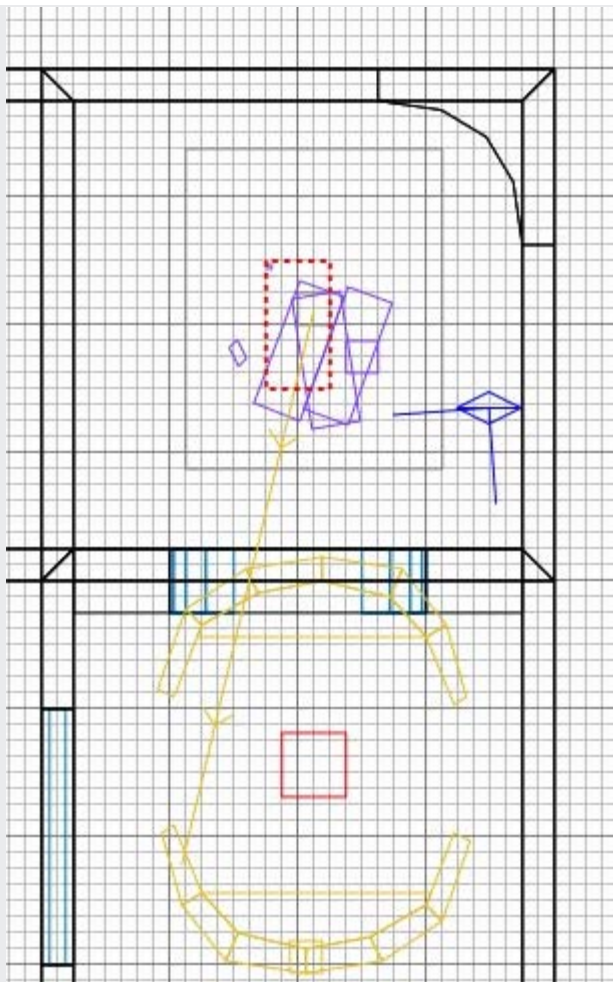
Now to fill in the gap: Curve\Cap Selection\Inverted Bevel\OK.

We end up with a func\_group consisting of the curve and two caps to plug the gap.

With the group still selected, get an overhead view, duplicate the group, mirror it in the x-axis and move it to the other side of the archway. Press ESC. It should now look like this:

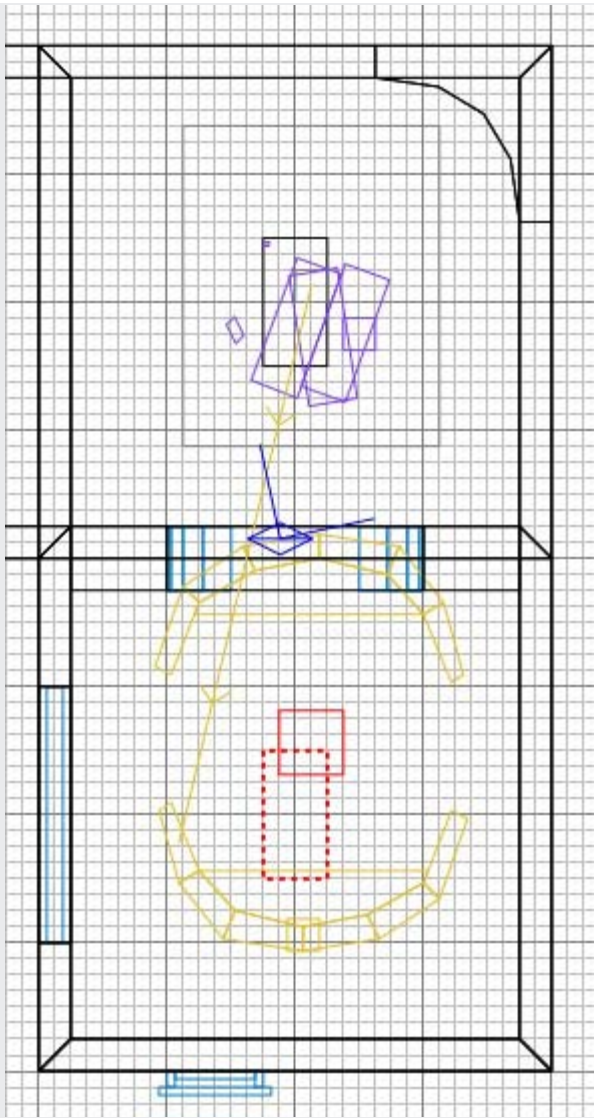


When you press shift+H to reveal the hidden brushes you will see the light is sticking through the arch wall, so move the light so that it isn't stuck in the wall. It is shown in the next picture with models filtered out so you can see where I moved it.



Duplicate the light and drag the new light down to the middle of the new room area.





Save, compile and test the look of the arch. Backup your work if you haven't, in case it all goes pear shaped later on.

Strictly speaking we should cut the upright columns and caulk the faces that are obscured by the arch patches. Let's take that as read and move on :)

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 22

### Topics

#### Cylinders, cones and curved roads

[Cylinders](#)

[Cones](#)

[Curved roads](#)

[Back to main menu](#)

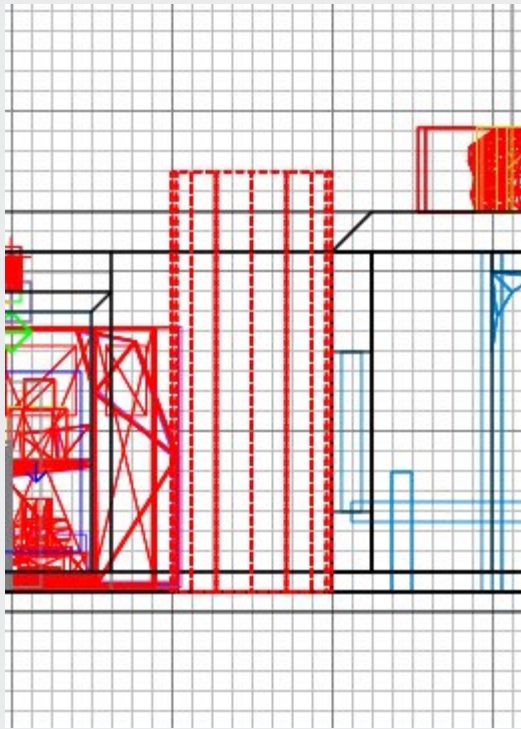
### Cylinders

[\[Top\]](#)

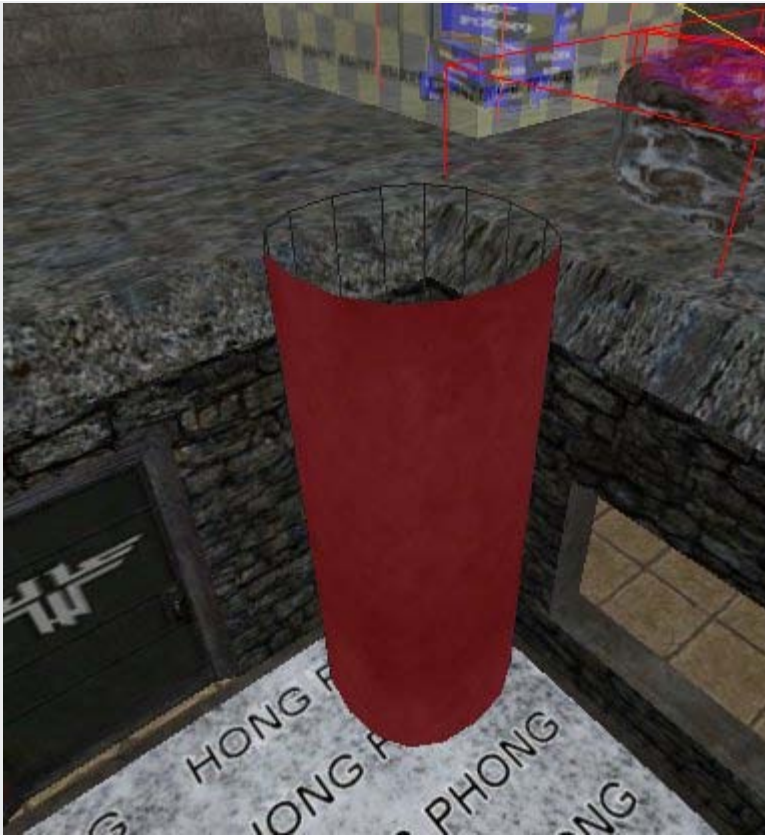
Run Radiant and open the map. Default grid scale 4 is ok. Draw a brush as shown.



Click Curve\Cylinder - the brush becomes a patch. Get a side view and stretch the cylinder upwards to make it into some sort of chimney-type pipe.

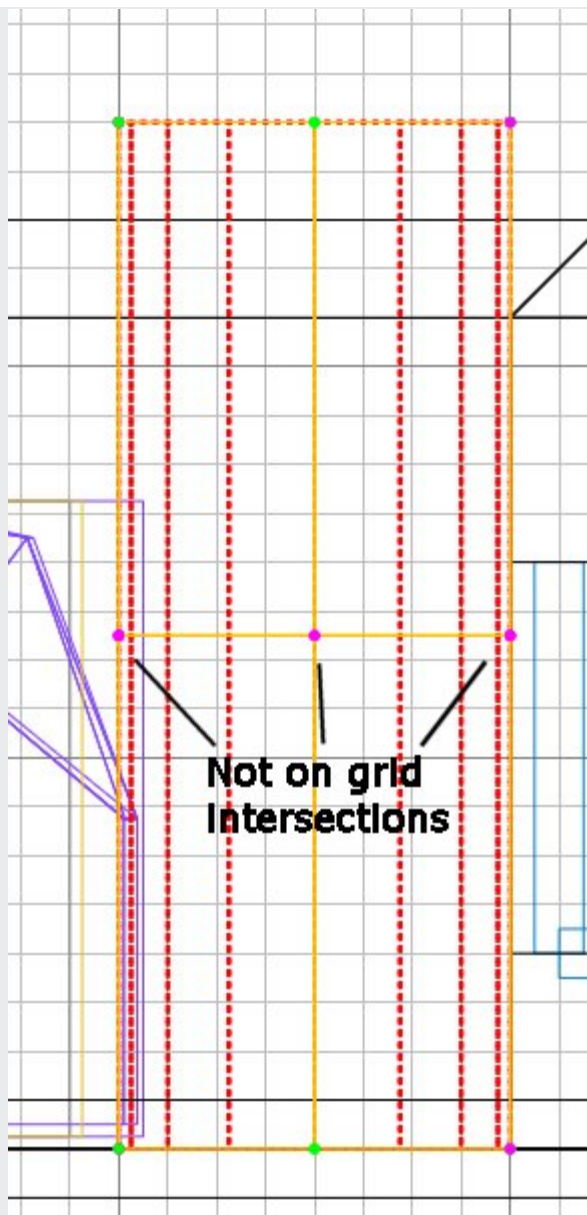


Click on the `metal_c07a` texture to texture the cylinder. The texture will probably look all stretched and horrid. Press **shift+S** and click **natural** and **done**.

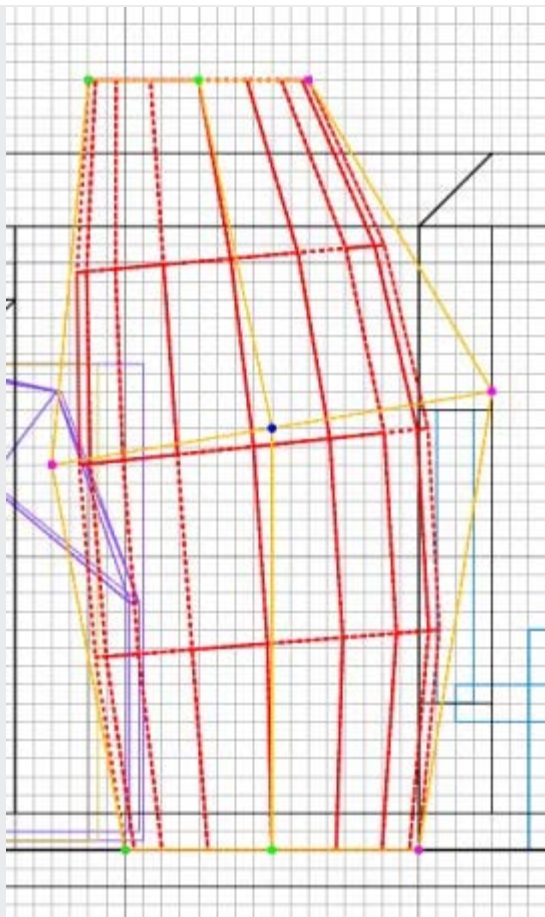


Note that only the outer surface of the cylinder is textured, the inner is effectively nodraw.

Let's warp the pipe. Press V and get a side 2D view. Filter the models (shift+M) to make the view clearer. Zoom in. Note that the middle height points may not fall on a grid intersection. If not, drop down the grid scale until they do.



In my example, grid **3** will do. Warp the tube in interesting ways by grabbing the illuminated vertices and dragging them, one at a time. The cylinder will warp as you drag the vertex point. In my example, I've dragged the top vertices to the left somewhat, and I've dragged the middle ones up, and outward to give a leaning, bulging cylinder.



Experiment, using different 2D views, to see what sort of weird shapes it is possible to create. You can also drag edges in 3D view, but this is trickier. Press V or ESC to cancel vertex mode when you've experimented enough, but leave the brush selected.

If you want to cap the open ends, and create a solid-looking cylinder, click Curve\cap selection. The top and bottom are neatly capped, and the three brushes are grouped for you automatically. We don't really want the bottom cap, so a quick way of getting rid of it is to change to grid scale 9, lift the group one notch so you can see under it in 3D, deselect the group and then select the bottom brush alone and delete (Backspace) it.

Then select the group (now only 2 brushes) with shift+alt+click and put it back down one notch.

If you want to see how your weird cylinder came out, save compile and test.





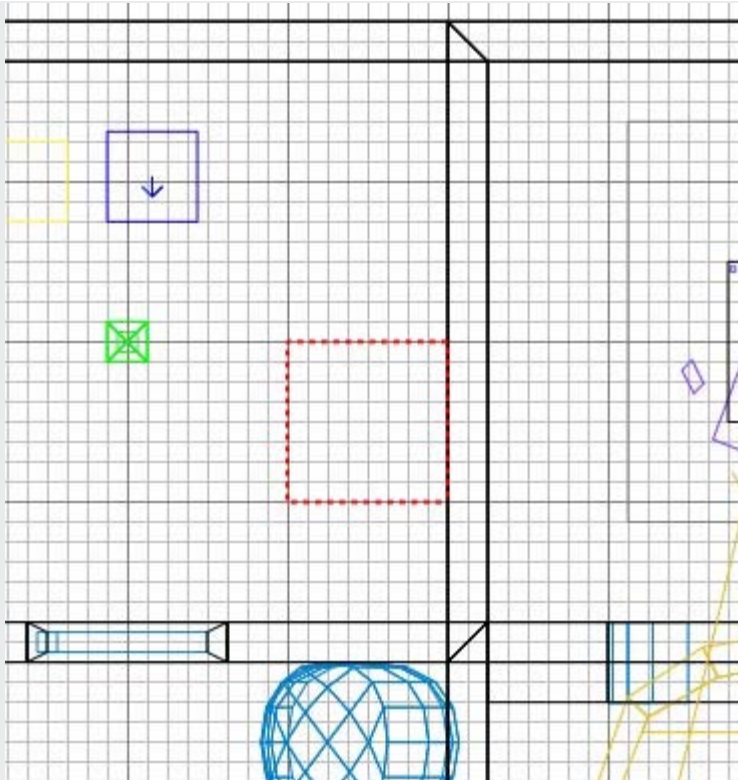
You can of course rotate these patches just like any other brush. But if you scale them, watch out! Scaling downward may result in some of the critical control vertices not aligning to exact grid co-ordinates, eg instead of a co-ord being (24, 56, 44) you might get (24.043, 56.324, 44.985), which tends to make things crash. :(

So if you *have* to scale a cylinder (as I did when I wanted to make my V-1 creation of multiple cylinders about 75% of its created size), you will need to examine all the component vertices in close up, with a grid scale of probably 1, and drag to intersections any vertices that fall in the gaps between them.

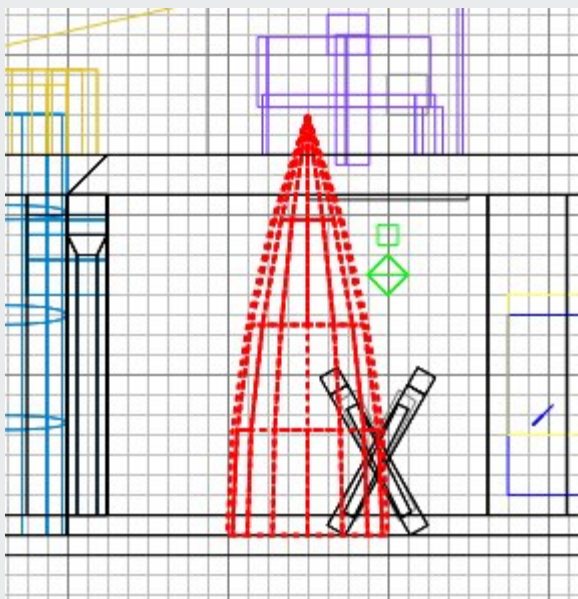
## Cones

[\[Top\]](#)

Lets put a conical bulge on the roof. Draw a brush as shown. It will probably be as tall as your previous cylinder - Radiant tries to guess your brush height requirement for a new brush based on what brush you last tinkered with.

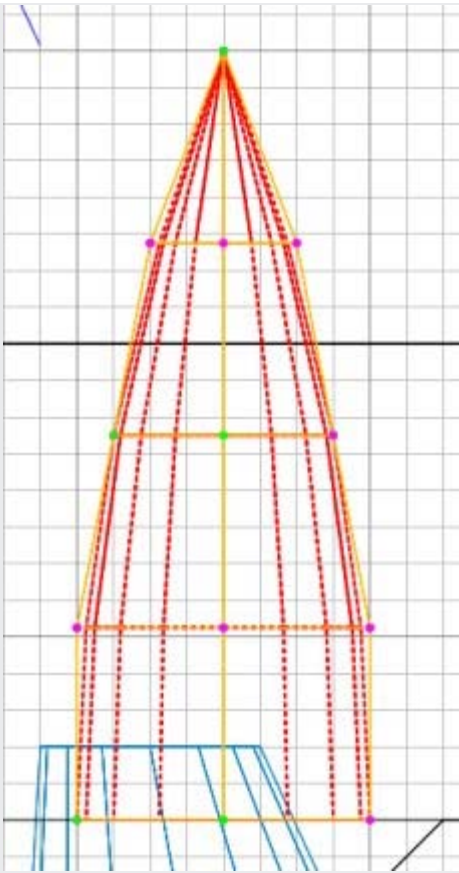


Click **Curve\cone**. The brush becomes a conical patch.

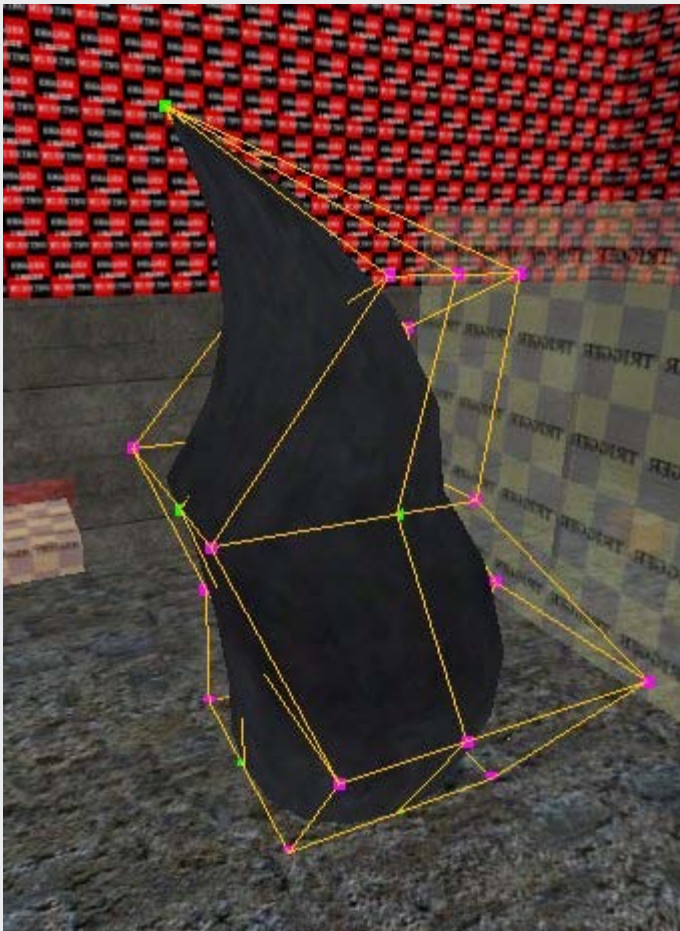


Move it up onto the roof.

My example looks like the nose of a V-2 rocket. But you can change the shape of a cone by using the **Vertex** tool, just like with a cylinder. Try dragging the vertices around to create a rounder or a fatter cylinder, whatever you need. If you need finer control of the shape, you can add more vertex points: click `Curve\Insert\Add 2 rows`.



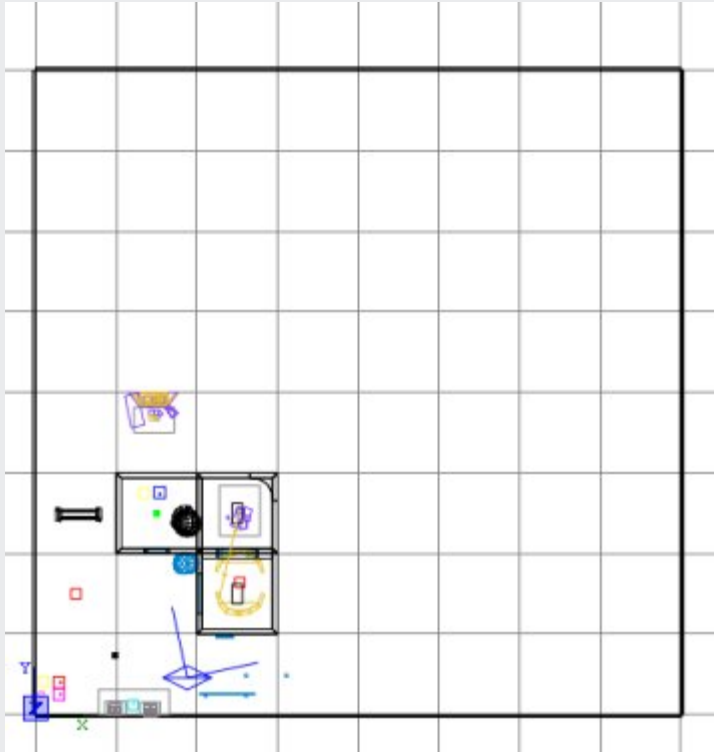
You can get some really weird shapes by dragging these vertices about. You can make some very interesting sculptures...



## Curved roads

[\[Top\]](#)

Make the tutorial map twice as big in area to give us some work space. Use grid scale 9 for this. Don't forget to set the mapcoordsmins etc to 2048 0 and 0 2048. Just enlarge each surrounding brush in turn, and hide them after resizing so you can easily get to the next brush.



Entities

worldspawn

----- KEYS -----

"target" the name of the enti

----- SPAWNFLAGS -----

(none)

----- NOTES -----

Compiler-only entity that spe

Should contain 1 or more patc

info\_null entity. The distanc

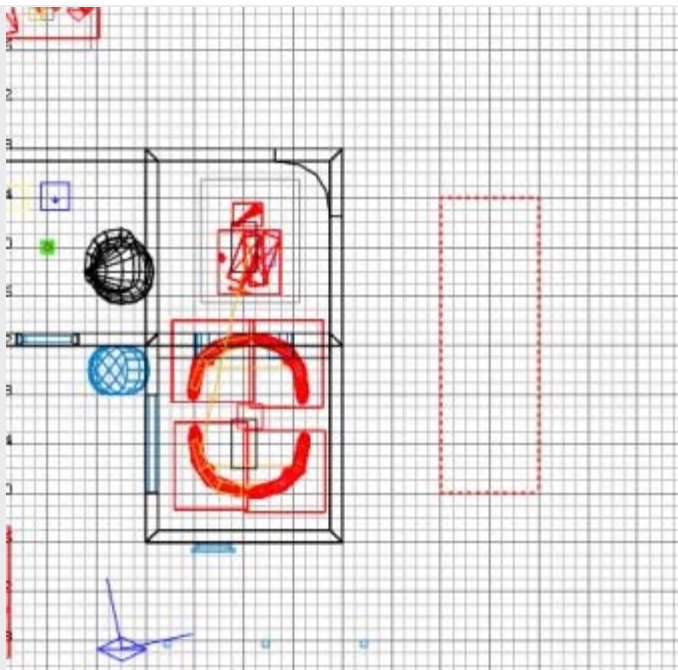
entity and the target is the

classname      worldspawn

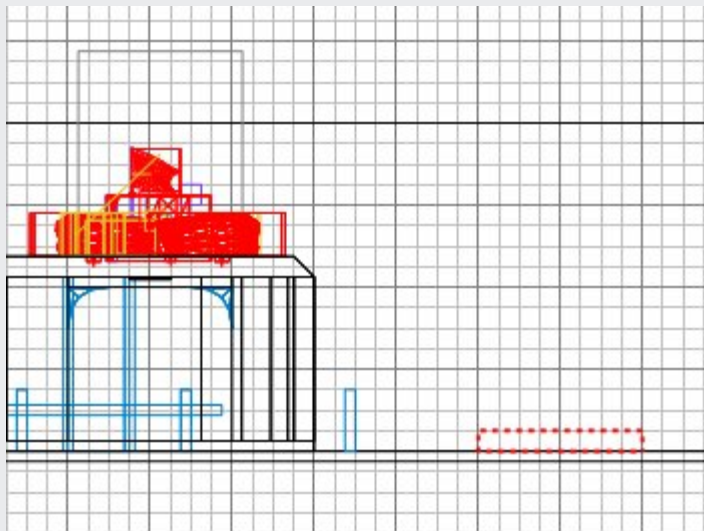
mapcoordsmins   0 2048

mapcoordsmaxs   2048 0

Out in the new space we will create a curving path. Hide the ceiling so it doesn't get in the way. Select grid scale 5 and zoom in a bit. Then create a brush as shown. You can show models again now if you like (shift+M).



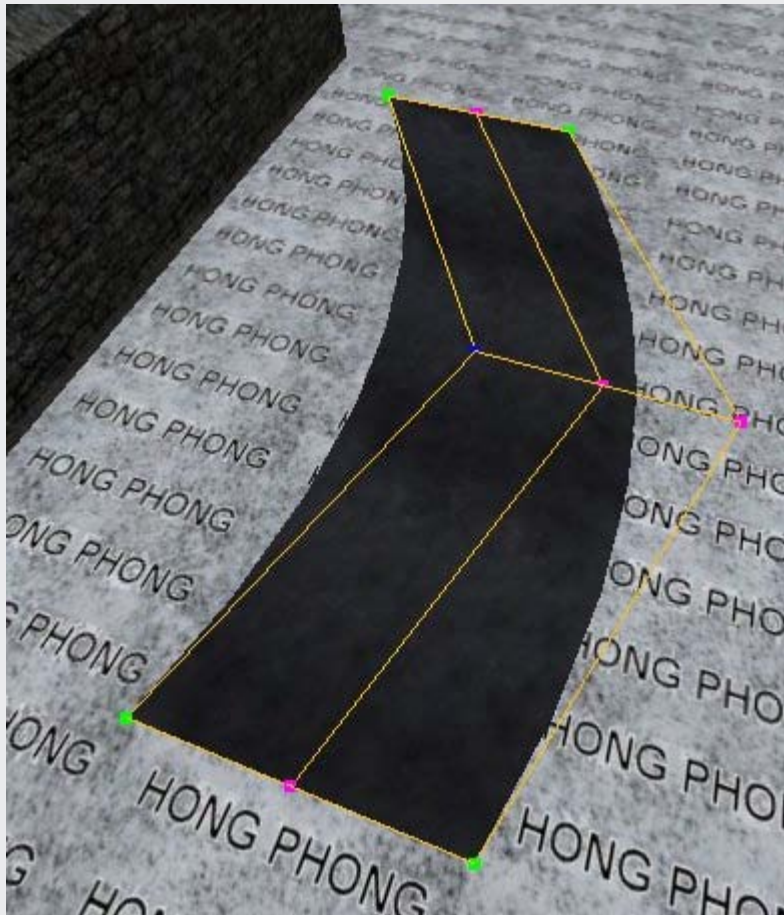
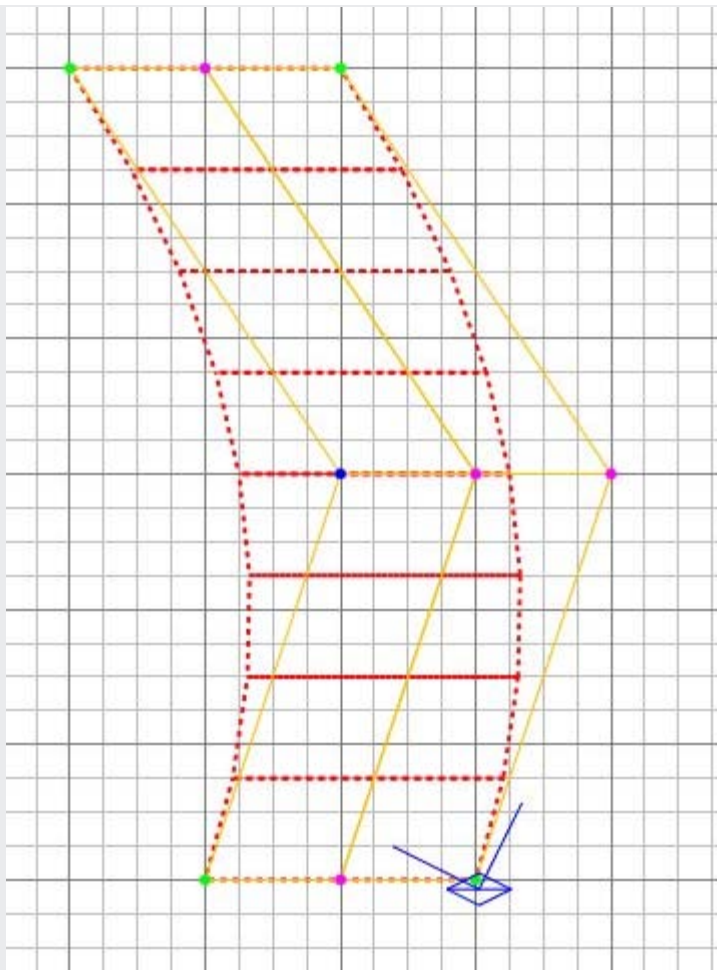
Move the brush down until it sits on the ground.



**Make sure you have the overhead 2D view.** Then click Curve\Simple Patch Mesh and click OK. This turns the brush into a patch of the same area as the brush, but with no depth.

Make the road curve by pressing V and dragging the vertices as wanted. I've made a gentle curve of uniform width for its length, by dragging all the vertices in the same row by the same amount, whether to the left or the right.







Press V to turn off the Vertex tool. Next we'll give the curve some depth.

Click Curve\Thicken... then as we want the curve to be 16 units deep (just so it touches the ground in this example), enter 16 and click OK. You get a chunky curve like a piece of scalextric track, defined as a group. We can delete the bottom face like we deleted the bottom cap of the cylinder, by lifting the group, deselecting it, selecting and deleting the bottom face, selecting the group again and putting it back in place.

You can create upward/downward slants in the curved path by lifting its vertices up/down before applying the Thicken.

You don't have to worry about Detail brushes for patches, ET treats them all as detail anyway.

Save, compile and check out your creations.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 23

### Topics

#### Making terrain using GtkGenSurf

[Concepts](#)

[Preparation](#)

[GtkGenSurf](#)

[Back to main menu](#)

### Concepts

[\[Top\]](#)

Sooner or later you're going to want to make some outdoor environments, and unless it's all in a concreted area, you're going to need to make hills, dunes, grassy paths, snowy mounds etc.

The usual way of making brushes to make an object doesn't really work here, so instead people use a third party tool. One is EasyGen, which I don't use, and another is GtkGenSurf (a plugin included within Radiant) which I do.

I haven't tried EasyGen so I can't comment on it. Also there's a new terrain-generating-kid on the block called FATE, which looks like it will be really good when finished. As I haven't used it I will stick to explaining how to do things in GtkGenSurf.

GtkGenSurf will generate an area of terrain using lots of triangular brushes, like the box Toblerone comes in, with one end face the required texture and all the rest as caulk. You tell GtkGenSurf the area dimensions, the required size of triangles, the textures to be used and the contours you want and hit the Go button: GtkGenSurf then produces the terrain mesh, already grouped and ready for you to slot into place.

The approach I take to making terrain is this:

- Identify the area in the map that needs terrain - it will need to be rectangular although when the terrain is generated you can always chop away the stuff you don't want.
- Make the dimensions multiples of 256 unless the terrain area is small, in which case you could use multiples of 128 or 64 etc. The smaller the triangles you create, the more of them there will be. More triangles means nicer looking, but higher demands on the PC to draw them. I have always used 256 to date.
- I give GtkGenSurf a rough description of the required geography and do the first generation.
- If it looks roughly ok in Radiant, compile it and go run around in ET.
- Note down the imperfections, like this bit is too tall, that bit is too steep, etc.
- Go back to Radiant, delete the terrain, refine the description to GtkGenSurf and try again.

- Repeat as many times as necessary to get it mostly right. This might take 20 goes or so.
- For fine tuning I drag individual triangle vertices about in Radiant. This is very tiresome and to be done last.



The brushes created this way for terrain should be of Detail type, otherwise you'd create a zillion portals in the compile. This also means that the undulating terrain does not block program line of sight, so for example the program will draw what's on the other side of a hill, even if a player couldn't see it. This is why in many maps you'll find buildings separating areas of the outdoors, so that the Structural brushes of the buildings can intervene between the Detail brushes of the terrain and reduce the demands on your graphics card and so improve the FPS.

This is why with large outdoor expanses like in Glider, 6Flags and 2tanks, I've had to be careful with the amount of cosmetic detail included, because so much of the map will be drawn all the time.

## Preparation

[\[Top\]](#)

GtkGenSurf is capable of producing terrain according to a number of styles, but I've found there is just one that seems to be the most practical, and I've stuck to it for all the terrain I've ever made.

It can base the undulations on the forms of waves, cylinders or fractals, or from a bitmap which specifies how the lumps and dips should be laid out. In practice I've found the bitmap method to be the most useful, so that is what I'll describe here.

You can create a rectangular bitmap, ie a graphical file of type .BMP, and by colouring the pixels in it in varying shades of black to white, tell GtkGenSurf how you want the terrain to rise and fall.

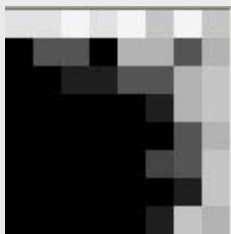
The tutorial map is 2048 \* 2048. We'll make the terrain mesh size 256 units (1 box in grid scale 9) as this will be fine for the demonstration. 2048 / 256 is 8, so the size of the bitmap will be 8 \* 8 pixels.

The tutorial map currently has lots of stuff in the bottom left quarter, with the rest more or less empty. We'll make the empty 3/4 a bit hilly, and make the bottom 1/4 flatter.

Use Paint Shop Pro or any graphics editing software to create an 8\*8 image. Make it **greyscale**, ie 256 shades of grey only. A coloured BMP will not do. Fill it with black.

Black represents 0 height. White means 255 high. The lighter the grey the greater the corresponding height, as interpreted by GtkGenSurf.

Make the top & right area some dappling of grey to give some slopes there. Here is a zoomed in image of an example 8\*8 BMP.



As can be seen I've left the bottom left corner flat for now, to avoid some of the features we've put there disappearing into the terrain.

Save the file as a .BMP file type, eg **tutorial.bmp**.

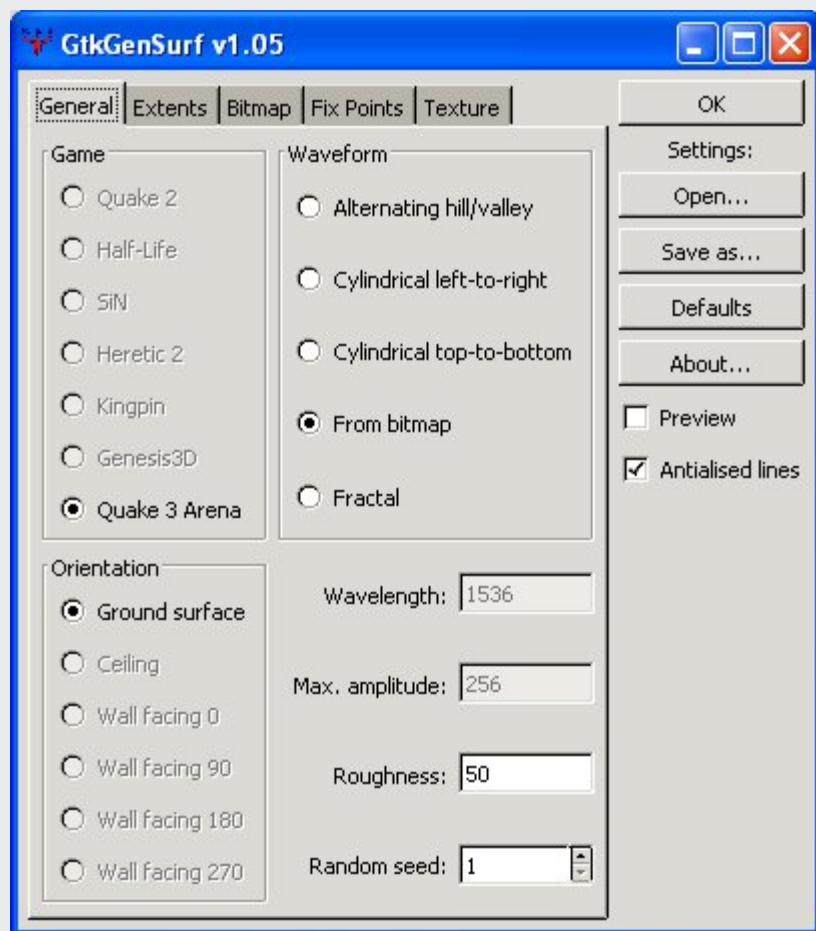
## GtkGenSurf

[\[Top\]](#)

Run Radiant and open the tutorial map. Select grid size 9.

Zoom/move the 2D view so you can see the whole map.

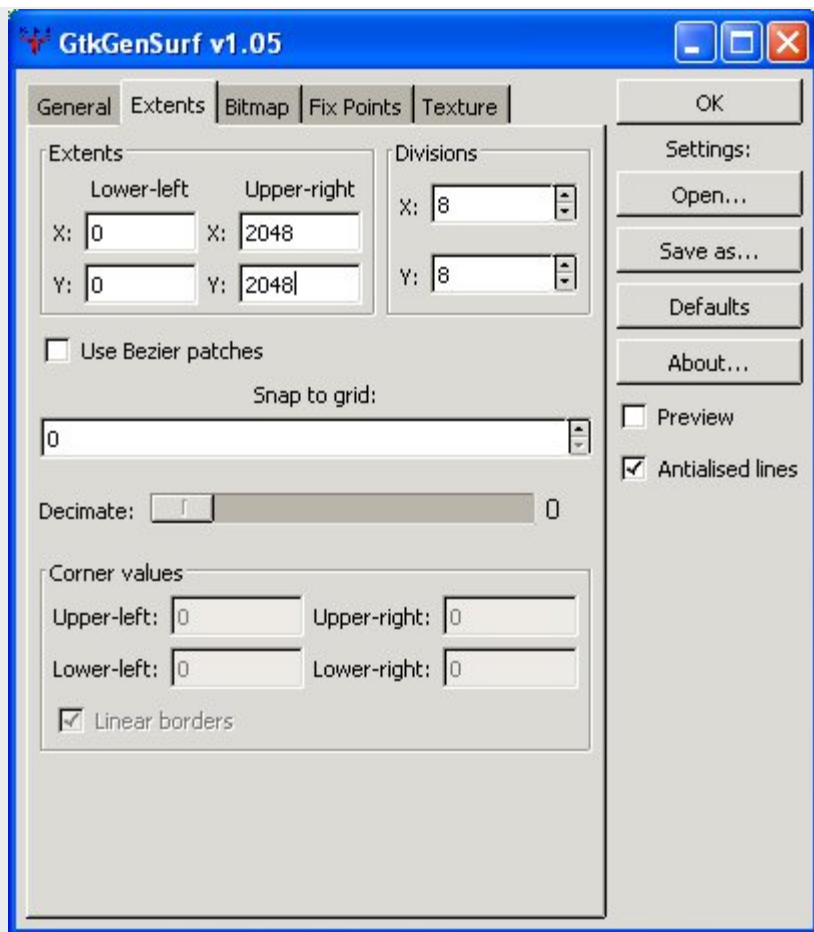
Click **Plugins/GtkGenSurf/Ground Surface...** and this window opens up.



Make sure that:

- "Quake 3 Arena" is chosen as the Game
- "Ground surface" is selected for Orientation
- "From bitmap" is selected for the Waveform
- A "Roughness" value (how much irregularity should be introduced to the undulations) of 50 is fine, and pick any random seed you fancy (a starting point for the random numbers used - who cares really).  
With a non-zero roughness it means that areas you've specified as black, ie flat at ground level, won't actually be completely flat, there will still be little curves and bumps.
- "Antialiased lines" ticked is nice

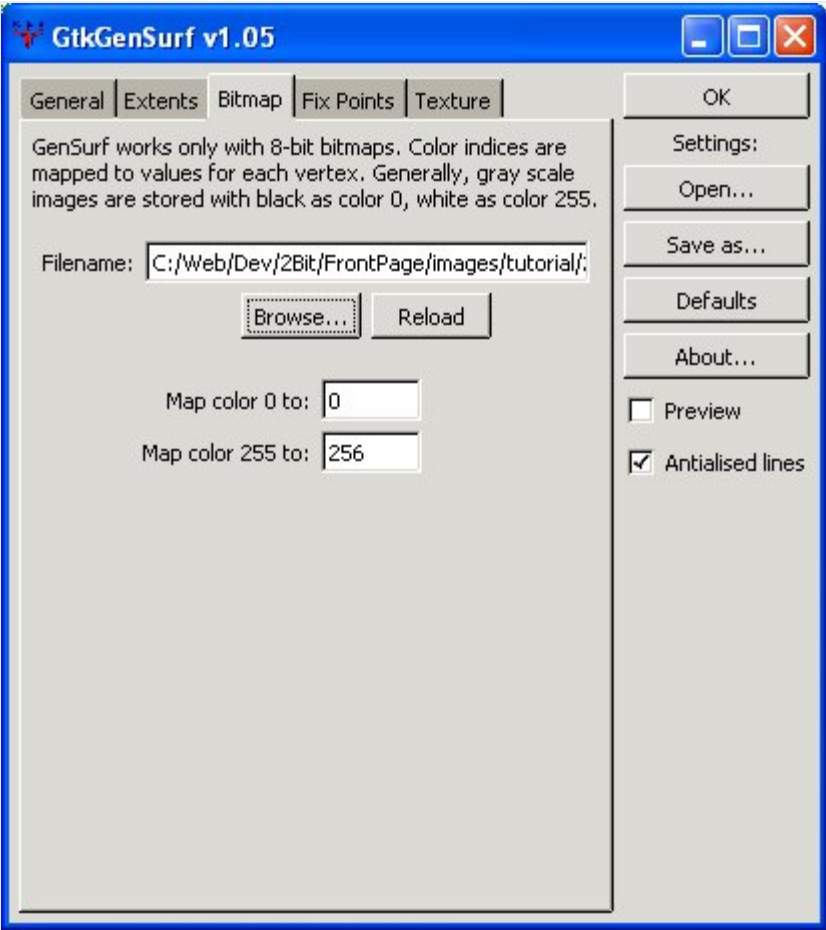
Click on **Extents** tab.



We must enter the map extents, ie tell GtkGenSurf the size of the area to create terrain for. Seeing as we are using the whole map and we created the map with the lower left corner at (0,0), this is easy: the bottom left is (0,0) and the top right is (2048,2048).

As it happens, the Divisions default is x=8 and y=8, which happens to be what we want for our 8\*8 terrain mesh. If the map had been 4096\*4096 and the box size were still 256, you'd enter 16 and 16 here.

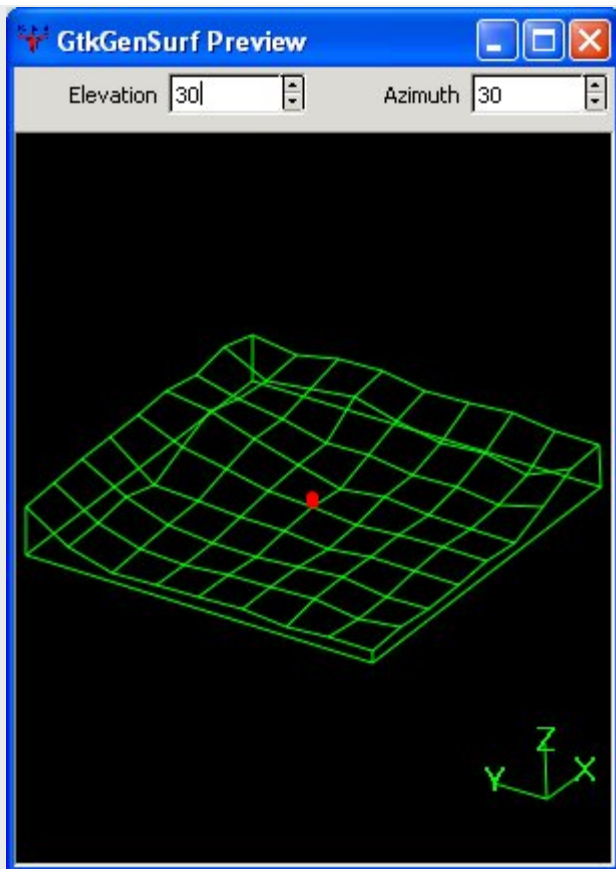
Click on **Bitmap** tab.



Browse to the BMP file you created. Don't worry about the map color boxes, they can be used to scale the undulations to increase the height range above 256, but we don't need that now.

Click the **Preview** box. This will show you what the geography will look like.





GtkGenSurf has some quirks and bugs. One bug is that if you run it again, even though the Preview box is ticked, it won't open the Preview window - you have to untick it and tick it again.

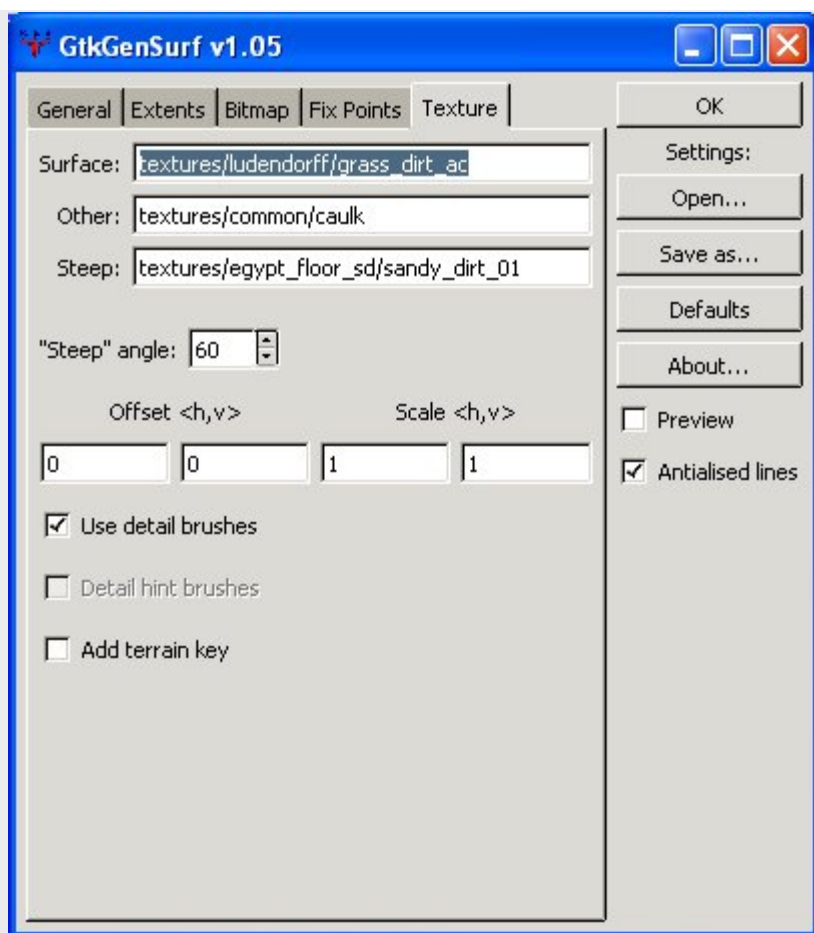
You can view the mesh from different angles by changing the Elevation and Azimuth of the viewpoint. Remember 30/30 is the default view.



Another oddity is that the 3D preview isn't easily matched to the top/bottom of your map. You'll have all sorts of fun trying to guess which bits of the terrain will go in which parts of your map. With a simple 8\*8 where the terrain is obvious, it's ok. When the terrain is 80\*80 with lots of bumps and valleys, it becomes less obvious. This is because of another quirk which you'll see next as we get the opportunity to fine tune any of the terrain grid intersections using the Fix Points feature.

By the way, I generally place the GtkGensurf window top left on my screen and put the preview window next to it, and then make the preview window as large as possible. This doesn't matter at 8\*8, but it will at 80\*80 or similar.

Click the **Texture** tab.



The **Surface** texture is the main texture that GtkGenSurf will use when it makes the terrain. You can change this to the texture you want. While I'm making the Ludendorff map I am using some grassy dirt texture. You should pick an organic texture, something like grass, dirt, snow, gravel, sand etc. There are some good grassy choices in the textures/temperate\_sd set, so for now I suggest you choose master\_grass\_dirt3. There is no "browse" option, so you have to type it in. Don't put .tga or .jpg at the end, just give the texture name.

The **Other** texture should be left as caulk.

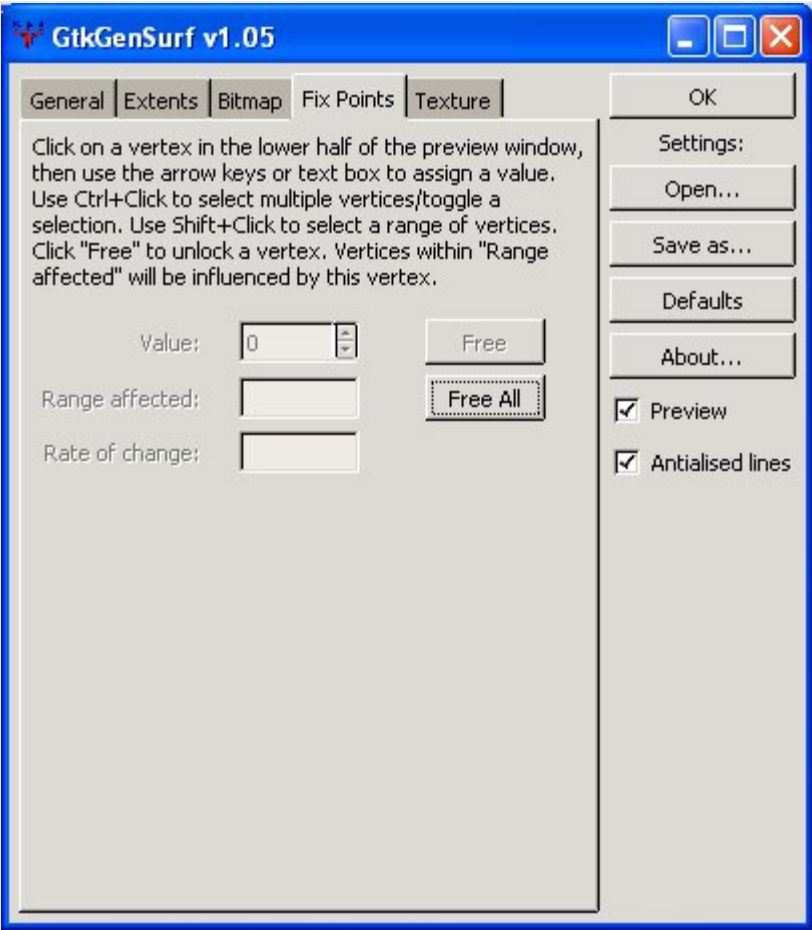
The **Steep** texture is the texture to use for steep angles, defined as 60 degrees or more unless you change it. You would make this either the same as the main surface, or if you wanted say hills to be grassy mainly but rocky on the steep slopes, you'd give a rocky texture here.

**Make sure the "Use detail brushes" box is ticked.**

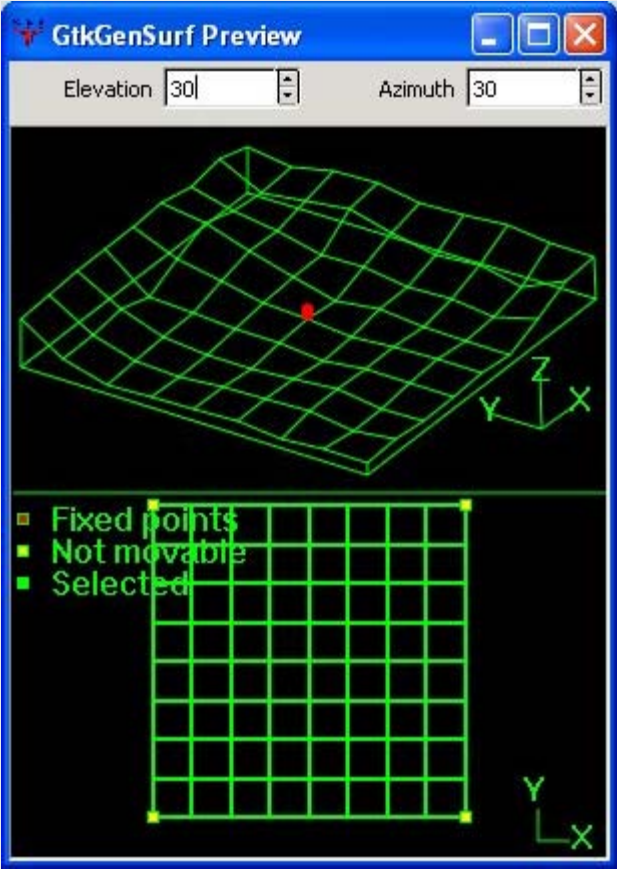


Yet another bug is that often GtkGenSurf will not use the texture you specified. It tends to get it right more often if the texture is already in use somewhere in your map, but this isn't guaranteed. Once it starts to get it wrong it's quite tiresome trying to convince it to use the right texture. It may also forget the texture used when you save an INI file to record the settings you are using. You may have to go into the INI file and change it by hand, as explained later. Sometimes closing Radiant and starting it again will help GtkGenSurf get its act together.

Click the **Fix Points** tab.



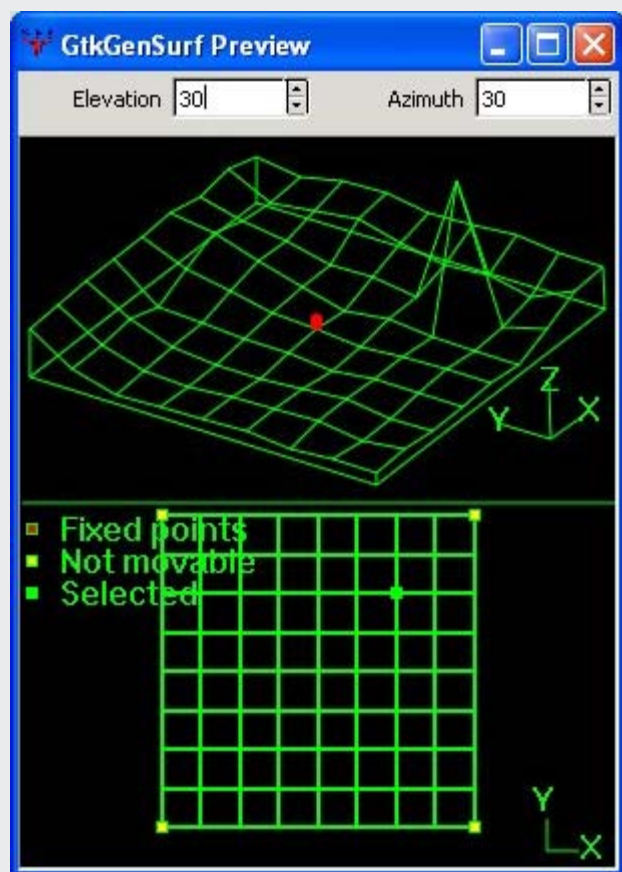
The main window of interest is the preview window.



We can change the height of any point (except the yellow ones) by clicking on an intersection, and then

entering the required height value. If you enter a value in the Range Affected box, nearby points will be adjusted too. As the adjacent points are 256 units away, you'd enter say 512 to affect your 2 nearest neighbours in all directions.

Click on a point and increase the height value so you can see the effect of your adjustment. I have made an exaggerated adjustment here to make it obvious.

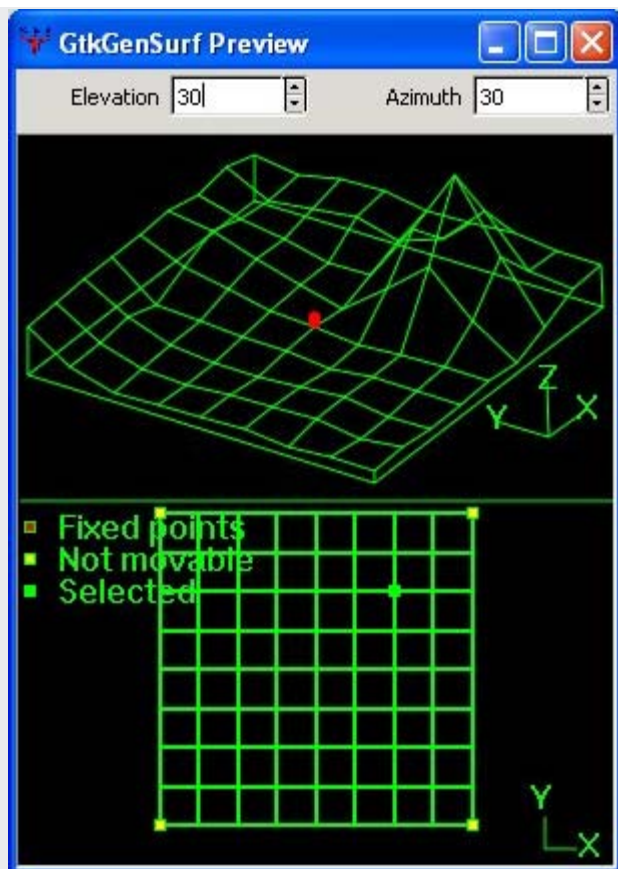


I don't really want this spike, so I can release it by clicking **Free** and it will return to its original value.

What you will find is that it is hard to equate where you click on the grid to where in the terrain you are actually affecting. This is a pain. Sometimes I flip the image upside down with the Elevation and spin it around with the Azimuth to get the left and rights to match up - but then the top and bottom are reversed. :(

It's trial and error and you must pick the view orientation that makes the most sense to you.

I'll repeat the point adjustment, but this time with a Range of 512.



You'll see how the neighbouring points are dragged upward. This is neat and goes some way to compensating for the aggro you'll have in trying to work out which point is which...

You can adjust multiple points simultaneously by ctrl+clicking on the grid, and you can do rectangular ranges by shift+clicking.

For now we'll just leave it to the default values. If you've made the grid a mess, click **Free All**.

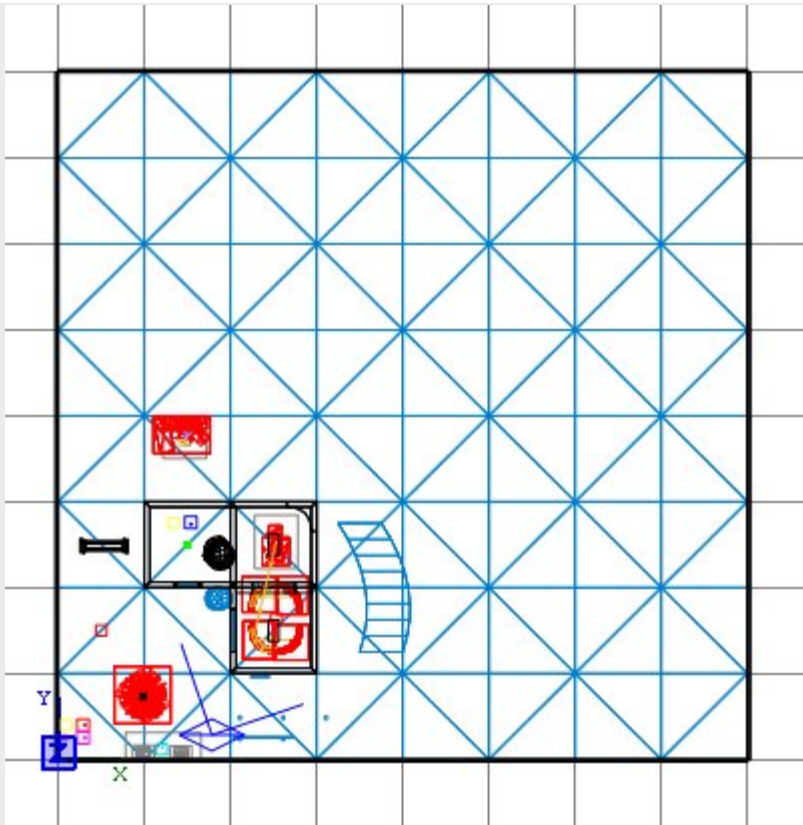
Nearly done. To record all these settings so you don't start from scratch next time, you should now click **Save As...** and save your settings as **tutorial.ini** in the Radiant folder.



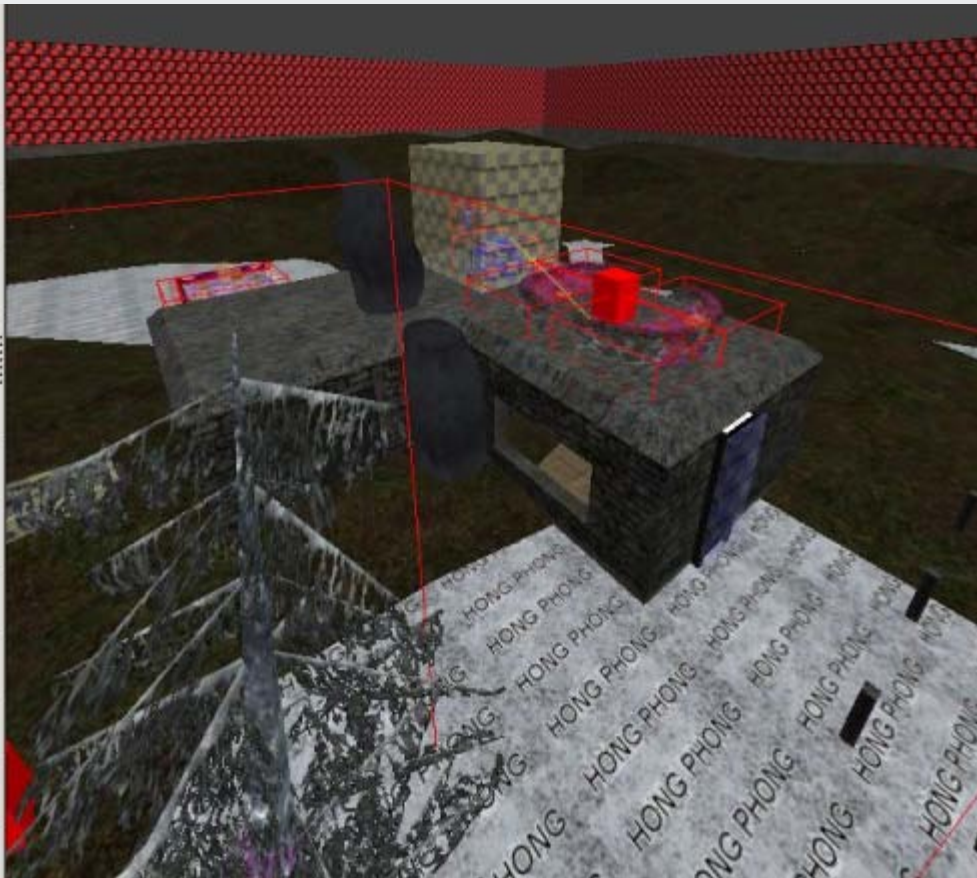
Yet another bug is sometimes the Save As dialogue box will have gibberish in the "Save as type" box. Ignore it and just type the name of the file you want to save to.

You are now ready to create the terrain. Click the **Ok** button on the GtkGenSurf window. It will create the brushes, being 2 triangles per box, already grouped (so they show as blue triangles) at the (0,0,0) co-ord.





Your view should look something like this:



We have new terrain and the old floor overlapping. Either we'll need to move everything up a bit, or move the old floor down. We can't just delete the old floor because it forms part of the structural boundary of the map. In general you'll make maps that are totally enclosed in hull caulk cubes, with all 6 faces of the surrounding brushes given hull caulk, so that you are free to create/delete anything inside it without worrying

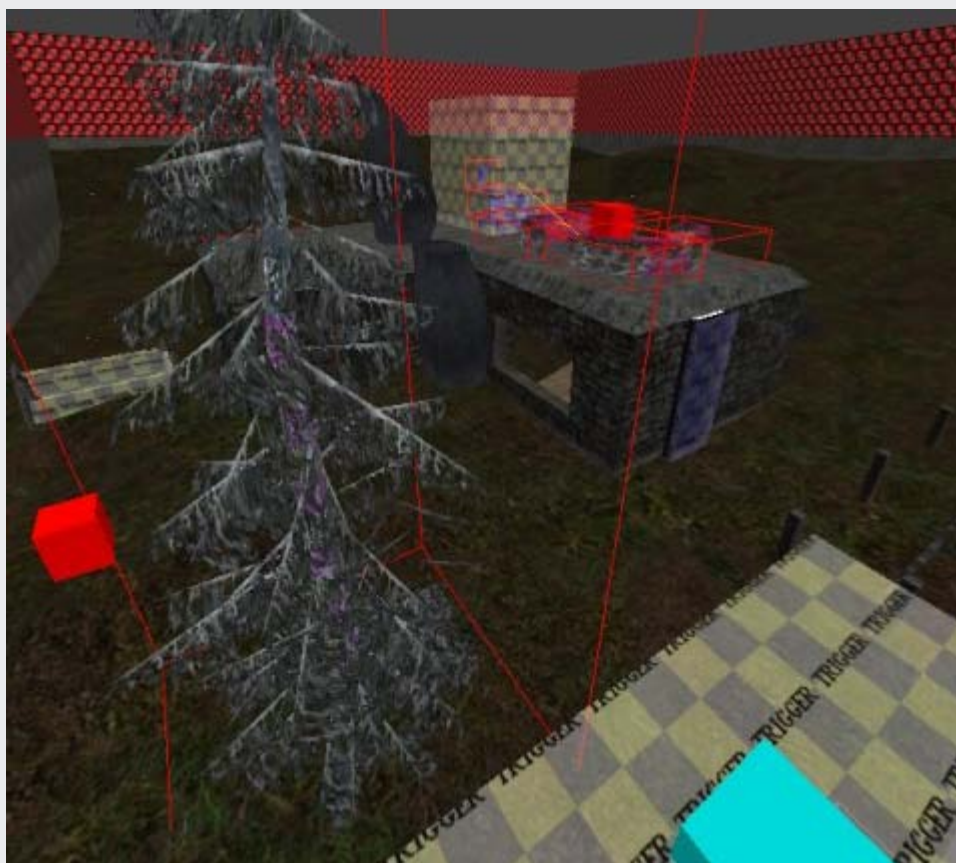
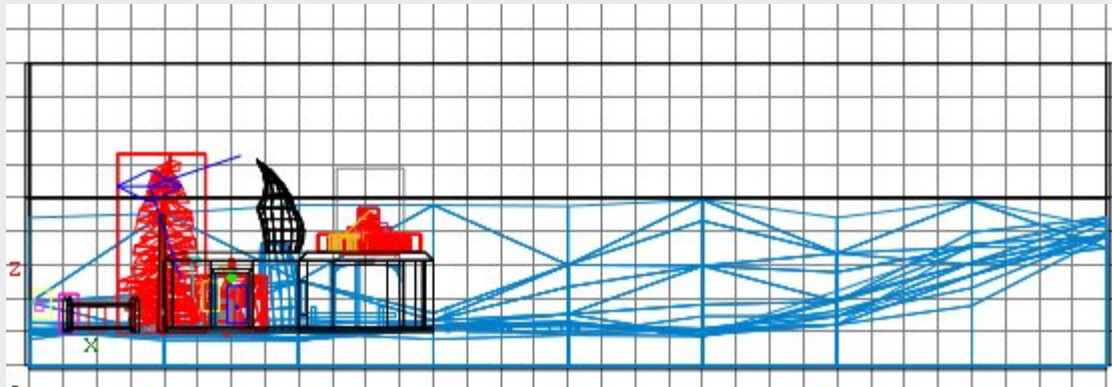


that you are making a hole into the void.

For the sake of simplicity we'll just move the floor down a notch. Get a 2D side view. Change to grid size 7. Select the old floor brush and move it down one notch. Click on the hull\_caulk texture in the textures window to give the whole brush that texture - we don't need it to be drawn as snow any more.

Press ESC. Select all 4 of the surrounding wall brushes, and extend them down one notch to meet the bottom hull\_caulk brush, thus making the solid surrounding cube again.

Press ESC. Your view should now look something like this:



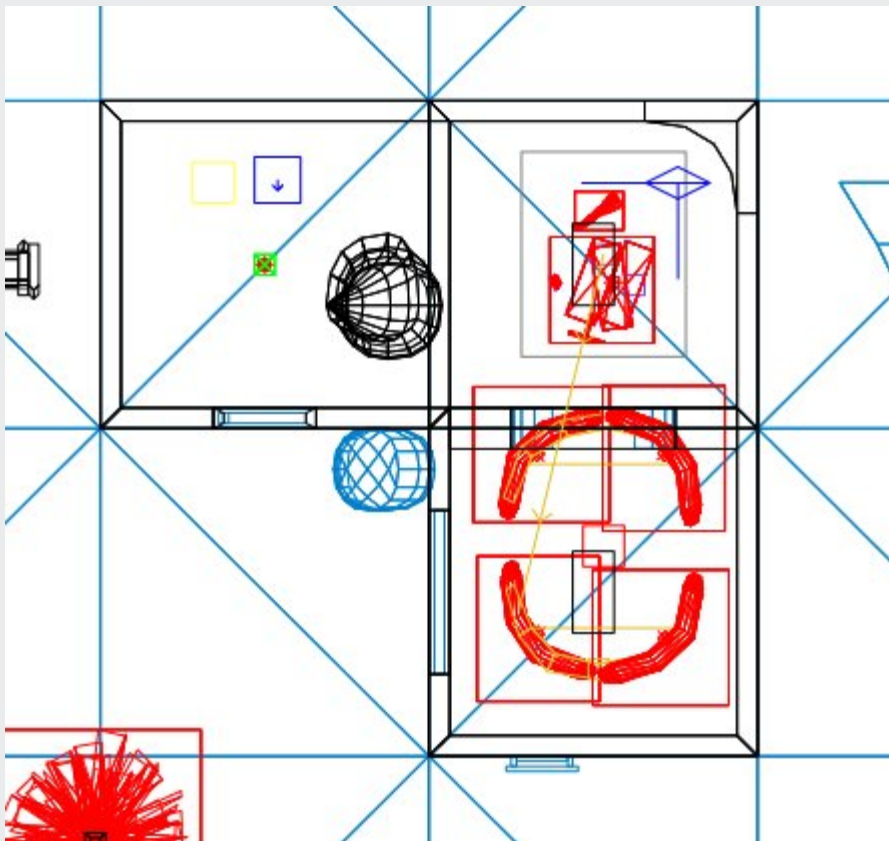
If you generally don't like how the terrain has come out, rather than specific detailed parts which we could fix by hand, select and delete the terrain group and go back to GtkGenSurf. It will still have your details so you can just Fix Points differently and try again.

If though you have exited Radiant, GtkGenSurf will not know your terrain settings, so you will first have to use GtkGenSurf's **Open...** button and tell it to open "tutorial.ini".

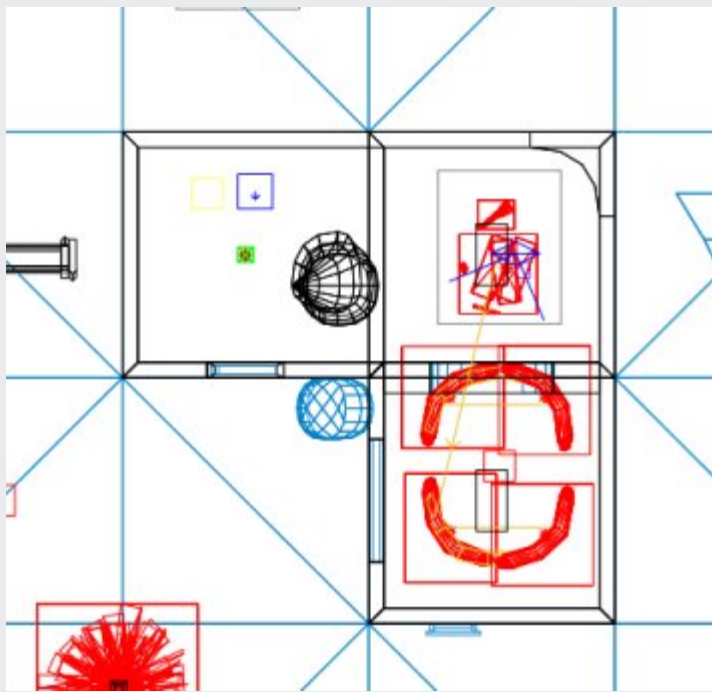
An important detail to note is that there is a chunk of terrain inside the building.



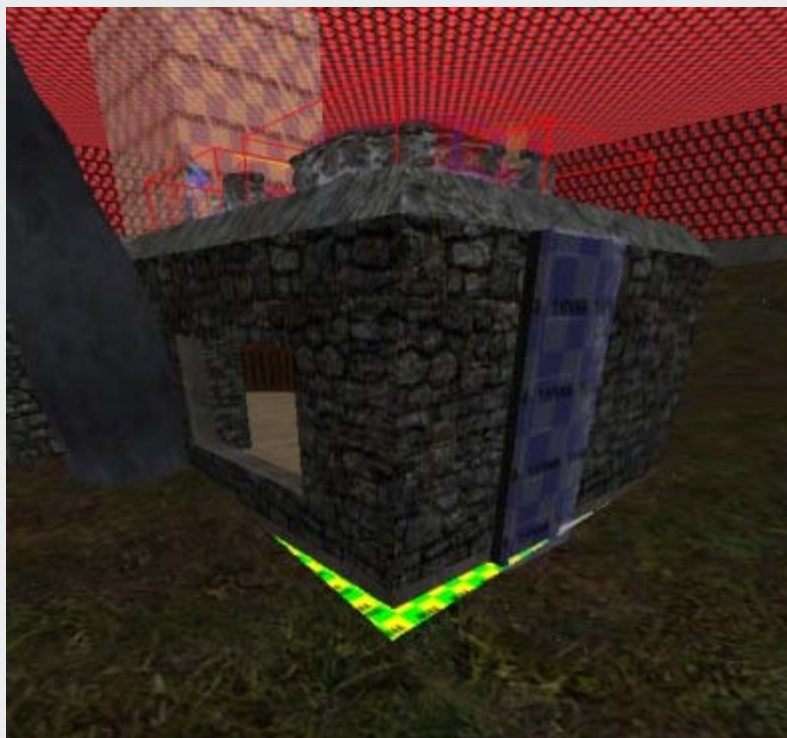
It's at this point that it starts to become clear why we have taken such trouble to make our buildings line up along big grid lines where possible. Choose grid size 9 and get the overhead 2D and you can see the terrain triangles that fall within the building. There are 6 of them.



Select all 6 triangles in the 3D view, by first hiding the intervening floor brushes, and delete them. Press shift+H to reveal the floor brushes again and it should look like this:



This will have revealed some gaps around the bottom of the building, as shown here:



The better solution to all this is to manually adjust the triangles in Radiant, but we'll come to that in the next lesson. For now just select all the floor brushes again and extend them downwards in the 2D side view using grid scale 5. This will still leave a problem with the door, because the terrain comes up the door a bit, and you'll see its caulk when opening the door from the inside. We'll fix all this in the next lesson.

Save, compile and run around in your new hilly environment.

By the way, when taking screenshots from ET you usually have to increase the brightness (eg 40%) and contrast (eg 20%) otherwise the JPG looks too gloomy.





Whether or not you can plant landmines will depend on the texture you used. If it's a shader with landmines enabled, you'll be able to plant landmines.



A neat way to see if a texture is actually a shader (ie there is a bit of script-like information for it which gives it properties instead of just being a picture) is to shift+click on the texture of interest in the textures window. Make sure you have pressed ESC first or you will apply the texture to any selected brushes/faces.

If the image is a plain texture, Radiant will report something like: ERROR: textures/egypt\_floor\_sd/block-16sq is not a shader, it's a texture.

If it's a shader, ie there is an entry for it in a shader file, Radiant will open the shader file using Wordpad or whatever you have associated with .shader file types. Find the texture name in there and see if "surfaceparm landmine" is given under it. If it is, you can plant landmines on faces with this texture.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 24

### Topics

#### Fine tuning the terrain by dragging vertices

[Fine tuning the terrain by dragging vertices](#)

[Back to main menu](#)

#### Fine tuning the terrain by dragging vertices

[\[Top\]](#)

Run Radiant and open the tutorial map.

It will help you follow this lesson if you use the same terrain BMP as I did. So download it [here](#), delete your existing terrain mesh (select the group and delete it), and use the supplied image1.bmp in GtkGenSurf to generate the terrain I have used.

Delete the couple of silly cones that we created earlier, to give us some clear space - we don't really need them, they were just for illustration of technique.

In the previous lesson we created the terrain which was generally fine, but a chunk of it obscured the door. To clear this chunk out of the way we will edit the terrain manually by dragging their vertices about. Manual editing is something you should do last, because if you decide subsequently to regenerate the terrain with GtkGenSurf, you will lose your editing (unless you go to great lengths to retain your edits and restore them after the regeneration - best avoided).



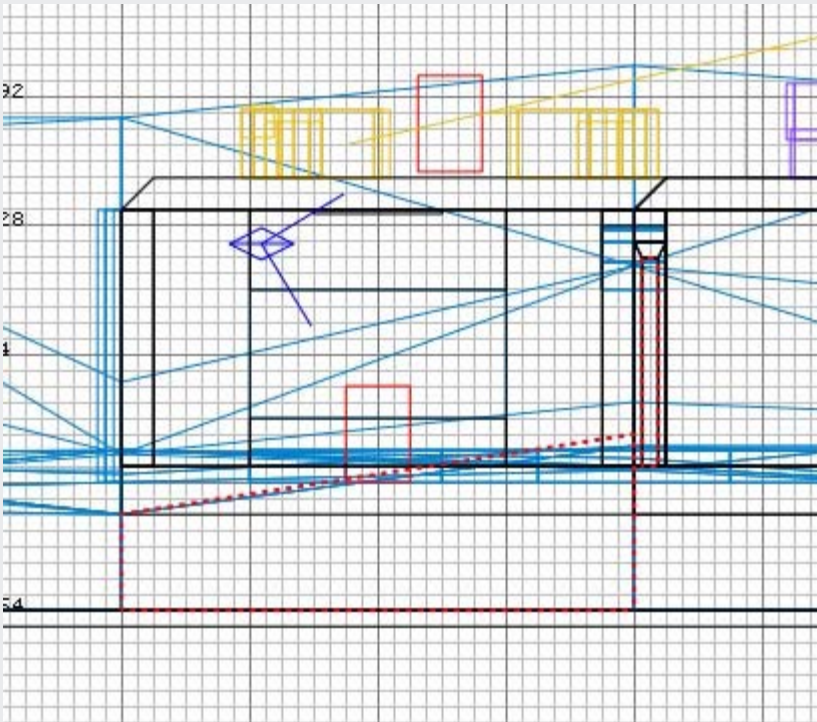
Manual editing of vertices is error prone. Very. Backup or save your map before you start. It is also highly likely that Radiant will create errors even if what you have done is correct. We can usually sort these, but then again, that backup may come in very handy. You have been warned :)

In the 3D view look, select the door and the terrain that is blocking it.



We've selected the door to help us identify the selected terrain triangle in the 2D view, and to see how far down we have to lower it.

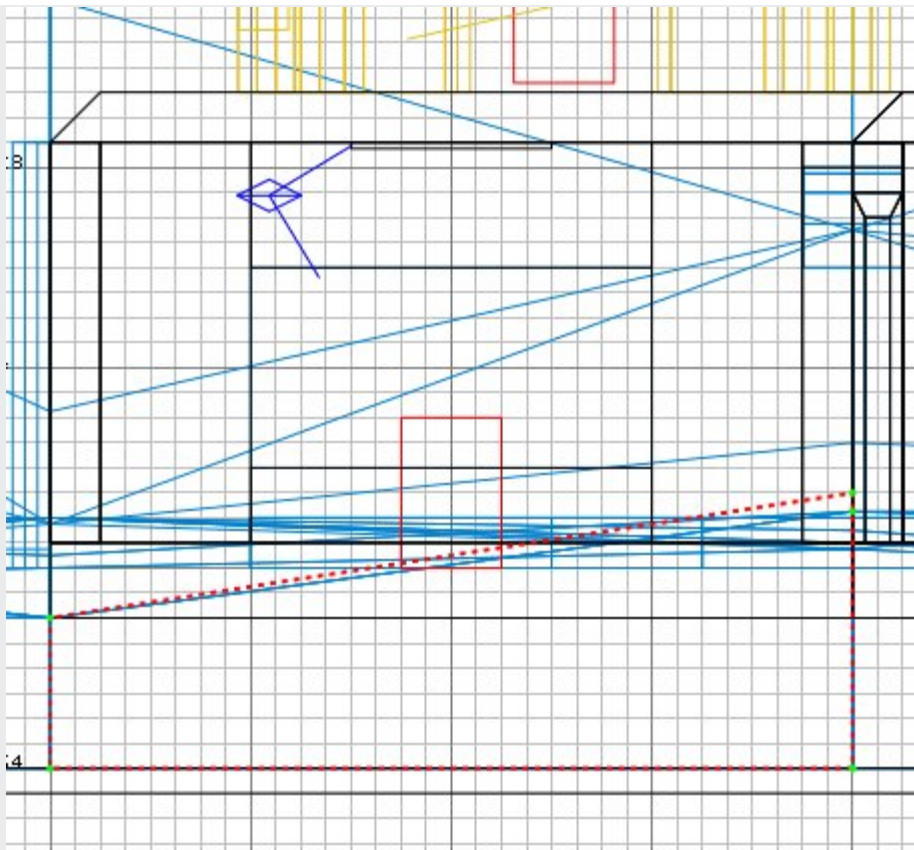
In the 2D view, get the side view that makes the selection the clearest to work on.



Deselect the door (in the 3D view will be easiest).

Press V to engage vertex mode. Little green dots appear at each vertex of the triangle.

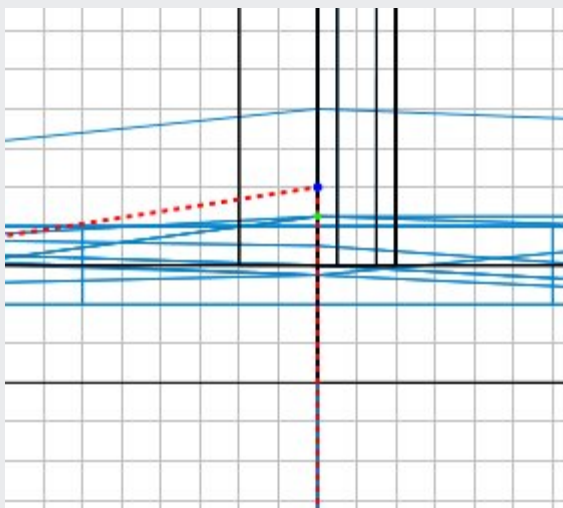




This illustrates one of the problems encountered with this activity: trying to spot which green dot to use. You can click on the required vertex in the 3D mode if you can see/guess where it is - or you click on the one you think it is in the 2D view. The one we want is the top right dot.

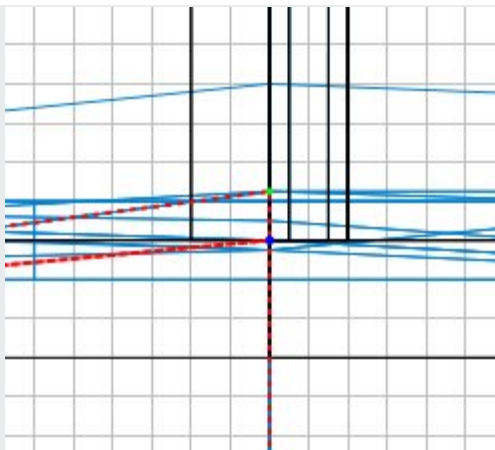


You should zoom in on the dot to ensure it is on a grid intersection. If it isn't, reduce the grid scale until it is. If you attempt to move the dot when it didn't start on an intersection you will make an almighty mess of your triangles.



If it isn't clear how far you need to drag the vertex, you can briefly select the door or the floor so you can see the required baseline, but you will need to press V again.

Drag the vertex down to the required baseline.

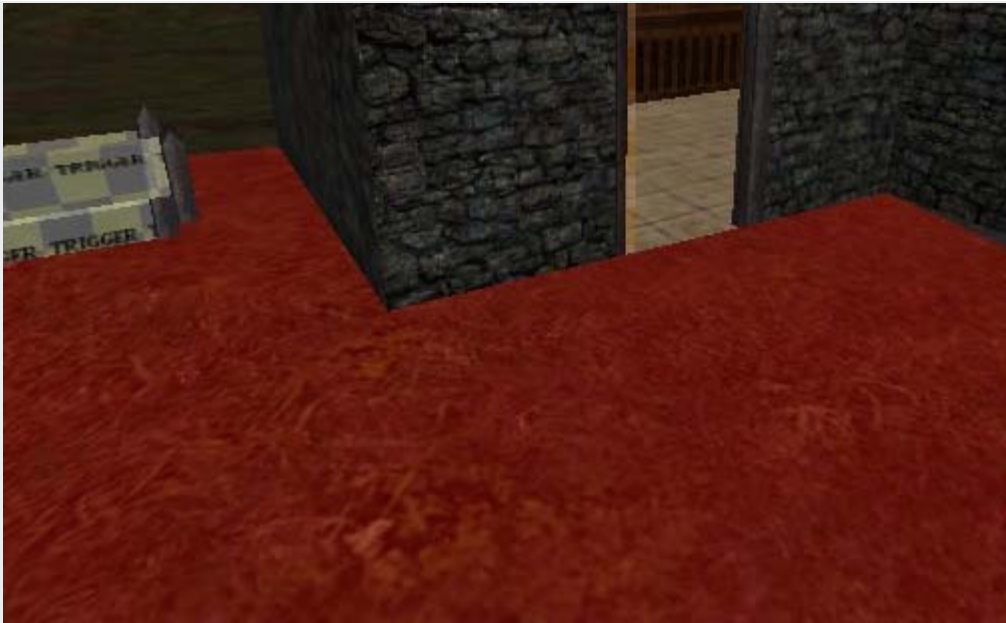


In the 3D view this now *appears* that the terrain is out of the way. Check it by going inside the building, selecting the door and hiding it.

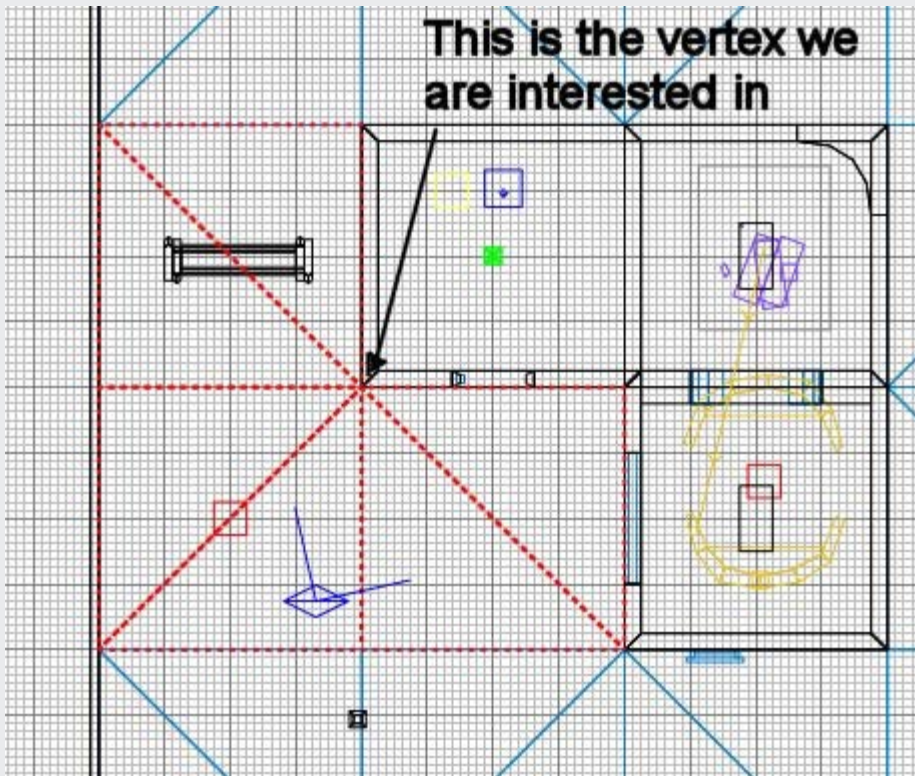


As you can see, there is still some caulk showing, so we need to drag another vertex down too. But this time it is slightly more involved, and in fact is much more the usual state of affairs, being that we'll need to drag multiple vertices at the same time.

Take your 3D view outside again. Get the 2D overhead view. In the 3D view (it is easiest unless you hide the sky to make 2D selection easy) select all of the triangles that share the vertex at the corner of the building. There are 6 triangles to select.

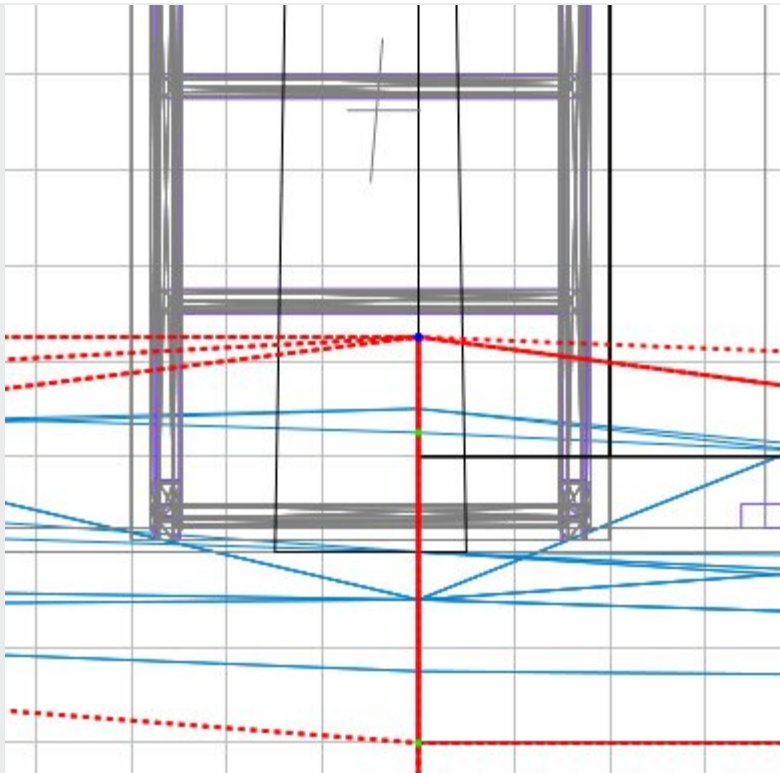


This is what the 2D view should show:



Press V and select the required vertex - this gets quite hard in 2D but sometimes with the rolling hills it can be impossible in 3D so even the quite hard 2D view is easier :(



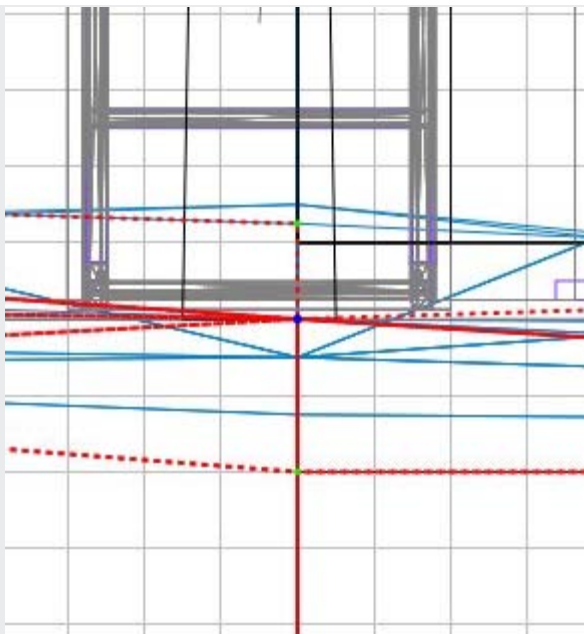


As can be seen, the required vertex is not grid aligned. As we are at grid scale 4 and the dot is clearly 1/4 up from a line, we'll need to drop down 2 scales: press 2 to get grid scale 2.

Ok now we have the required scale, but it has become hard to see where we have to drag it. The best thing is to drag it just enough to put it onto a bigger, better grid scale. So drag it one notch down, then press 4 to go back to grid scale 4.

Drag it down a couple of notches, and it should look like this:





Press ESC to let go of all the brushes. You can see now that the door is unobscured, so you can reveal the door again.



When you edit your terrain mesh this way, Radiant may well generate duplicate planes, which will cause your compilation to open the debug window. So after each editing session and before you



compile, you should click the Brush Cleanup button. If it finds errors it deletes them and tells you how many invalid planes were removed. It also selects the mesh group. You should deselect the group and click the button again until the Brush Cleanup reports 0 invalid/duplicate planes removed. Finally deselect the group and save your work.



There will be occasions where the vertex you wish to manipulate shares its Z co-ordinate with another vertex of one or more of the selected triangles (ie, that point of the triangle shares its height with other points on the selected triangles). In the 2D view you will not always be able to see the other vertices at the same height as they may be obscuring each other depending on whether you are looking down the X or Y axis at them.

When you try to select and drag the vertex, you may find you are dragging the wrong vertex :(

This is because even though you may have selected the right vertex in the 3D view, when you click on the blue dot in the 2D view, Radiant will realize that there is another vertex at that point which is closer to you, the viewer, and so handily selects that one for you. You may notice this has happened in the 3D view - but if you don't, you're going to start dragging the wrong vertices. Watch out for this. If you find this happening, try using the other 2D side view. Or selecting the other troublesome triangles and move their offending vertex up/down a little to get it out of the way. Final option is to carefully drag a notch up/down in the 3D view - this is hazardous as it is easy to accidentally drag sideways.

Final tip is to remind you to get down to the lowest grid scale needed to get your vertex onto a grid intersection. Then move it to a nearby larger scale grid intersection, change the grid scale up accordingly by pressing a bigger number, even 9 if possible, then zoom out in the 2D view and drag in larger increments. Always try to leave any dragged vertices on nice big grid intersections, it will help you a lot.

Do not drag vertices sideways in the 2D view, only up and down!!!!

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 25

### Topics

#### Skyboxes

[Intro](#)

[Skyboxes](#)

[Back to main menu](#)

### Intro

[\[Top\]](#)

A skybox is a device to make it appear that there is more terrain in the far background around your map.

You don't need one if your map is entirely indoors with no distance view outside. Its use is optional if you have outdoor areas or views.

If you don't use one, your sky will look like the sky texture you've chosen. If you do use one, the sky faces will instead show the distant background. This works best if the sky ceiling is 512 units high. If it gets hugely high, like 3000+, the projected image gets so elongated vertically that it starts to look bad. In which case you may be better off without a skybox.

To create one you will place a self-contained hollow cube somewhere outside your map volume, ie out in the void somewhere. The inner faces of the cube refer to the image to be used as distant background, and a special camera entity is placed in the middle of the cube so that the compiler knows it is being used as a skybox.

You will need to add a skybox definition to your <yourmapname>.shader file.

Making a skybox texture is a bit of an art, and not easily done by the uninformed like you and me :)

Happily, people create skybox images and make them freely available. One such excellent chap is Amethyst7. You can get a number of great skybox textures from him at:

<http://amethyst7.gotdoofed.com/env.htm>. If you do use one of his, please credit him in the ReadMe that you'll create for your PK3.

### Skyboxes

[\[Top\]](#)

So you won't have to create one from scratch, download this file: [prefab\\_skybox.zip](#)

This is the Siege skybox, one of Amethyst7's.

- Create an empty text file in the etmain/scripts folder called <yourmapname>.shader.



- Create a folder called "skybox" in the etmain/textures/<yourmapname> folder.
- Open the zip file.
- Put the .map file into your etmain/maps/prefabs folder.
- Put the .shader file into your etmain/scripts folder.
- Put all of the .jpg files into the etmain/textures/<yourmapname>/skybox folder.
- Edit the prefab\_skybox.map file and change all references of <yourmapname> to "tutorial" (no quotes) or whatever map name you are using.
- Do the same with the prefab\_skybox.shader file.
- Copy the text in prefab\_skybox.shader into the <yourmapname>.shader file.

Make sure you have "<yourmapname>" eg "tutorial" added to the shaderlist.txt file.

Run Radiant and open the map. Import the prefab\_skybox.map into it, and move the cube without resizing it until it is ***outside*** your own map volume, that is, it doesn't come anywhere within your own map space. I always put it near the bottom, off to the left.

Compile your map and go take a look in ET. You should see some nice snowy hills in the background with clouds scudding across the sky :)

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 26

### Topics

#### Bespoke graphics

[Bespoke graphics](#)

[Back to main menu](#)

#### Bespoke graphics

[\[Top\]](#)

If you want to include your own textures, you simply create them using an image editor like Paint Shop Pro, and store them in the etmain/textures/<yourmapname> folder. They will need to have widths and heights of 64, 128, 256 or 512 pixels. If your image isn't precisely any of those sizes, maybe it's an irregular shape like a person outline, then make the dimensions of the image just large enough to accommodate your irregular picture, still keeping to one of those 4 pixel sizes, and just use black for the unwanted image area.

You can always crop a brush to remove the unwanted black surround.

All textures have a shader, even the ones you haven't created a shader for. This is because ET uses a default shader for any texture lacking a user-defined shader. A shader tells ET what special properties apply to the faces with this texture on it. The default shader would have things like: use the default footsteps sound, the textured face blocks players and bullets and missiles, the faces should show bullet holes and scorch marks etc.

If you wanted areas of transparency you must create a shader for your texture, else the transparent area will just show as black or whatever colour you used to represent transparent.

We'll cover shaders later on.

Be aware that if you made a cubic brush and applied your texture to all of them, 3 would show your image as is, and 3 would show a mirror image of it. If you had writing on your texture you'd need to make a copy of it, apply your own mirror image, and use that one for the 3 "wrong" faces.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 27

### Topics

#### Making a constructible object

[Preparation](#)

[Making the constructible](#)

[Writing the script](#)

[Back to main menu](#)

### Preparation

[\[Top\]](#)

Constructibles are best included by importing one made earlier, rather than create all the components manually. I've found the best way is to have a template map and script, edit them to yield the required names and then import them ready assembled straight into the map.

I have put an allied constructible template [here](#) for you to download. Unzip it and put the 2 files into your maps/prefabs folder. Don't amend these, just use them as the template to stamp out allied constructibles as you need them.

Make a copy of [etmain/maps/prefabs/prefab\\_allied\\_constructible\\_template.map](#) and call it [etmain/maps/\\_temp.map](#). By calling the copy "\_temp" it will appear early in your list when you import it, and you know to chuck it away later on because it is "temp".

Edit \_temp.map with Wordpad or similar. Make these changes:

- Replace all instances of "alliedconstruct\_n" with "alliedconstruct\_1" as this is the first allied construct to add to the map. You could change the name to something more meaningful if you like, eg "allied\_ramp", but I'm sure you get the principle.
- Replace "Allied Item" with the name to appear on the Command Map, in this instance we'll make an assault ramp, so make it "Assault Ramp".
- Replace "Allied Construction" with the words to appear next to "you are near...". In this example, "the Assault Ramp".
- Save and close the file.

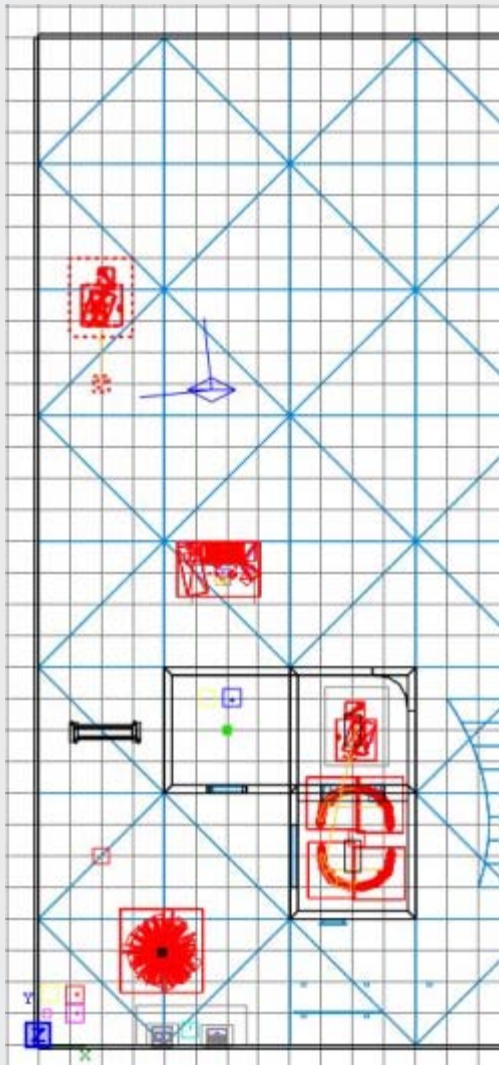
### Making the constructible

[\[Top\]](#)

Run Radiant and open the map.

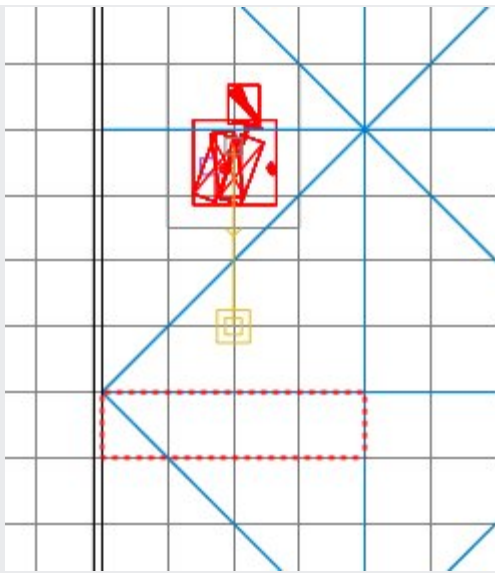
```
Import "_temp.map".
```

While it is still selected, drag it into some space in the map, as shown: It should already be at about the right height, but in your own creations you'll need to make sure the crates and surrounding trigger are placed at the right height - the ramp we'll build where we want it, so the little box that is the template constructible can go anywhere at the moment.

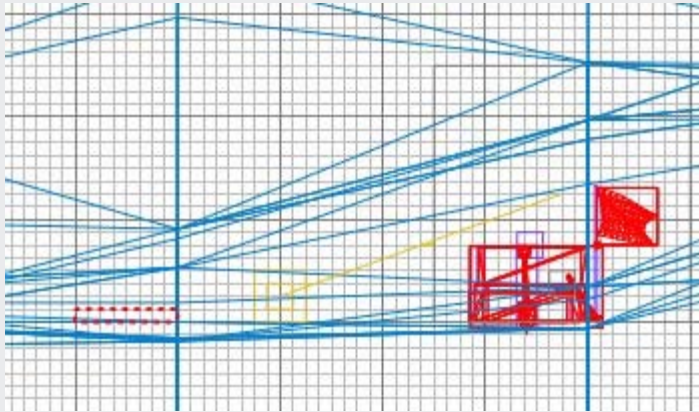


Press ESC. Let's make the ramp. You could of course make anything you like as a constructible, but for this example we're going to make a very simple wooden ramp.

Draw a brush as shown.



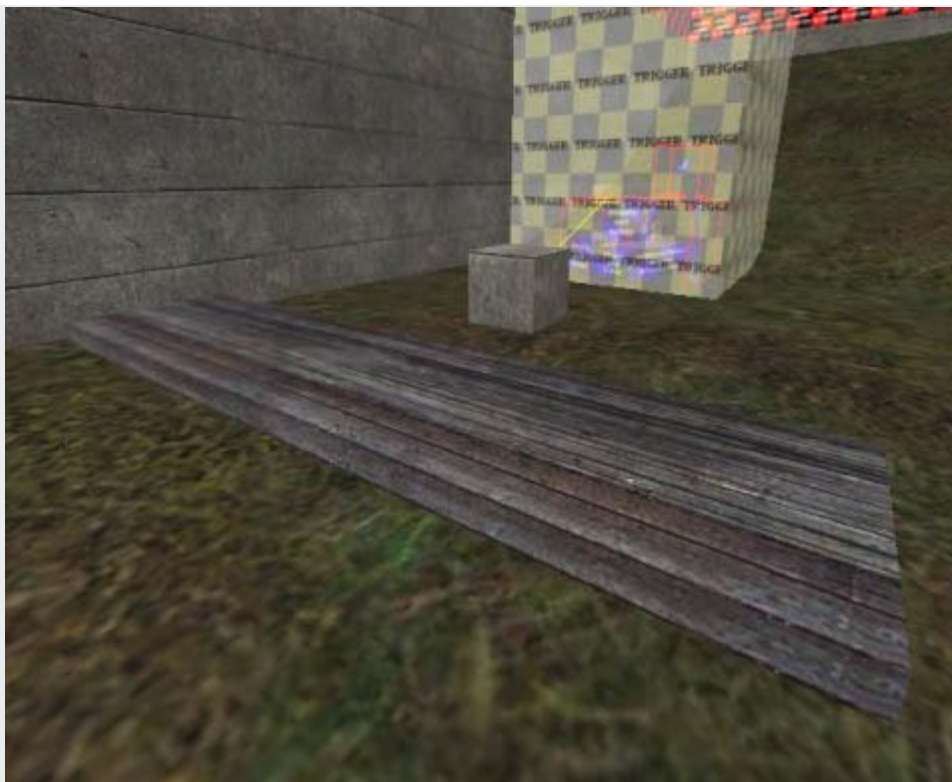
In the 2D side view, make it a reasonable thickness for a wooden ramp. I used grid size 4 and made it one notch thick.



We'll assume the ramp can be seen from all angles, so there will be no caulked surfaces. Therefore, make the whole brush wooden by clicking on the wood\_c01 texture in the textures window.

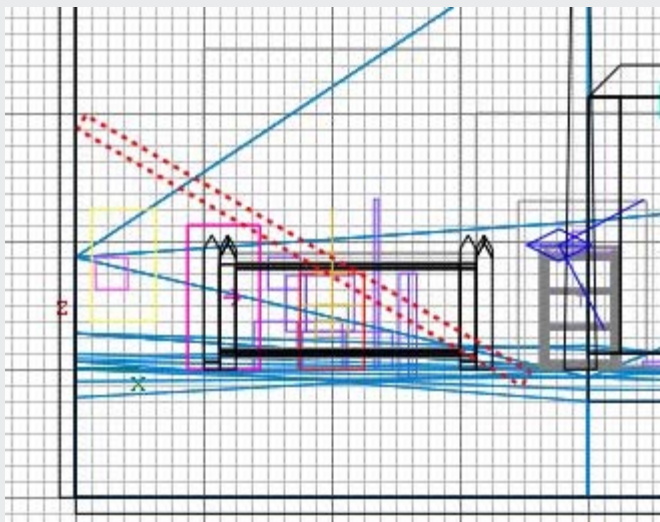
If the textures are aligned the wrong way, press S and rotate them by 90 degrees, and click Done.

Make the brush Detail and by now you should have something like this:



Now we'll angle it and put it against the wall. Select the brush and click Selection/Rotate/Arbitrary Rotation, put "30" in the "Y" box and click OK. Remember to use nice round degrees of rotation whenever possible - it gets very nasty trying to get other elements to line up if you don't.

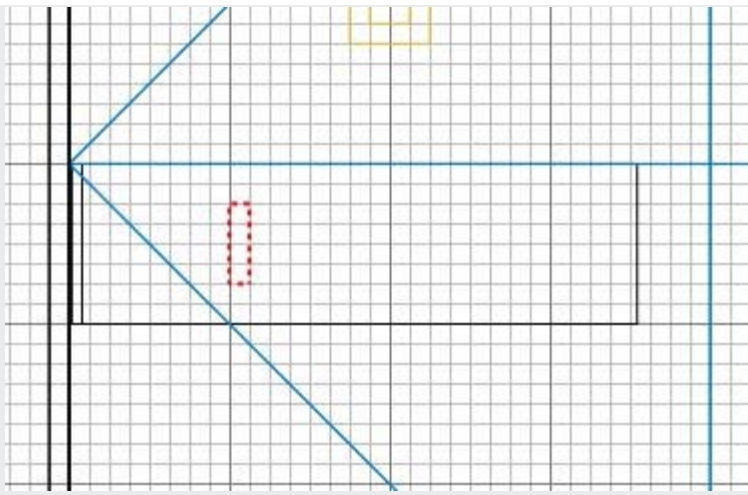
Now slide the brush so that it rests against the wall and on the surface of the grass. The following image has models filtered to make it easier to see.



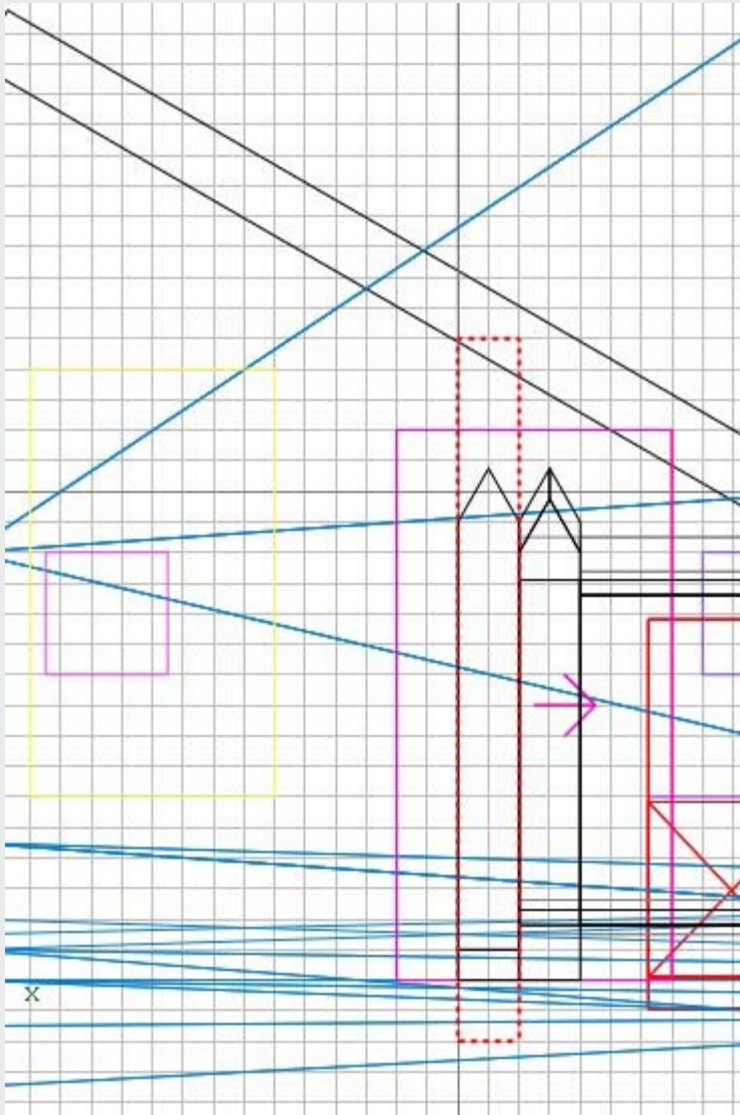
Press ESC. You'll see that the edges of the ramp now have misaligned textures. Select the 2 affected faces, press S, enter "-30" in the Rotate Offset box and click done.

Press ESC. We'll add an upright support. Draw a brush:

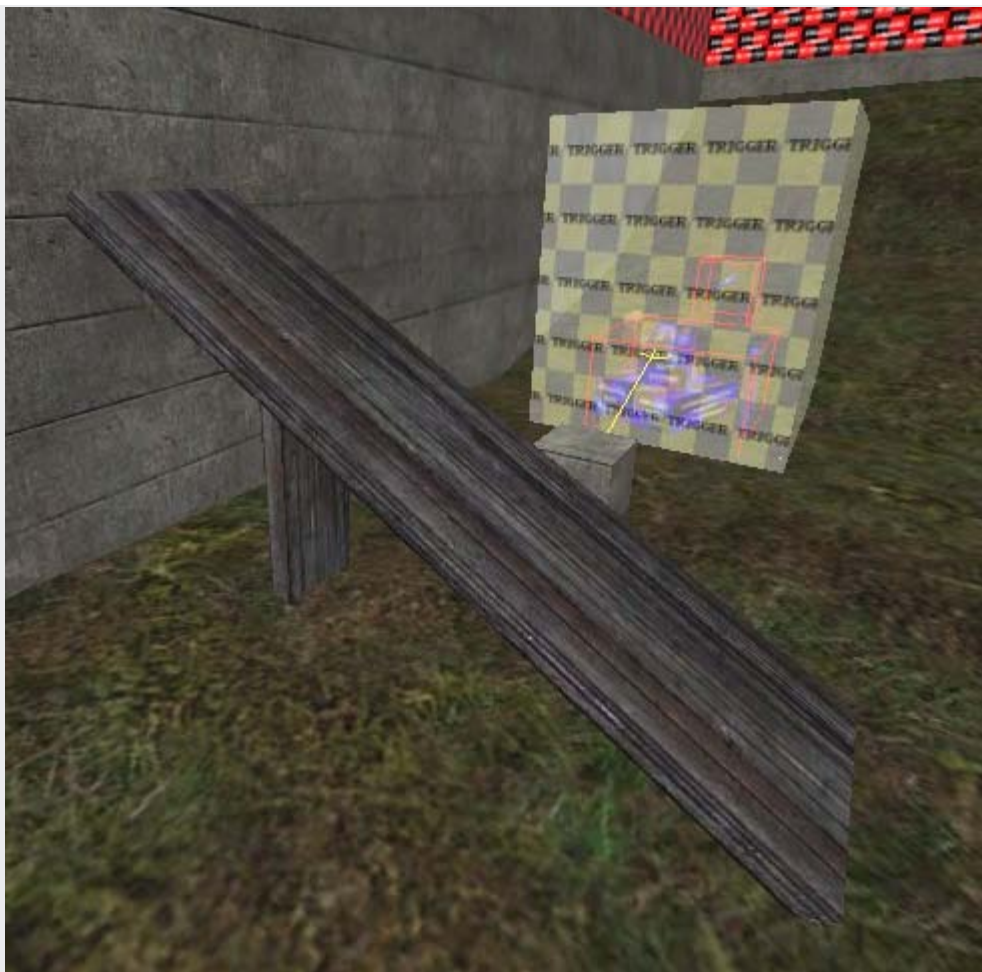




Caulk it and position/resize it as shown so that it supports the ramp.



Press ESC then select the 4 visible faces and make them the same wooden texture. You should now have a ramp like this, ready to be made into our constructible.



Select both ramp brushes, **then** select the constructible box brush. Right-click in the 2D and select "Move into entity". You have now made the ramp part of the constructible. Press ESC. Select the box brush and delete it. This will reveal the origin brush inside it. The origin brush tells ET roughly where the middle of the constructible is deemed to be. Select the origin brush and move it to the middle of the ramp.



Press ESC. Prove that the origin brush and both ramp brushes are now one entity by shift+alt+clicking one of the brushes in the 3D view - they should all be selected.

Press ESC. Save and compile the map, but don't go into ET yet - we have to write the script to make it work.

Delete the "\_temp.map" file, we don't want that any more.

## Writing the script

[\[Top\]](#)

Using Wordpad or similar, open the `prefab_allied_constructible_template.script` file. Copy all the text, and paste it into the `tutorial.script` file, after the `game_manager` section and before the "axis construct" section.

Replace all 7 instances of `alliedconstruct_n` with `alliedconstruct_1`.

Where it says "Allied Team have built the box!" change it to "Assault Ramp constructed!" or whatever you want. You can use the usual ET colouring convention if you want words in different colours, eg a prefix of "^3" would make the text yellow.

Where it says "Axis have destroyed the box" change it to "Assault Ramp destroyed!".

Later on we'll add some speech but we'll skip that for now.

As the ramp will be destroyable by a satchel, we can leave the `constructible_class` set to 2.

Save and close the script file.

Run ET and have fun making and breaking your assault ramp :) In practice, you wouldn't place a constructible so close to a CP or other constructibles/destructibles.

I'll get some Axis templates together for you shortly.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 28

### Topics

#### Making a destructible object like a gate

[Preparation](#)

[Making the destructible](#)

[Writing the script](#)

[Back to main menu](#)

### Preparation

[\[Top\]](#)

Like constructibles, destructibles are best made by using a template already created.

I have put an allied destructible template [here](#) for you to download. Unzip it and put the 2 files into your maps/prefabs folder. Don't amend these, just use them as the template to stamp out allied destructibles as you need them.

Make a copy of [etmain/maps/prefabs/prefab\\_allied\\_destructible\\_template.map](#) and call it [etmain/maps/\\_temp.map](#). By calling the copy "\_temp" it will appear early in your list when you import it, and you know to chuck it away later on because it is "temp".

Edit \_temp.map with Wordpad or similar. Make these changes:

- Replace all instances of "allied\_destructible" with "gate" as this is the name we'll use for the gate destructible. You could of course use any name that suits you.
- Replace all instances of "Allied Destructible" with the name to appear on the Command Map and next to "you are near...". In this instance we're making a big gate, so make it "Main Gate".
- Save and close the file.

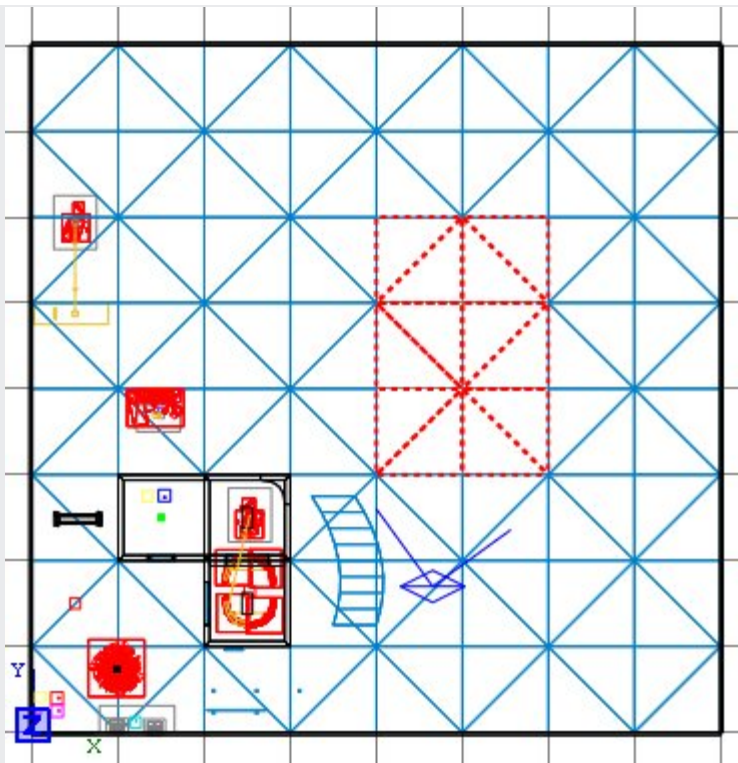
### Making the destructible

[\[Top\]](#)

Run Radiant and open the map. We're going to put the destructible gate into some clear space.

Select grid size 9. Get the 2D view over the clear space and delete the few bits of terrain as shown.



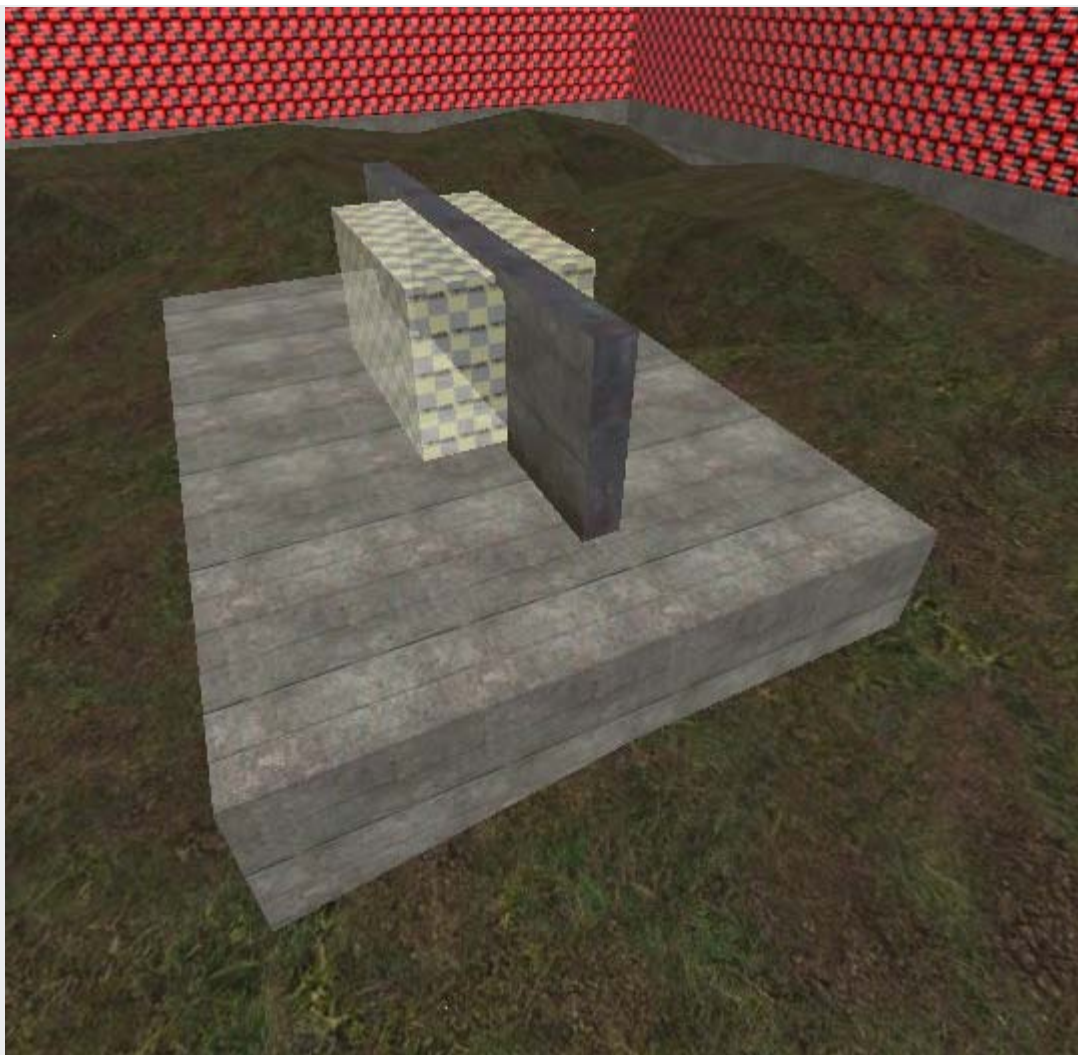


Now draw a brush in the space we've made, make it wall03\_mid texture, and make it Detail. Then select it again, lift it up so you can see the bottom and caulk the bottom face (Tip: select the face to be caulked, then also select a nearby caulked face - this will bring the caulk texture into view in the textures window, ready for your quick selection.)

Then move the brush back down into the gap, and ensure it neatly fits with no gaps showing around the sides where the terrain brushes meet it.



We will put the gate to be destroyed on this nice flat surface. Import \_temp.map and move the imported selection onto the middle of the flat slab.



Now to explain what we have here.

Shift+alt+click the trigger brush, and press N. Close the window and press N again so we see the tick boxes.

<input checked="" type="checkbox"/> axis_objective	<input type="checkbox"/> allied_objective	<input type="checkbox"/> mobile_tank	<input type="checkbox"/> is_objective
<input type="checkbox"/> is_healthammocabinet	<input type="checkbox"/> is_commandpost		

```

scriptname gate_toi
targetname gate_toi
target      gate
objflags    4
spawnflags  1
track       the Main Gate
classname   trigger_objective_info
shortname   Main Gate
    
```

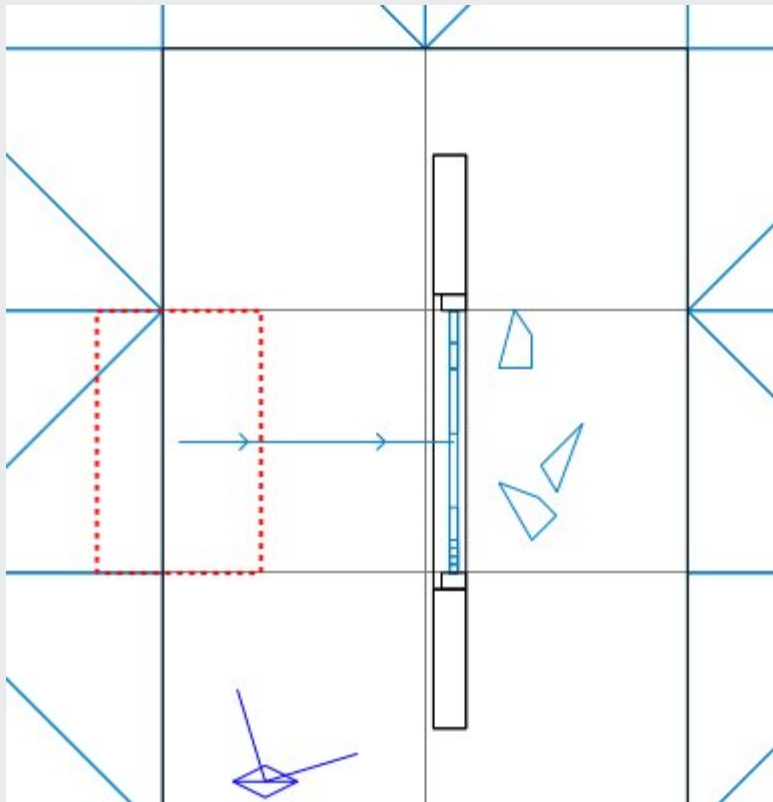
This is an **axis\_objective** because it is owned by the axis team. It seems to me that something to be destroyed by the allies ought to signify an allied objective, which is exactly why you should use a prefab as it is easy to tick the wrong boxes when creating one from scratch.

The **targetname** and **scriptname** are gate\_toi, being the name of the destructible plus "\_toi" to indicate the trigger\_objective\_info. This is not a universal naming standard, it's just one I adopted for myself so I could



easily identify an entity type by the type of name it had.

The **target** is the name of the destructible entity - you can tell if you've made a naming error because the trigger brush won't have an arrow line connecting it to the target brush(es). This line can't really be seen when the target brush is inside the trigger, which it normally would be because the trigger indicates where the attacker must plant dyna or chuck the satchel. So to see it, select grid scale 9 and drag the trigger brush away a notch.



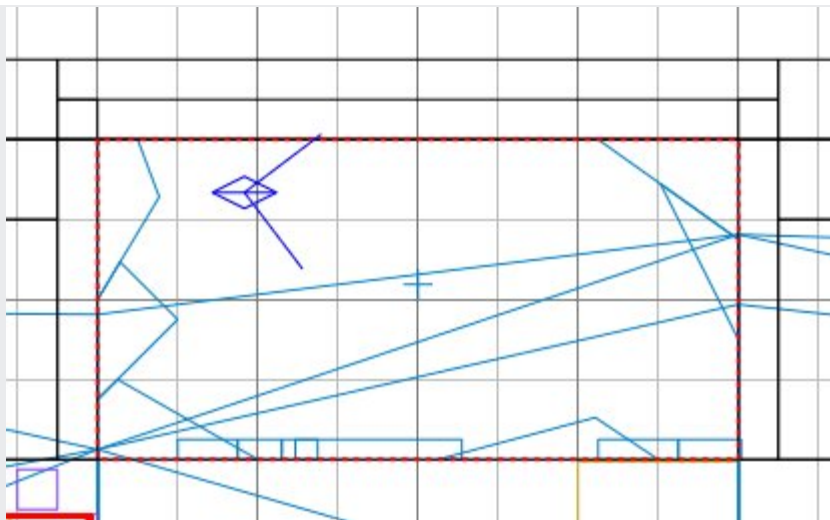
Now you can see the arrow connecting line, so the names are ok. Put the brush back over the gate.

The **objflags** is set to 4, telling ET to show a dyna symbol when a player enters the brush. I haven't seen a full list of possible values and I don't remember where I read even the partial list that told me to use 4 for dyna, but there will be a different value for different symbols. The only other one you might want really is the satchel symbol. Experiment with powers of 2 (1, 2, 4, 8 etc) to discover what the symbols are if you need symbols other than dyna.

The **spawnflags** is set to 1, which is a reflection of the tick box settings we have. The other values are obvious.

Hide the trigger so we can examine the other components.

Shift+alt+click the middle of the gate until the whole rectangle is selected (I'm using grid scale 6 again now).



This is the thing we want to blow up. Press N.

destroyed

☐ start\_invis ☐ touchable ☒ useshader ☐ lowgrav  
☐ tank

mass 600  
 targetname gate  
 type metal  
 scriptname gate  
 spawnflags 4  
 classname func\_explosive

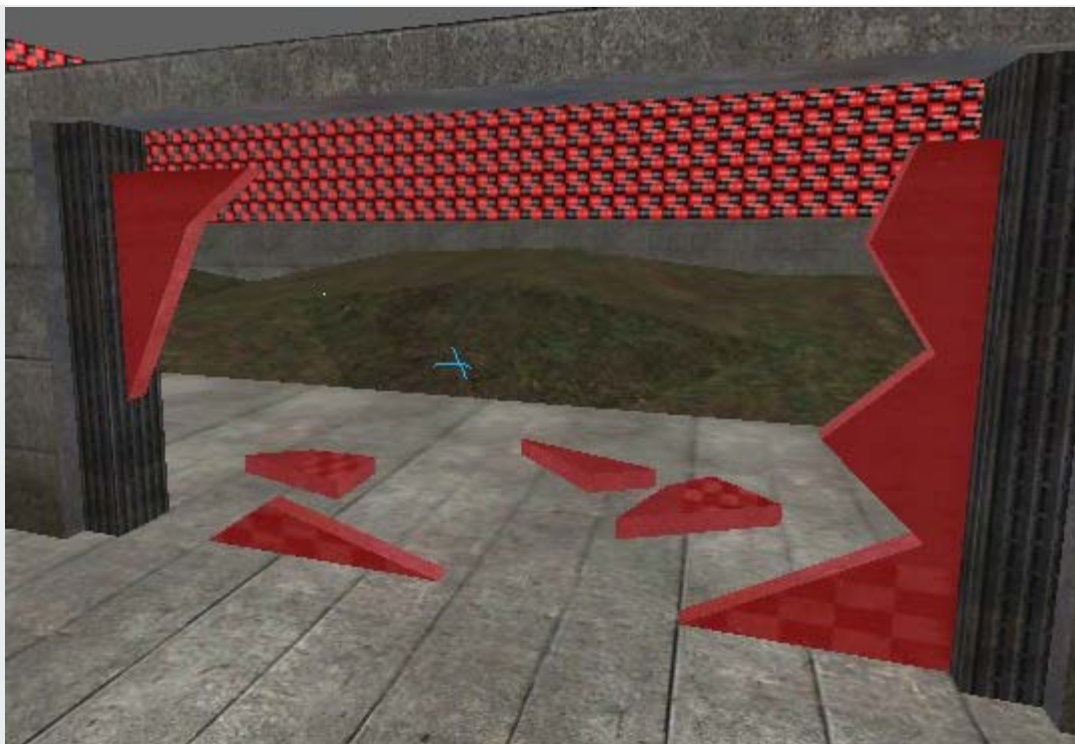
We have the **useshader** ticked so the temporary fragments are textured nicely.

The **mass** tells ET how chunky to make the bits that fly away from the bang. These bits will all disappear after a short time.

The **targetname** and **scriptname** are "gate", which will be referenced in the script.

Hide the gate. Now you can see the remnants that will be permanently shown after the explosion. They won't be visible before the explosion because they will be invisible when the map starts. You don't have to have remnants, it's up to you. I include them here so you can see how they are made if you want them.

Shift+alt+click on a remnant brush and they will all get selected.



☒ start\_invis ☐ pain

classname func\_static

spawnflags 1

targetname gate\_bits

These bits were made by copying the gate brush (and then right-click and "Move into Worldspawn" so that the copy wasn't a destructible). The copy was then chopped up with the clipper tool. Most of the copy was deleted, some of the bits were left in place, and some scattered onto nearby ground (with the downward face caulked of course).

Then all the bits were selected and made into a **func\_static** which is a way of making some brushes into an entity so you can refer to that entity. We will need an entity name because we will want to make these bits visible after the explosion.

The **start\_invis** box is ticked. Makes sense. I gave the entity a **targetname** of gate\_bits so I knew what I would be referring to in the script.

Reveal the hidden brushes, close the Info window if you have it open, save the map and compile it. Don't run ET yet.

## Writing the script

[\[Top\]](#)

Using Wordpad or similar, open the `prefab_allied_destructible_template.script` file. Copy all the text, and paste it into the `tutorial.script` file, at the bottom of the file. This is because we will name this section "gate", and so that will be the right place alphabetically.

Replace all 5 instances of **allied\_destructible** with **gate**.

Later on we'll add some speech but we'll skip that for now.

As the gate should be destroyable by dyna, change the `constructible_class` to 3.

As the script is simple I will explain what's happening as if we were writing it from scratch.

You start by declaring a procedure by entering its name. A procedure must be wholly enclosed by "{" and "}".

So you would begin writing the procedure by typing:

```
gate
{
}
```

Within the procedure you can have as many blocks of script as you need. These blocks are called **triggers**. There are several triggers with predefined names, such as **spawn** and **death**.

The **spawn** trigger is optional in any procedure. If included, it gets executed once-only, at the start of the map. You should always put a small delay in at the start of a spawn trigger, to allow all the spawning entities to arrive in the map before you might start trying to reference them.

First we'd add the spawn trigger. It is created inside the brackets of its parent procedure. I always add the "{" and "}" immediately I create a new trigger, so I don't accidentally leave out a closing "}". If you do, it can create some interesting errors when the map loads and you'll wonder what the hell is wrong.

```
gate
{
    spawn
    {
    }
}
```

We need a spawn trigger because we have to tell ET what weapons can destroy this destructible. So we add this info to the spawn trigger, following a wait of 300 milliseconds. A number between 50 and 500 is usual, with the choice being fairly random really. Two slashes "/" mean the rest of the line is just comments and not script.

```
gate
{
    spawn
    {
        wait 300
        constructible_class 3 // 2=satchel 3=dyna
    }
}
```

We also need a **death** trigger, to tell ET what to do when the gate is destroyed. The death trigger is also placed within the "{" and "}" of its parent gate procedure, not within the brackets of the spawn trigger.

```
gate
{
    spawn
    {
        wait 300
        constructible_class 3 // 2=satchel 3=dyna
    }
}
```

```

    }
    death
    {
    }
}

```

This is what we want to happen when the gate is destroyed:

- Hide the original undamaged gate - this is automatically done and we don't have to script anything
- Show the damaged gate bits - **alertentity gate\_bits** : alertentity toggles the visibility of the gate\_bits entity, ie reveals it in this instance
- Remove the "You are near..." prompt - **trigger gate\_toi remove** : which means execute the **remove** trigger in the **gate\_toi** procedure
- Tell the players that the gate has been blown up - **wm\_announce "The Allies have destroyed the gate!"**

```

gate
{
    spawn
    {
        wait 300
        constructible_class 3 // 2=satchel 3=dyna
    }
    death
    {
        alertentity gate_bits
        trigger gate_toi remove

        wm_announce "The Allies have destroyed the gate!"
    }
}

```

Finally we have to create a procedure for the trigger\_objective\_info entity, so that it can be removed when the gate is destroyed.

```

gate
{
    spawn
    {
        wait 300
        constructible_class 3 // 2=satchel 3=dyna
    }
    death
    {
        alertentity gate_bits
    }
}

```

```

        trigger gate_toi remove

        wm_announce "The Allies have destroyed the gate!"
    }
}

gate_toi
{
    trigger remove
    {
        remove
    }
}

```

The trigger called "remove" is not a standard ET provision, so we have to show that it is a user-defined trigger, ie made up by us, by prefixing it with the word "trigger".

The "remove" instruction within the trigger tells ET to delete this brush from the game as we no longer need it at all - which is why we don't just hide it.

Save and close the script file.

Run ET and have fun destroying the gate :)

I'll get some Axis templates together for you shortly.



You may now have visions of making a map with dozens of constructibles and destructibles. Unfortunately there is a limit of 18 trigger\_objective\_info entities in a map. And it's easier than you might think to eat away at that total amount, as TOI entities get used for other main game elements too. Consider Fueldump, and count the TOIs:

1. Tank
2. Ammo cabinet in the Allied shack
3. Allied MG42 outside the shack
4. Bridge
5. Footbridge
6. Axis CP
7. Axis MG42 tower overlooking bridge
8. Allied CP
9. Ammo cabinet near allied CP
10. Axis MG42 tower near allied CP
11. Allied MG42 near allied CP
12. West fence
13. East fence
14. Fuel dump

[Next lesson](#)





# ET Mapping Tutorial

## Lesson 29

### Topics

#### Forward Spawn Flags

[Forward Spawn Flags](#)

[Scripting](#)

[Back to main menu](#)

#### Forward Spawn Flags

[\[Top\]](#)

These are quite easy to do, but you need to be careful with the script to make sure they will perform the right way for the situation you want.

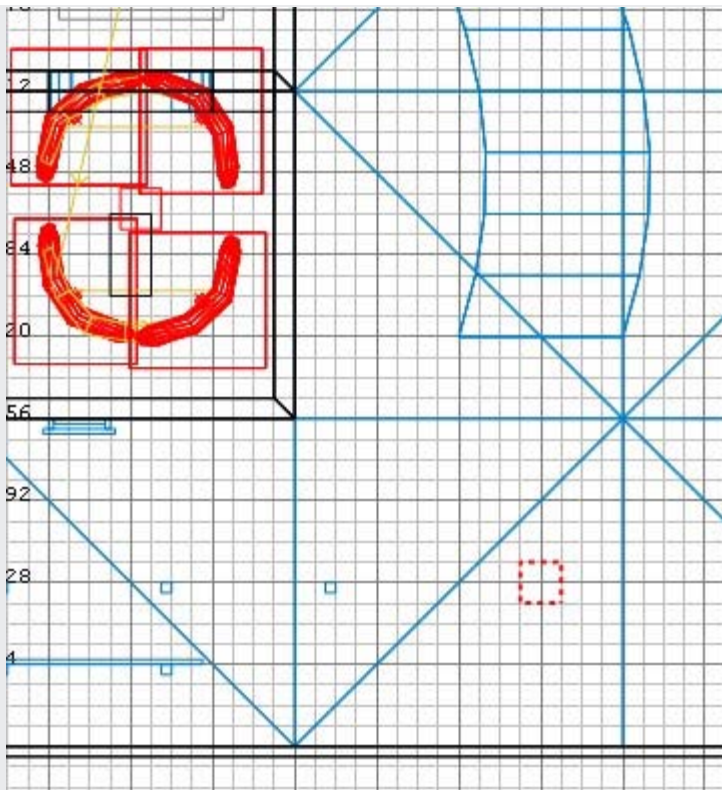
There are a few varieties of behaviour, such as:

- the flag is owned by no-one at the start
- the flag is owned by a team at the start but they can't spawn there
- the flag is owned by a team at the start and they can spawn there but it's not the default
- when the flag is captured, it becomes the default spawn for the capturing team
- when the flag is captured, it *doesn't* become the default spawn for the capturing team

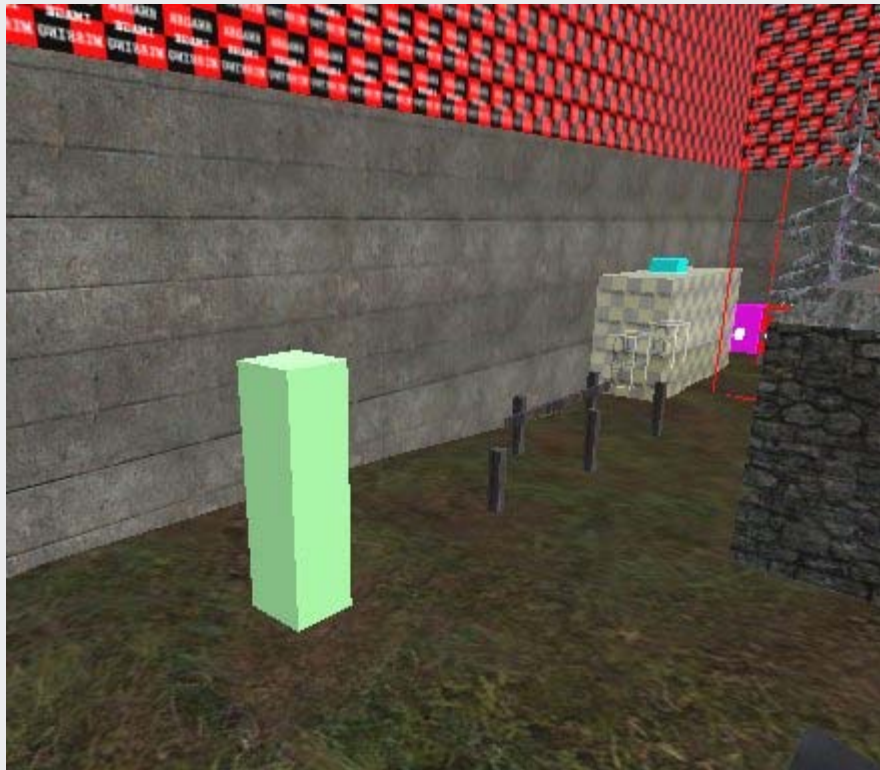
All variations on a theme. The one I'll demonstrate is the flag owned by no-one, which will become the default spawn for the capturing team. The losing team will then spawn back at their original spawn point.

Run Radiant and open the map. Don't forget to make backups from time to time.

We'll start by putting the flag down. In the 2D view, right click at the point shown and select **team/team\_wolf\_checkpoint**.



In the side 2D view, move the entity until its bottom just about goes into the ground. Be aware that if you put a flag on a rooftop, the rolled up flag of the losing team will appear below the bottom of the flagpole, which might appear to be hanging from the ceiling of the room below...



Press N. Close the window and press N again.

Tick the **spawnpoint** box.

We need to provide a name for the entity so we can have a script procedure for it. Enter a **targetname** of **forward\_flag** and then a **scriptname** also of **forward\_flag**.

The name can be anything of course, but we'll use `forward_flag` for now. This value is not shown to players.

Finally we need to tell the flag the names of the spawn locations that it is controlling: enter a **target** of **forward\_spawns**.

```
spawnpoints for opposing team
*****
[checked] spawnpoint [ ] cp_hold [ ] axis_only [ ] allied_only
target      forward_spawns
scriptname  forward_flag
targetname  forward_flag
spawnflags  1
origin      960 128 0
classname  team_WOLF_checkpoint
```

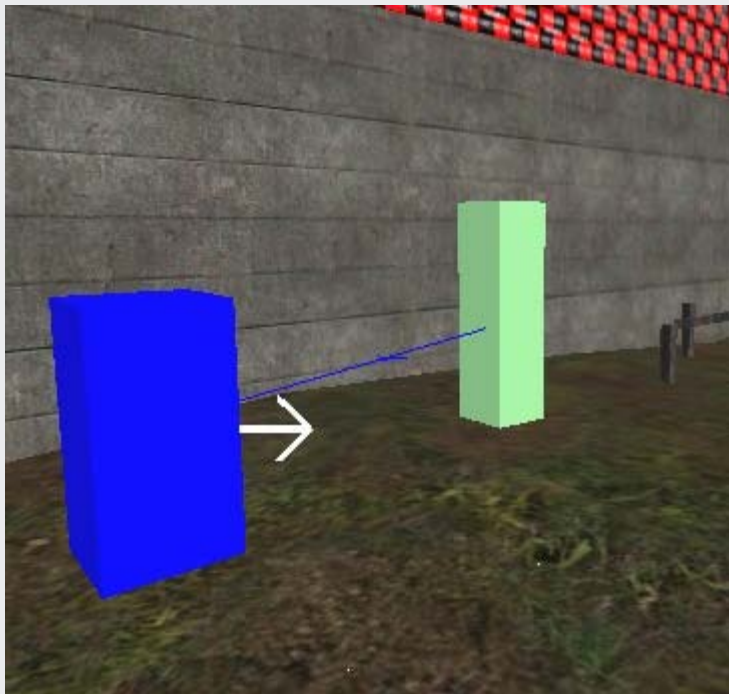
Ok, now we'll add the spawn points for both teams. We'll add just one of each to illustrate the technique - but you would need to add 32 for each side.

Right-click in the 2D overhead view where you want the first allied soldier to arrive, and select **team/team\_ctf\_bluespawn**. In the side view make sure the entity is on or just above the ground. For irregular surfaces like terrain, make it just above, otherwise he'll start with his feet in the ground and not be able to move.

Press N, and tick the **invulnerable** box. I think this may be redundant but what the heck. We won't tick the **startactive** box because we don't want him spawning here yet.

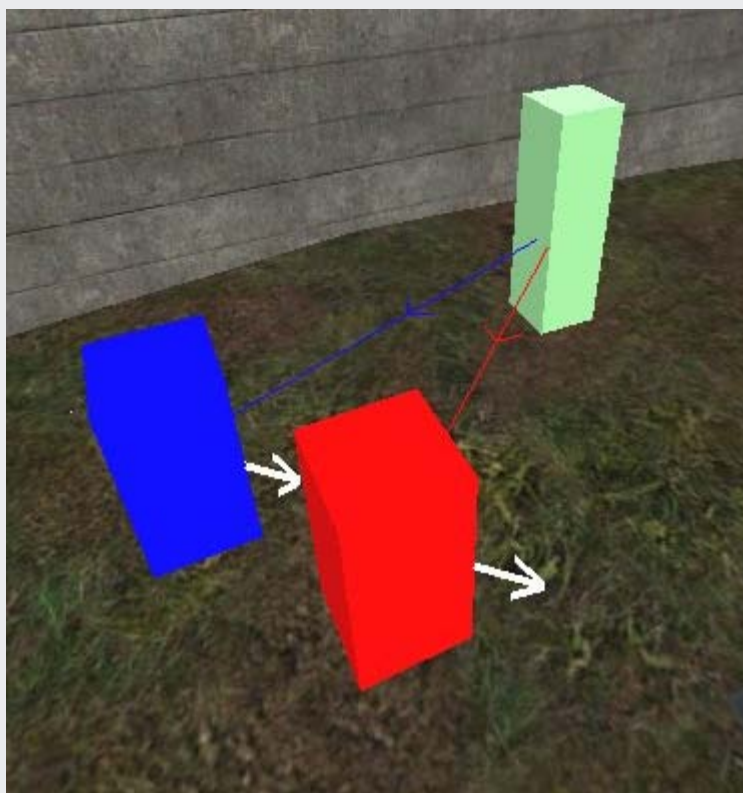
Click on one of the 8 directional buttons at bottom left of the Entities window, to set the direction the player will be looking in on spawning.

Enter a **targetname** of **forward\_spawns** and close the entities window. If you have done everything right, the flag will now have a line drawn to the blue spawn entity. Press ESC.



Now right-click in the 2D overhead view where you want the Axis soldier to spawn. If space is tight, the axis spawns can overlap the allied spawns. Select **team/team\_ctf\_redspawn** and position it clear of the ground and press N.

Now do the same things that you did for the blue spawn, that is, set the **invulnerable** box and give it the same **targetname**, and any facing direction you want. Close the window and press ESC.

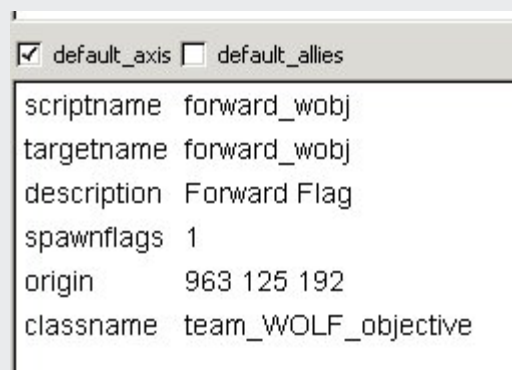


You should have a red line connecting the flag to the spawn.

Finally we need to add the command map marker entity to show where the flag is. Usually you put this over the flag entity. So right-click on the flag entity in the overhead view and select **team/team\_wolf\_objective**. Position the entity a little over the flag.

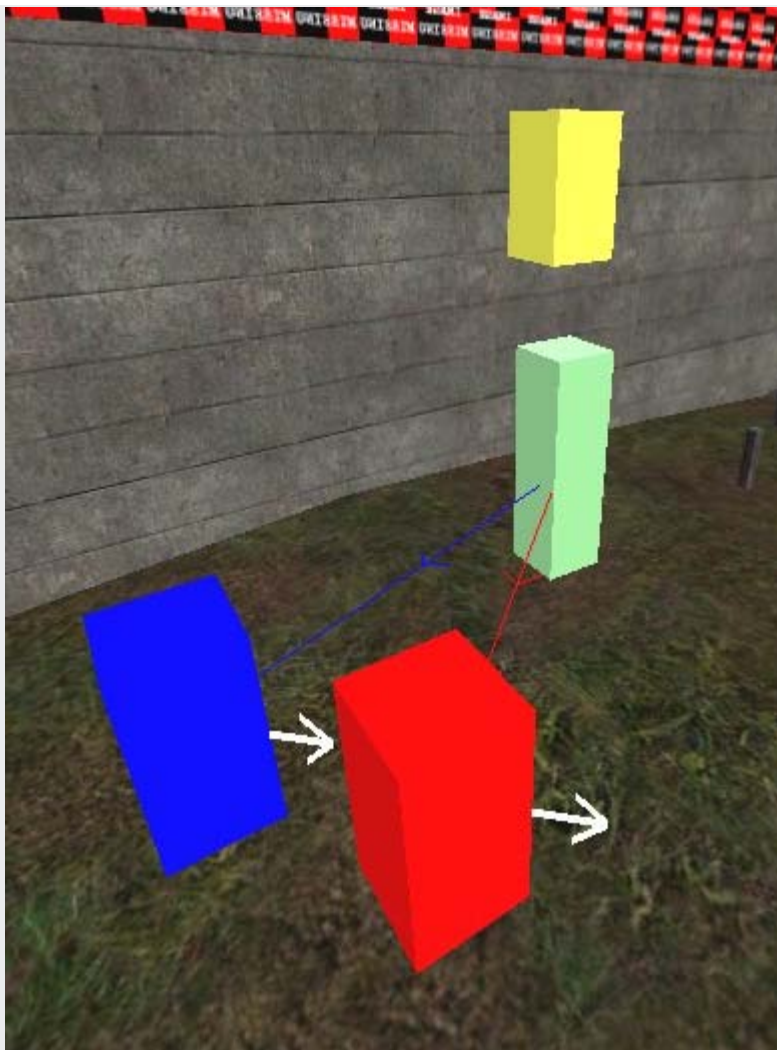
Press N. There has to be a default owner, even if really there is none. So tick the **default\_axis** box. We'll take care of the real situation in the script.

Enter a **description** of **Forward Flag** - this text will be shown on the command map. Enter a **targetname** and **scriptname** of **forward\_wobj**.



Close the window and press ESC. We have completed the mapping element.





Save the map and compile it. Don't run ET yet.

## Scripting

[\[Top\]](#)

Open tutorial.script, and enter the following script just before the **gate** procedure, as this procedure will start with "F".

**forward\_flag**

```
{
}
```

Then inside the curly brackets, put in the following text. I suggest you cut and paste it from here, but you ought to add the indentation manually so you can see where everything belongs:

```
spawn
{
    accum 0 set 2 // Who owns flag: 0-Axis, 1-Allied, 2-Nobody
}
trigger axis_capture // Touched by an Axis player
{
    accum 0 abort_if_equal 0 // do Axis own flag?
```

```

accum 0 trigger_if_equal 1 forward_flag axis_reclaim // Reclaimed from Allies

accum 0 set 0 // Axis own the flag

wm_announce "Axis have captured the Forward Flag!"

setstate forward_wobj default
}

trigger axis_reclaim
{
    alertentity forward_wobj // Switch command map marker
}

trigger allied_capture // Touched by an allied player
{
    accum 0 abort_if_equal 1 // do Allies own flag?

    accum 0 set 1 // Allied own the flag

    wm_announce "Allies have captured the Forward Flag!"

    setstate forward_wobj default

    alertentity forward_wobj // Switch command map marker
}

```

It should look like this in layout:

```

// =====
forward_flag
{
    spawn
    {
        accum 0 set 2 // Who owns flag: 0-Axis, 1-Allied, 2-Nobody
    }

    trigger axis_capture // Touched by an Axis player
    {
        accum 0 abort_if_equal 0 // do Axis own flag?
        accum 0 trigger_if_equal 1 forward_flag axis_reclaim // Reclaimed from Allies

        accum 0 set 0 // Axis own the flag
        wm_announce "Axis have captured the Forward Flag!"

        setstate forward_wobj default
    }

    trigger axis_reclaim
    {
        alertentity forward_wobj // Switch command map marker
    }

    trigger allied_capture // Touched by an allied player
    {
        accum 0 abort_if_equal 1 // do Allies own flag?

        accum 0 set 1 // Allied own the flag
        wm_announce "Allies have captured the Forward Flag!"

        setstate forward_wobj default
        alertentity forward_wobj // Switch command map marker
    }
}

```



Finally you'll need to add this line into your `game_manager` procedure (so that the flag doesn't initially show on the command map)

### setstate forward\_wobj invisible

...and change the autospawn locations to "Forward Flag" so that both teams will automatically spawn at the forward flag, if it is available to them. If not, the players will spawn back at the original spawn.

So your `game_manager` ought to look like:

```
game_manager
{
    spawn
    {
        wm_axis_respawntime      10
        wm_allied_respawntime    10
        wm_set_round_timelimit   30

        // Stopwatch mode defending team (0=Axis, 1=Allies)
        wm_set_defending_team    0

        // Winner on expiration of round timer (0=Axis, 1=Allies, -1=Nobody)
        wm_setwinner 0

        wait 500

        setstate forward_wobj invisible

        setautospawn "Forward Flag" 0
        setautospawn "Forward Flag" 1
    }
}
```

You can now have a go in ET and delight in your forward spawning opportunities :)

You may find the flying flag is a bit dark. You can either add a little light entity over or near it, or experiment with changing the direction the flag flies in - just set the **angle** of the **team\_wolf\_checkpoint** (flag entity).

If you are not interested in what the script is doing, skip to the next lesson now. Otherwise keep reading, as this starts to get us into the scripting fundamentals.

### Explanation of the script

On game start, the flag's **spawn** trigger is executed. It sets **accum 0** to **2**. An "accum" (short for accumulator, a term used in assembly programming) is a variable, in which you can store integers.



Each procedure can use up to 10 accums, numbered 0 to 9. The values stored in these procedural accums are not accessible to other procedures. If you need to have values accessible across procedures, you use a **globalaccum** which are also numbered 0 to 9. A map can only have a maximum of 10 globalaccums.

Accum 0 is used by this procedure to indicate the condition that the flag is in, ie who owns it. It uses the value 2 to indicate "no owner".

The **allied\_capture** and **axis\_capture** triggers are predefined names, and the relevant trigger is executed when a player touches the flag (even if his team already own the flag).

We'll look at **allied\_capture** first:

When an allied soldier touches the flag, the trigger is executed. The first thing it does is check the value of accum 0.

If the value in accum 0 is 1 it indicates that the allies already own the flag, so the code is aborted and nothing else happens.

Otherwise the allies have captured the flag, so accum 0 is set to 1 to record this.

Then the **wm\_announce** is executed to tell the players about this event.

Now that the flag has been captured, the allies can spawn at it, so it must appear on the command map: **setstate forward\_wobj default** means makes the command map icon visible.

Finally, **alertentity forward\_wobj** toggles the owner of the spawnpoint, from Axis to Allied.

Now look at **axis\_capture**.

When an axis soldier touches the flag, the trigger is executed. The first thing it does is check the value of accum 0.

If the value in accum 0 is 0 it indicates that the axis already own the flag, so the code is aborted and nothing else happens.

Then it checks to see if the flag was previously owned by Allies - this will always be the case after the allies have captured the flag for the first time. But if the allies are yet to capture the flag, the **default\_axis** setting means the axis are deemed to already be the owners, so we wouldn't want to toggle the owner away.

So if the allies *were* the owners, the **trigger axis\_reclaim** trigger is executed (this is a routine I created, not an ET pre-supplied one). Its function is just to toggle the owner back to axis.

Then just like the allied routine, the new owner is recorded, an announcement made, and the command map icon made visible.

That's it.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 30

### Topics

#### Water

[Water](#)

[Back to main menu](#)

#### Water

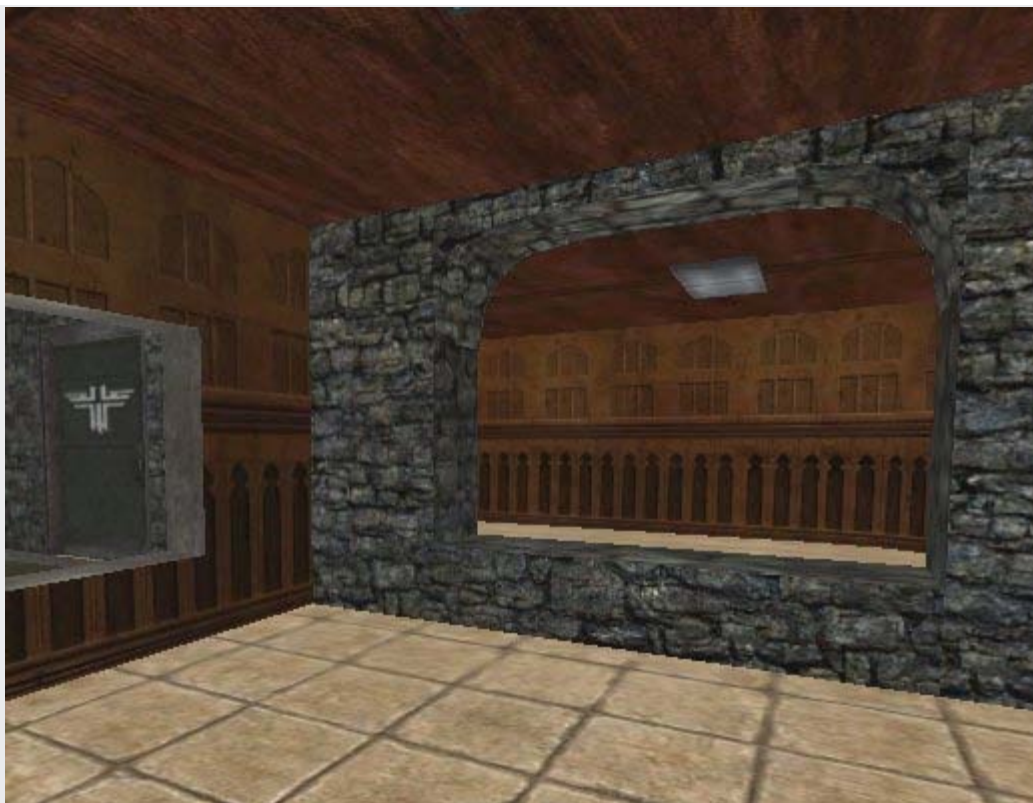
[\[Top\]](#)

Run Radiant and open the map.

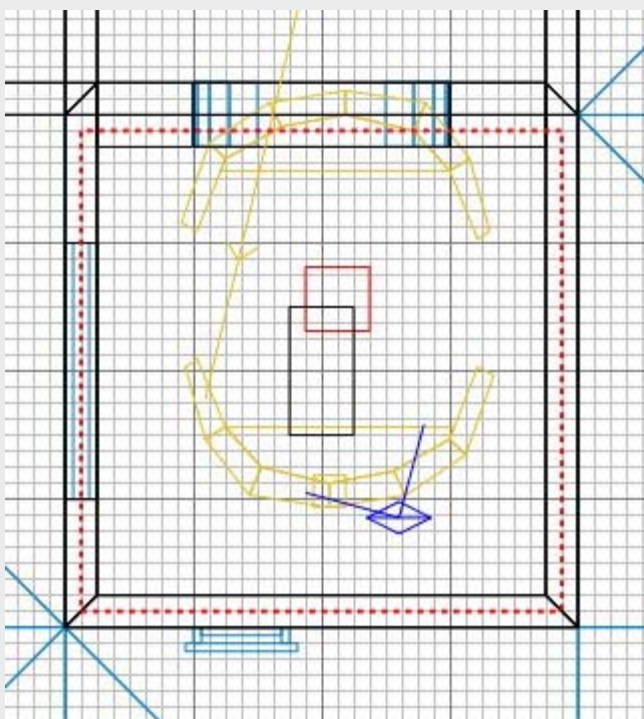
Unlike the usual setup, in which players run around in the volume space *between* all the brushes of your map, with water, players splash/swim about *inside* the brush.

To illustrate this, we'll flood one end of the little building we've made.

Block off one end of the room by making a low wall as shown. It's caulked and been made Detail, and had some brick texture applied.



Then create a brush that is just a little bit lower than this new wall, but has edges that extend a little way *into* the floor and *into* the surrounding walls, rather than meeting them flush. Use grid scale 4 to do this. I have filtered models to make the view clearer. See how the brush goes into its surrounding walls (and the ground, not visible in the overhead view).



Click **textures/liquids/liquids\_sd** and select the **siwa\_waternodraw** to texture the whole brush. Then select just the surface face of the brush and apply **siwa\_water**. Then press ESC.

Only the surface needs to be visibly textured with water, the other faces won't be visible, but they can't just be caulk etc, they have to be a water-type shader; the siwa\_waternodraw will do fine.

Save and compile the map, and go and splash around in it in ET. If it were deeper you would be able to

swim in it.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 31

### Topics

#### Team speech

[Adding speech to the script](#)

[Adding data to the sounds file](#)

[Back to main menu](#)

### Adding speech to the script

[\[Top\]](#)

To get speech to be broadcast to each team, like "Build the Command Post!", takes two elements: something in the script, and something in the sounds file. We'll address the scripting element first.

Open tutorial.script.

To make speech that is heard by all the players in a team, you use the **wm\_teamvoiceannounce** instruction.

We'll add speech that tells the allies to blow up the main gate, and the axis to prevent this.

Add these 2 lines to the end of the **spawn** trigger in the **game\_manager** procedure:

- `wm_teamvoiceannounce 0 "radar_axis_entrance1_stop"`
- `wm_teamvoiceannounce 1 "radar_allies_entrance1_destroy"`

The "0" means that the Axis team will hear the speech; the "1" means the Allies will hear it.

The use of the word "radar" indicates that the speech comes from the Radar map. Generally you use "axis" in the speech name so that it is clear to you that the speech is with a german accent, and similarly you use "allies" for american speech.

"entrance1\_destroy" and "entrance1\_stop" then make it clear what the speech says.

The "radar\_axis\_entrance1\_stop" tells ET to look up this reference in the sounds file, and play the WAV associated with it. Same for "radar\_allies\_entrance1\_destroy".

As soon as the **wm\_teamvoiceannounce** instruction is executed, the speech is queued up to be spoken, that is, if there is already speech being spoken, they won't shout each other down or interrupt each other, they just queue up.

But you will sometimes want certain speech to be spoken as soon as a player joins a team, even if it is after the start of the game.



So add these lines after the other two:

- `wm_addteamvoiceannounce 0 "radar_axis_entrance1_stop"`
- `wm_addteamvoiceannounce 1 "radar_allies_entrance1_destroy"`

**wm\_addteamvoiceannounce** means "don't speak this now, but speak it to any players who join the team later".

The last bits to go into the script are the instructions to stop telling newly arrived players to destroy the main gate, *if* the main gate has already been destroyed.

Add these lines to the **gate** procedure at the end of the **death** trigger:

- `wm_removeteamvoiceannounce 0 "radar_axis_entrance1_stop"`
- `wm_removeteamvoiceannounce 1 "radar_allies_entrance1_destroy"`

When the gate is destroyed, and its "death" trigger is executed, ET will now remove these speech elements from the list of speeches to play to newly arriving players.

## Adding data to the sounds file

[\[Top\]](#)

You will need to create a folder called **scripts** inside the `etmain/sounds` folder, if it isn't already there.

Now create a text file called `etmain/sounds/scripts/tutorial.sounds`.

Add this text to the new file:

```
radar_allies_entrance1_destroy
{
    sound sound/vo/radar/allies/hq_entrance1dyn.wav
    voice
    streaming
}

radar_axis_entrance1_stop
{
    sound sound/vo/radar/axis/hq_entrance1stop.wav
    voice
    streaming
}
```

As you can see, you will need a definition with the same name as that specified in each **wm\_teamvoiceannounce** instruction.

The **sound** line tells ET which WAV to play.

It is always **voice** and **streaming**. If you have additional **sound** lines then ET will play one or the other, giving you some variety in intonation.

You can look into the `.sounds` files of other maps to get ideas for the sort of speech other people have used - the WAVs don't have to be those that came with ET; you can create your own WAVs and have them spoken in just the same way.

I prefer not to put on a fake german accent for the speech I need, if there isn't one already supplied by ET - instead I found it better to cut up existing WAVs using a WAV editor and then put the words together in the

construct that I wanted. For 2tanks I had to assemble 53 new WAV files for the various team speech, and it was all done by using words lifted from the various standard ET maps.

With the script and sounds files now ready, you can run ET and you should hear the "destroy the main entrance" etc on arrival - but not after the gate has been blown up.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 32

### Topics

#### Limbo camera and objectives narrative

[Placing limbo cameras](#)

[Scripting the objectives](#)

[Writing the objective descriptions](#)

[Back to main menu](#)

### Placing limbo cameras

[\[Top\]](#)

We've reached the final set of lessons needed to reach the point where you can create and distribute a fully operational PK3.

This lesson covers the use of limbo cameras, which are the little (dynamic) images that accompany the objectives text on the limbo screen. With the views, those blocks of text, and the ticks and crosses you can superimpose on them, this part of the limbo screen tells the players what to do, the order to do it in and roughly where to do it.

I'm not actually sure how many players ever look at this information, whether they are brand new to the map and could use the instructions, or whether they are familiar with the map and need to know which objectives have been accomplished - but you have to provide this information or you get some ugly default views in the limbo camera section.

The components to providing limbo camera information are:

- the limbo cameras placed in the map
- the script which associates each camera to an objective
- the script which dynamically puts ticks and crosses on the objective text
- the descriptive text which accompanies the views

We'll start with the camera placement.

Run Radiant and open the map.



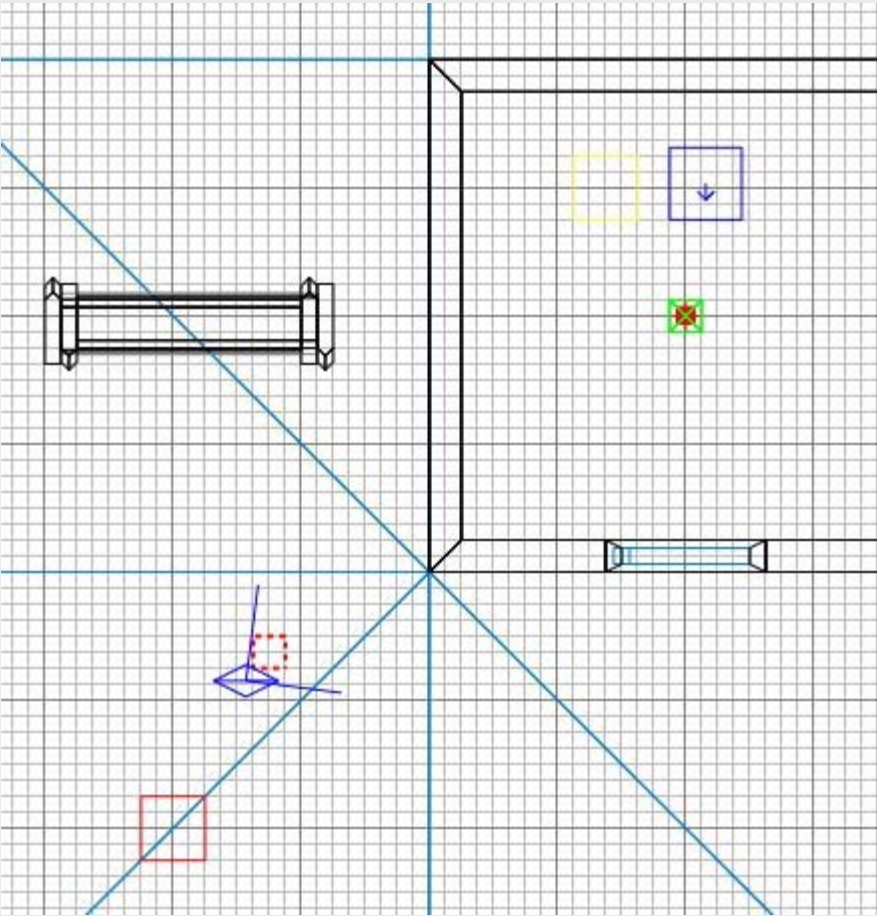
You are limited to 8 objectives. Along with the general introduction limbo camera view, this gives you a total of 9 limbo cameras available to you.

We will place 2 cameras and assume there is one objective - for the allies to destroy the main gate.

The first camera will show the general view which introduces the map. In the 3D view, place the view where you want the camera to be looking from, and aim the view in the direction you want it to be looking in.

In the 2D view, position the overhead view so you can see the little blue eyeball indicator. I've picked a general view, just above the roof height, that offers a far view of the gate.

Right-click just in front of the eyeball and select **info/info\_limbo\_camera**.



Now in the 2D view, adjust the height of the camera until it is just in front of the view in 3D.

Press N.

Enter **objective** and give it a value of **0**.

Enter **target** and a value of **limbo0**.

Press ESC.

We must now place an indicator over in front of the gate to tell the camera what to look at.

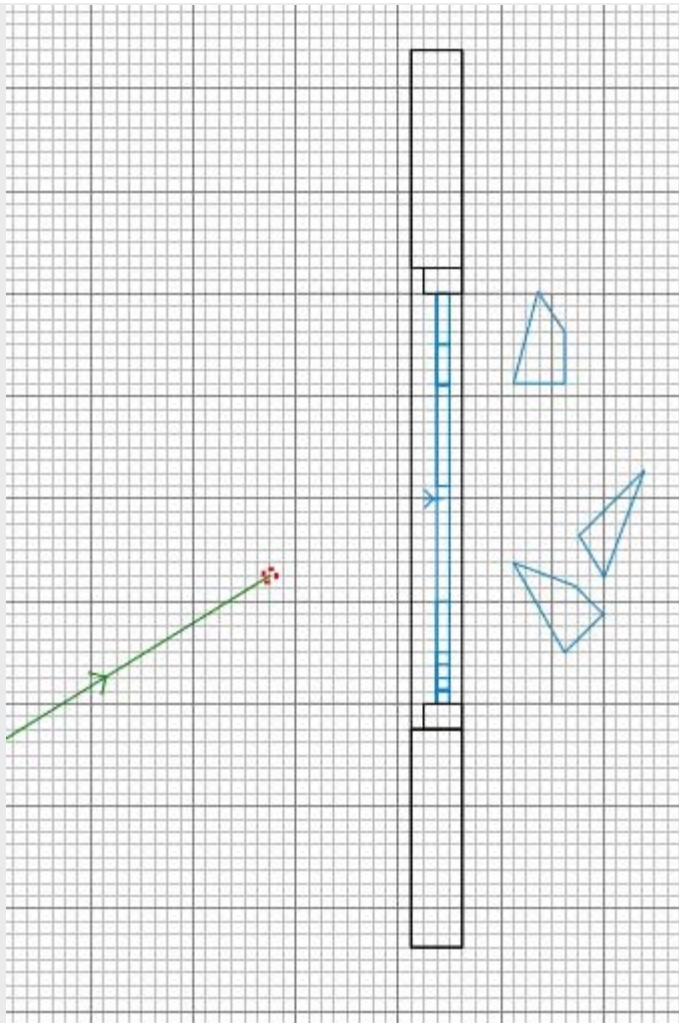
Right-click in front of the gate and select **info/info\_notnull**.

In the side view, move the info\_notnull to a height about the middle of the gate.

Press N.

Enter **targetname** with a value of **limbo0**.

If you have done this correctly, the camera will now join to the info\_notnull with a green line.



Press ESC.

Now we'll place the second camera - it will show a close-up view of the objective, ie the gate. Get a close view in 3D and place and position another limbo camera entity as you did before.

Press N.

Enter **objective** and give it a value of **1**. This is the first of a possible 8 objectives.

Enter **target** and a value of **limbo1**.

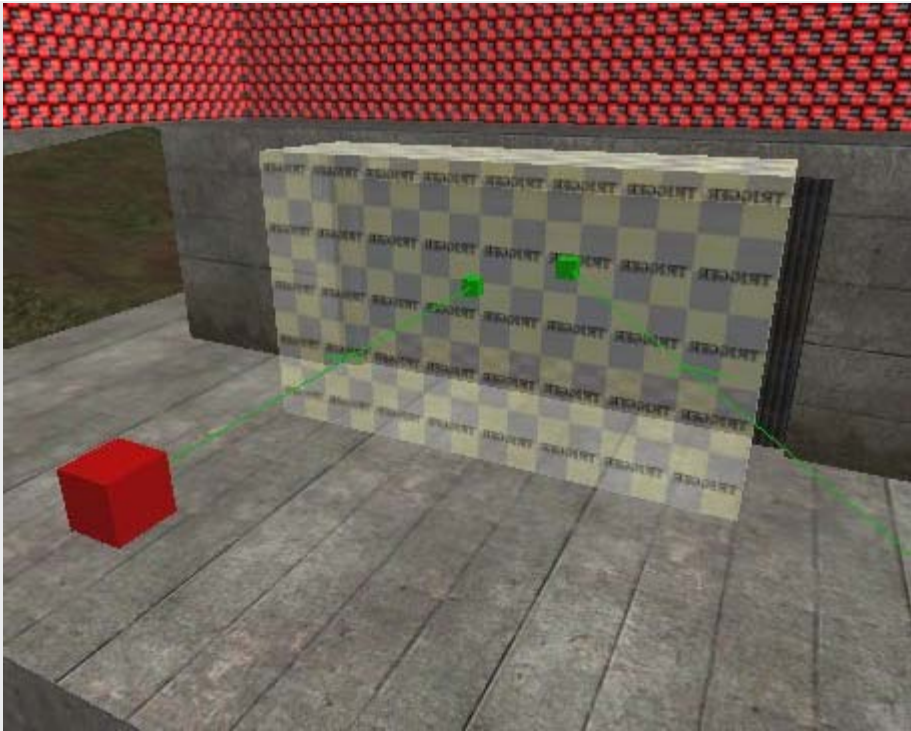
Press ESC.

Place another info\_notnull in front of the gate.

Press N.

Enter **targetname** with a value of **limbo1**.

If you have done this correctly, the camera will now join to the info\_notnull with a green line.



Save and compile the map. Don't run ET yet.

Remember that these are dynamic cameras, and will show the view at the moment the player is looking at it. Bear that in mind if your view looks wrong or inappropriate if the game circumstances change. Don't worry too much though, as I said, I doubt many people look at them :(

Scripting the objectives

[\[Top\]](#)

Open etmain/maps/tutorial.script.

Add this line into the **game\_manager spawn** trigger, after the line **wm\_set\_round\_timelimit**

- **wm\_number\_of\_objectives 1**

This number will be a value between 1 and 8, depending on the number of *declared* objectives. You can have more than 8 objectives, but you can only have limbo cameras for 8 of them.

Add these lines into the same trigger, after the **wait 500**:

- **// Objectives**
- **// 1 Primary Objective: Allies destroy main entrance**

These are just comments, to help remind you what the following instructions are actually referring to.

Add these lines below those just entered:

- **// obj nbr, team, status (0=none, 1=passed, 2=failed)**
- **wm\_objective\_status 1 0 0**
- **wm\_objective\_status 1 1 0**

These lines set the starting conditions for each objective. You would have 1 pair of these lines per objective. This is what the lines mean:

<b>wm_objective_status 1 0 0</b>	The instruction that sets the status for an objective
<b>wm_objective_status 1 0 0</b>	The objective number
<b>wm_objective_status 1 0 0</b>	The team number - 0=axis 1=allies



wm_objective_status 1 0 0	The status - 0=unmarked 1=ticked 2=crossed
---------------------------	--

Fiinally add these lines to the **gate death** trigger, it doesn't matter where precisely:

- wm\_objective\_status 1 0 2
- wm\_objective\_status 1 1 1

So when the gate is blown, the axis objective is marked with a cross, and the allied with a tick. If the objective you were marking in this way were say a flag, which could be expected to switch back and forth, you can repeatedly set the objective status to show a team a tick, then a cross, then a tick, etc.

That's the end of the script changes. To finish we now just need to create the text to go with the limbo views.

## Writing the objective descriptions

[\[Top\]](#)

The descriptions are written in a new text file.

Create a text file called etmain/maps/**tutorial.objdata**.

Edit the tutorial.objdata file, and add these lines to it:

```
// Set scenario information

wm_mapdescription allied "Destroy the main entrance!**Run through the opening and laugh madly!"

wm_mapdescription axis "Don't let them destroy the main entrance!"

wm_mapdescription neutral "The Allies must blow the main gate."

// Axis Objective Descriptions

wm_objective_axis_desc 1 "Primary Objective: **Don't let them destroy the main entrance.**This will lose
the forward bunker and give them access to the factory."

// Allied Objective Descriptions

wm_objective_allied_desc 1 "Primary Objective: **Destroy the main entrance.**This will capture the forward
bunker and give you access to the factory."
```

The first three lines of text give the general map objectives display to spectators, axis players and allied players, before they look at any of the objective limbo camera views. In limbo this will appear as objective 1 of 2.

The axis and allied specific descriptions are numbered **1**, and you'd have one line per objective, obviously numbered 2, 3, 4, etc. Make sure these match up with what you do in the script! I have used the text from 110 Factory to illustrate the sort of thing you might put.

Generally you should make **Primary Objective** those objectives that are critical to game success, and the others merely **Secondary Objective**.

You can use **\*** to force a new line, and **\*\*** to force a new paragraph.

Run ET and revel in your limbo views and objective text :)

You'll most likely find that some views are slightly wrong or poorly illuminated, so it is an iterative exercise of tweaking camera positions until you are happy with all of them.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 33

### Topics

#### Making the game end

[Making the game end](#)

[Back to main menu](#)

#### Making the game end

[\[Top\]](#)

The map will end when a winning team has been specified in the script, and the game ending conditions have been met.

The instruction **wm\_setwinner** is used to specify the winning team:

- -1 = No winning team specified yet
- 0 = Axis win
- 1 = Allies win

In the tutorial script, we have set **wm\_setwinner** to 0 right at the start, so that if the game time expires, Axis will win.

What we will do is add some script to make the allies win if they blow the gate.

Open the tutorial.script file, and add this line at the end of the **gate death** trigger:

- trigger game\_manager allies\_win

So when the gate is blown up, the last thing the death trigger does is call (execute) the **allies\_win** trigger in the **game\_manager** procedure.

Add this script after the closing "}" of the spawn trigger in the game\_manager procedure, but before the closing "}" of the game\_manager itself.:

```
trigger allies_win
{
    wm_announce "The Allies blew up the gate!"
    wm_setwinner 1
    wait 3000
}
```

```
wm_endround
```

```
}
```

So if the allies succeed, an announcement is made, `wm_setwinner` is changed to 1, a 3 second delay to give people time to see it, and then the **`wm_endround`** instruction is executed, which forces the game to end now.

As you can see, there is no provision for a draw: `wm_setwinner` is either 0 or 1 when the game ends. If the timer expires and `wm_setwinner` is -1, then the game continues until you set it to 0 or 1, at which time the game immediately ends.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 34

### Topics

#### Generating a tracemap

[Generating a tracemap](#)

[Back to main menu](#)

#### Generating a tracemap

[\[Top\]](#)

A tracemap is not strictly necessary, which is why sometimes the mapper omits one and you might notice an error like "tracemap not found" fly past in all the loading text before the map starts.

But really you should do one. A tracemap is generated by ET when in developer mode, and once generated, it can be used to tell ET about the contours and geography of the map so that it knows how to apply weather effects.

Without a tracemap, if you made your map snow, the snow would fall into the buildings.

Run ET, and run your map.

Once the warmup has ended, bring up the console and type:

- \developer 1
- \generatetracemap

You'll then see a lot of lines telling you how the generation is progressing. When it's all finished, type in:

- \developer 0

Don't worry about other sundry messages that might get displayed, they don't matter. You can quit ET again now.

The tracemap file will actually come in handy for the next lesson, which is Making a Command Map...

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 35

### Topics

#### Making the command map

[Making the command map](#)

[Back to main menu](#)

### Making the command map

[\[Top\]](#)

You'll need an image editor for this.

First of all though, we'll create a new shader file which is also required.

Create a text file: etmain/scripts/**tutorial\_levelshots.shader**

Put these lines in it:

```
levelshots/tutorial_cc_automap
{
    nopicmip
    nocompress
    nomipmaps
    {
        clampmap levelshots/tutorial_cc.tga
        depthFunc equal
        rgbGen identity
    }
}
levelshots/tutorial_cc_trans
{
    nopicmip
```

```

nocompress
nomipmaps
{
    clampmap levelshots/tutorial_cc.tga
    blendfunc blend
    rgbGen identity
    alphaGen vertex
}
}

```

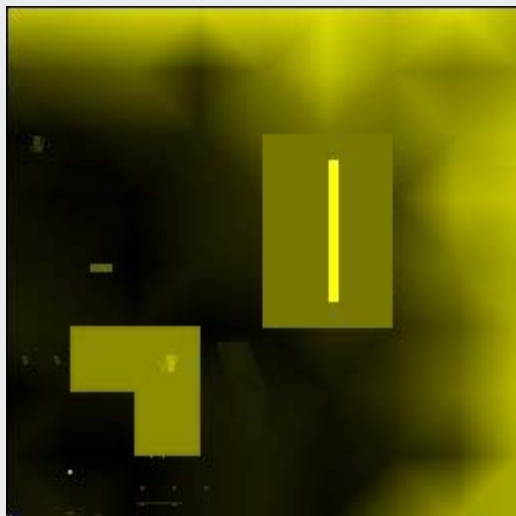
When making this file for your own maps, change all instances of "tutorial" to whatever map name you are using.

I use Paint Shop Pro for image editing, so I will explain the next step using that tool. Your own image editor will no doubt offer the same features but maybe under different names.

Run PSP or equivalent.

Open **tutorial\_tracemap.tga** which will be in the **maps** folder.

This is what mine looks like:



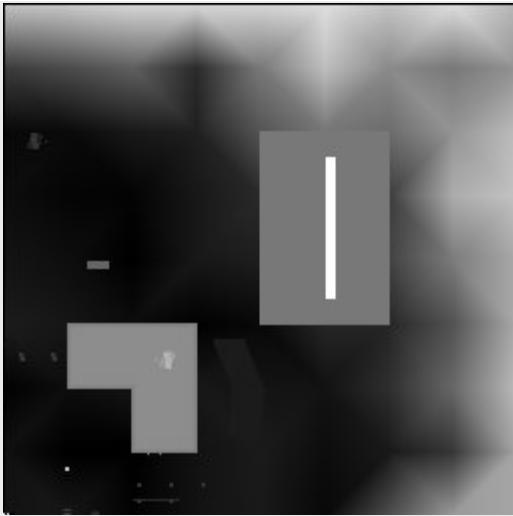
You can see the building and the wall on the slab, and the rest is pretty much smudgy.

We want to split the image into its 3 constituent RGB (Red/Green/Blue) channels - one of which might give us a better picture.

In PSP, click **colors/split channel/split to RGB**

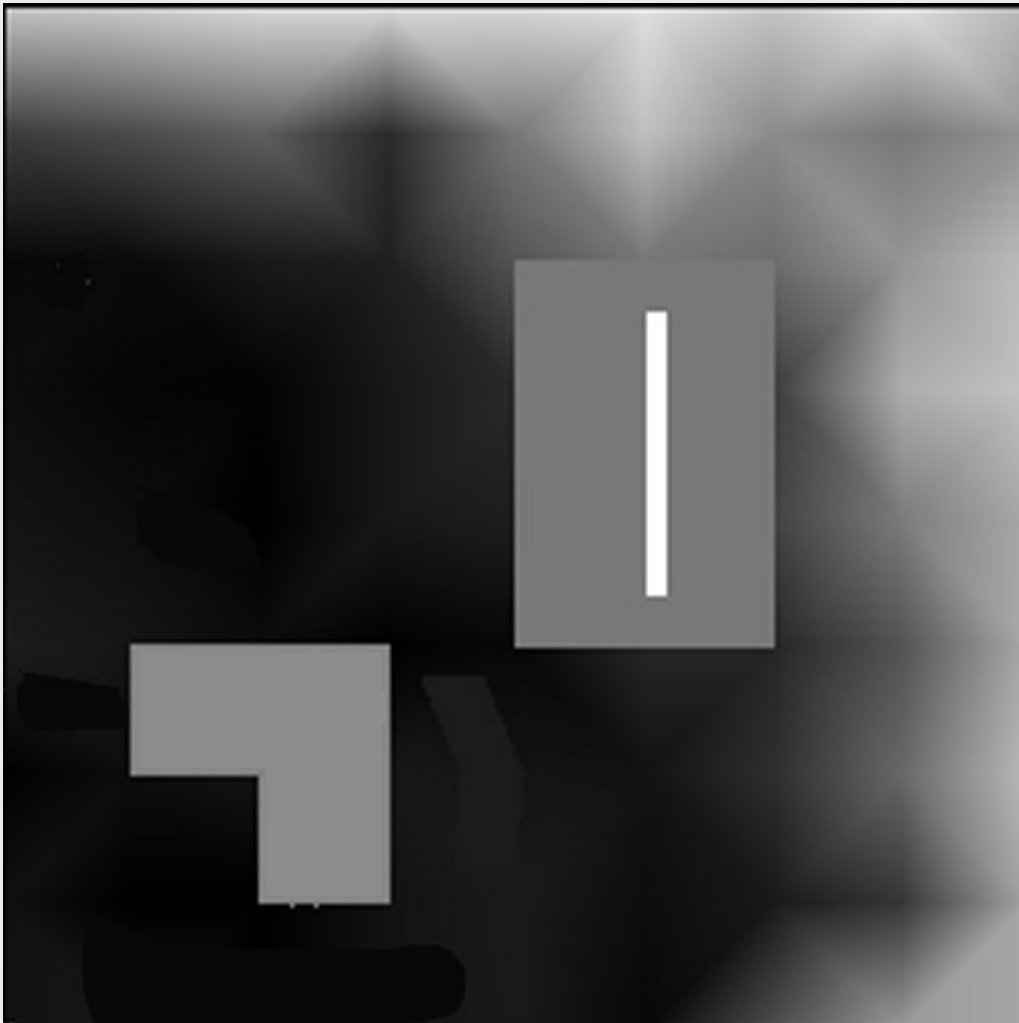
You get 3 new images. Discard the 2 most useless ones, and keep the clearest. I'm using:



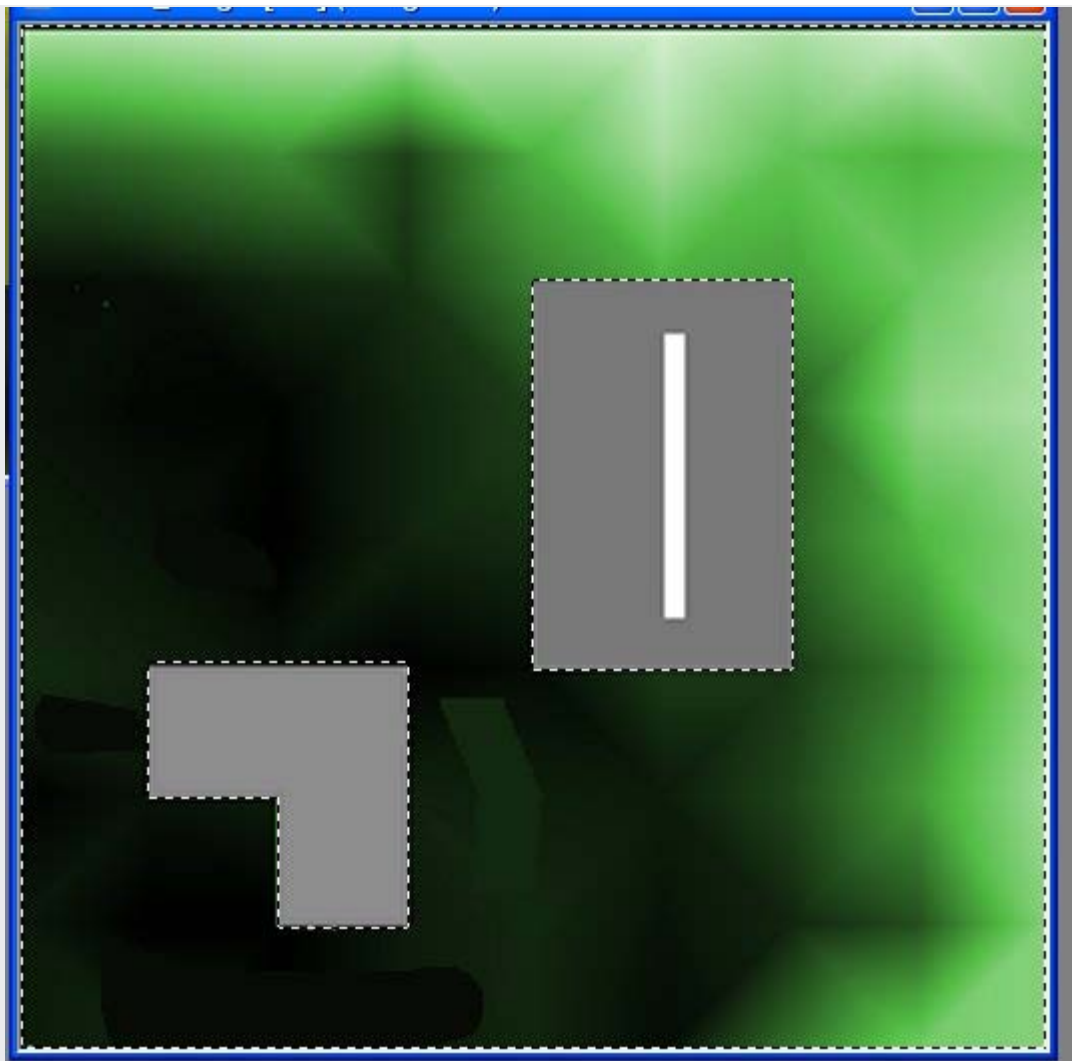


The image is 256\*256 - expand it to 512\*512 (don't just zoom in, actually make the image bigger). Also increase the colour depth to 24 bit, ie around 16 million colours.

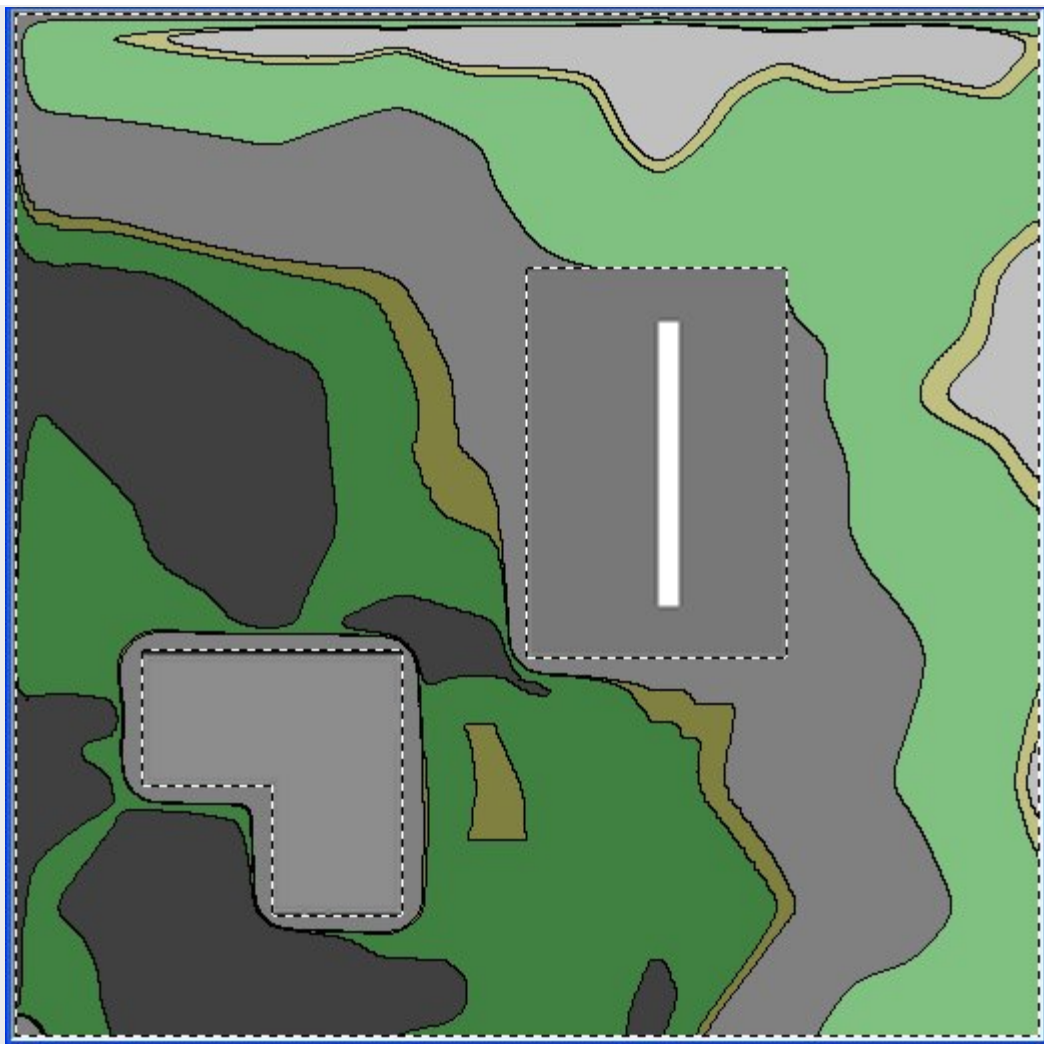
Using picture editing tools, tidy up the image to remove little grey specks (eg the fence posts, CP, ramp boxes, etc).



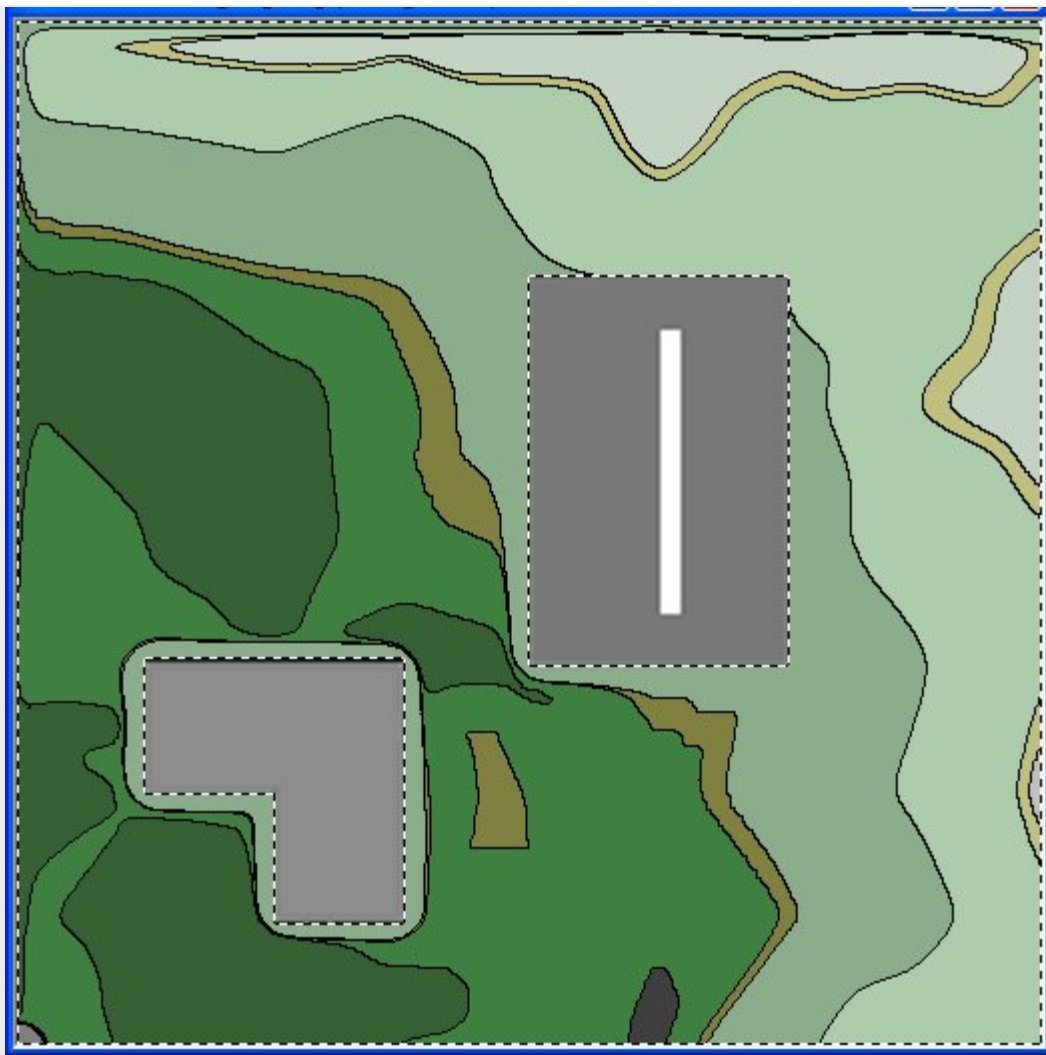
Then you should try to improve the look of it. I will give it a greeny colour, other than the buildings. So I select the buildings, invert the selection (to select everything else) and colorize it to a shade of green.



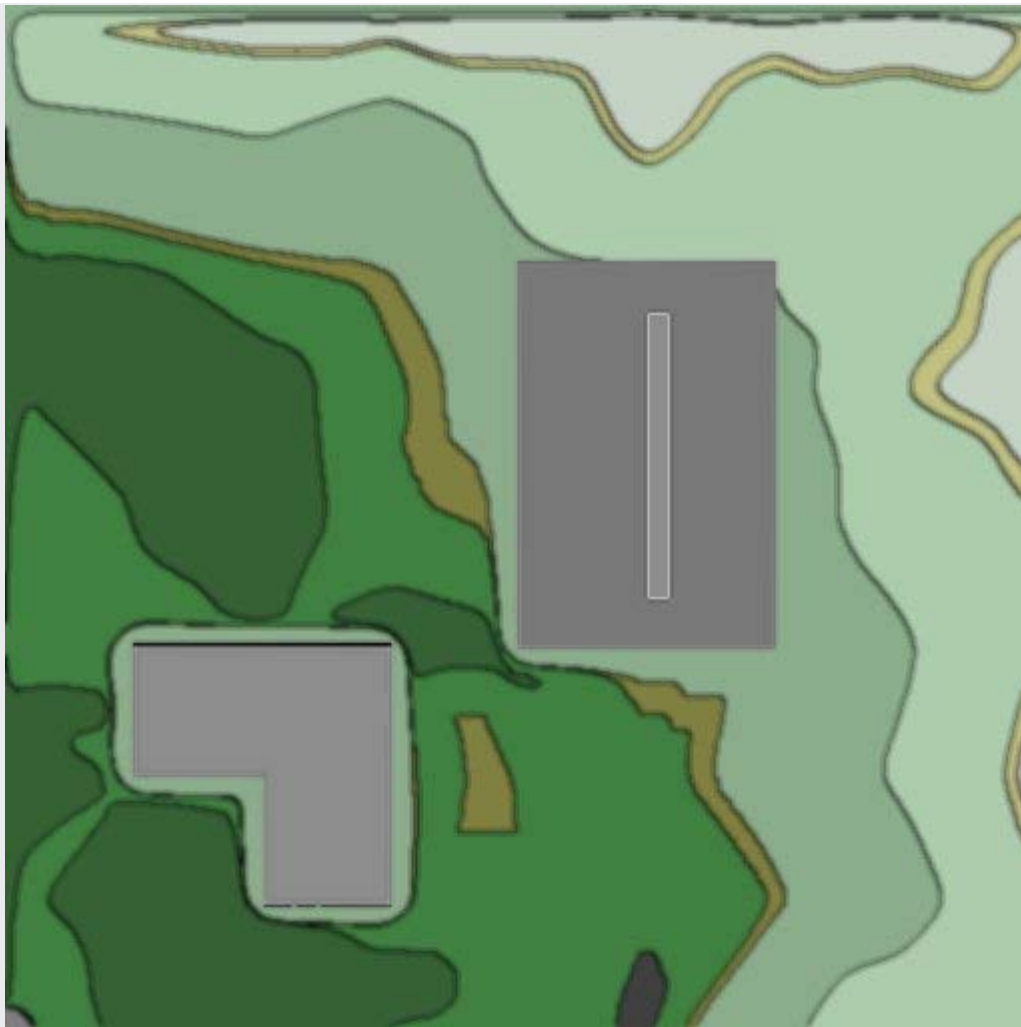
Then I apply a **contour** function, to make the green area look like a paper map.



Then I fill in some of the areas with better colours:



Finally I perform a **blur** function to soften the edges of the contours.



If you want to add writing (bear in mind there will be icons and marker text put onto the map in ET) you should create new layers and put the text onto them. Then if you want to change/move the text later on, you can. You won't be able to if you've put the text straight onto the map image.

There are many graphical effects you could employ to make the map more stylish and interesting - it's up to you, your imagination and your patience.

Save the file to a PSP format (to retain the layers info) then save a copy of it to etmain/levelshots/**tutorial\_cc.tga** - make sure the file is saved as a TGA type. Always work on the PSP version and save revised copies to the TGA version.

Run ET and run the map - you should see your own command map :)

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 36

### Topics

#### Making the picture to be shown while the map loads

[Making the picture to be shown while the map loads](#)

[Back to main menu](#)

#### Making the picture to be shown while the map loads

[\[Top\]](#)

Either source an image that you want to use, or more commonly, take an in-game screenshot (F11).

If you took a screenshot, the image will be in etmain/screenshots.

Run your image editor, I'm using Paint Shop Pro.

Open the required image file. A screenshot is likely to be too dark; if so, increase the brightness and the contrast until the image is good.

Then crop to focus on the relevant bit, and resize/crop again to 341\*256. Then resize to 256\*256 - you will need to turn off the usual feature that aspect ratios be maintained during a resizing.

If you want the image to be black and white, convert it to greyscale. Likewise, if you want to put text onto it, create new layers for the text.

Save the image as a PSP type (or the file type native to your editor) or you will lose the layer information, making later rework hard or impossible.

If you want the image to look like a crooked old photograph, like the original ET map images, resize to 232\*232 and enlarge the canvas (not resize) to give you some whitespace around the image. Select the 232\*232 image in the centre and rotate 1 degree to the left.

Use Pakscape (the link is in [lesson 10](#)) to open **etmain/levelshots/battery.tga** with your image editor, then copy the 232\*232 rotated image over the top of it - it should fit nicely within the white photo borders, and the surrounding black area is already marked out as a transparent area using the alpha channel. In other words, you don't have to worry about creating a transparency mask, as it is already present in the battery.tga.

Now save the image as **etmain/levelshots/tutorial.tga**.

Run ET and admire your photo as the map loads...



[Next lesson](#)



# ET Mapping Tutorial

## Lesson 37

### Topics

#### Making a PK3 file

[Making a PK3 file](#)

[Back to main menu](#)

#### Making a PK3 file

[\[Top\]](#)

Wow, just 37 lessons later and the survivors among you are ready to make your first PK3 file, so you can publish your map to the gaming community.

This is probably a good time to mention that not everyone will love or appreciate the couple of hundred hours of effort you put in. Despite being generally well received, I've had "This map is crap" from somebody or other for all of the maps I've made. You can't please everyone all of the time, so develop a thick skin now...

Your PK3 must contain all of the material that doesn't come as standard with the regular ET installation.

If you miss any components, either your map won't load/run properly, or the players will see the ugly yellow/black "missing texture" squares.

Run Pakscape (link is in [lesson 10](#)), click **File/New**

Create these folders (substitute **tutorial** for your map name):

- levelshots
- maps
- maps/tutorial
- scripts
- sound
- sound/maps
- sound/scripts
- textures
- textures/tutorial

Now we need to import all the required files into the appropriate pakscape folders. Be sure that you have removed any testing aids you added in the script, at least by commenting them out (eg, you may have shortened the respawn time, lengthed the map time or included "wm\_announces" to tell you when things are happening).

Click on **levelshots** and then right-click in the right-hand pane, and select **object/import file**. Navigate to **etmain/levelshots** and select **tutorial.tga** and **tutorial\_cc.tga** and click **Open** to import them.

Click on **maps** and import from **etmain/maps** these files: **tutorial.bsp**, **tutorial.objdata**, **tutorial.script** and **tutorial\_tracemap.tga**.

Then right-click again in the same pane and select **object/import directory** and select the **etmain/maps/tutorial** folder to import it and its contents (a load of files named **lm\_0000.tga**, **lm\_0001.tga**, etc). The number of files will depend on the size/shape of your map.

Click on **scripts** and import from **etmain/scripts** these files: **tutorial.arena**, **tutorial.shader** and **tutorial\_levelshots.shader**.

Click on **sound/maps** and import **etmain/sound/maps/tutorial.sps**.

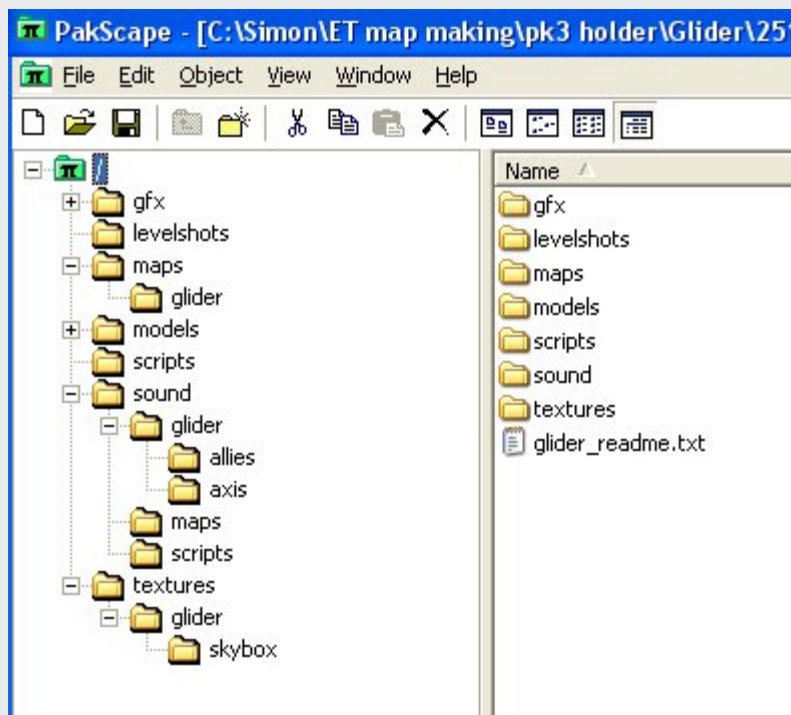
Click on **sound/scripts** and import **etmain/sound/scripts/tutorial.sounds**.

Click on **textures/tutorial** and import any bespoke textures files from the **etmain/textures/tutorial** folder.

If you are using a skybox, import the **skybox** folder and its contents also into the **textures/tutorial** pakscope folder.

Create a **readme.txt** file in Wordpad or similar, copying the style of an existing one from another pk3. Then import it into pakscope at the root level.

Here's an image of the **glider.pk3** contents as an example of how yours might look. There are additional folders than those we've covered, as there are components included that have not been covered in the tutorial thus far. I had allied and axis specific bespoke sounds in glider, hence the presence of **allied** and **axis** folders within the **glider** folder.



Save the file - I use a file naming convention that includes the map version. This helps people identify when new versions have come out, and also allows players to play different versions on different servers without perpetually having to download one version over the top of another.

So I would suggest calling version 1 either **tutorial1.pk3** or **tutorial\_100.pk3** (I use **v1.0.0** syntax for version naming).

You should now test the PK3 in your **etmain** folder. If you see horrible random shadows all over the place, it generally means you haven't properly included the **etmain/maps/tutorial** tga files contents.

Be sure to remove the pk3 from etmain before any Radiant reworking - if you forget, when you attempt to test your changes ET will instead still load the old pk3 version.

If you can, test your PK3 on a different computer, preferably one with a vanilla installation of ET, to highlight any omissions you may have made.

Your map is now publishable. The lessons to follow are optional, and rather more complicated, and only needed if you want to get on to more sophisticated elements such as tanks and fancy terrain merging shaders etc.

[Next lesson](#)



# ET Mapping Tutorial

## Lesson 38

### Topics

#### The Tank

[Creating the Tank](#)

[Plotting the route](#)

[Making a tank barrier](#)

[Writing the script](#)

[Back to main menu](#)

### Creating the Tank

[\[Top\]](#)

The tank is the biggest single element that you definitely don't want to create from scratch. The Jagdpantner I first used was taken from the sample Goldrush map supplied by Radiant. It took some time to even recognize some of the tank components as being tank components, especially those that the author didn't happen to put immediately adjacent to the tank. It took even longer to understand what the sizeable script was doing and rework it to do what I wanted.

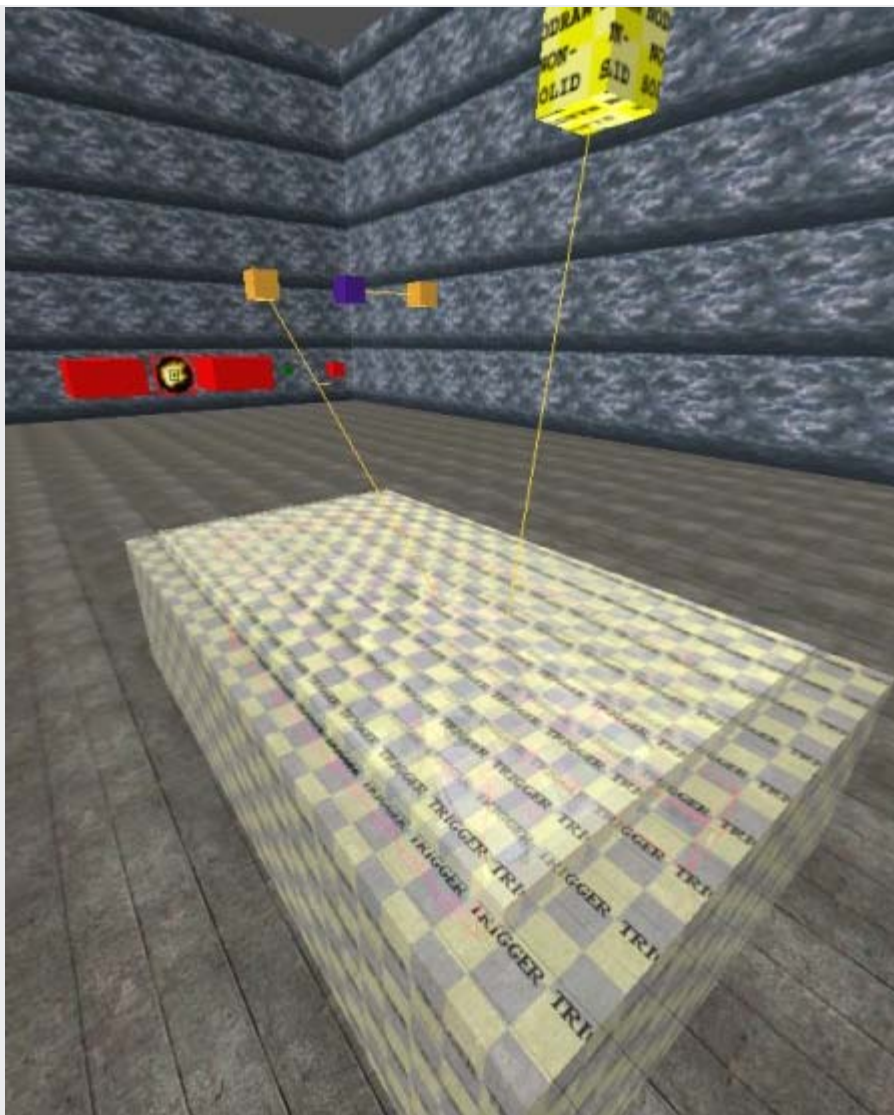
So to save you a lot of time and grief, I have made a [self-contained map](#) (we won't be using the tutorial map for this) and script, which includes a fully operational tank and tank barrier. I will explain all the components so that you can get them working in your own map. I have rationalized and re-ordered the script components (eg arranged them alphabetically, improved the comments, re-coded some elements for clarity) in an effort to make the scripting side as easy as possible. Note that even this minimal example tank script is 715 lines long - happily you only need to understand a fraction of it to make it work for you. :)

Download the tank.zip file and put the tank.map and tank.script in your etmain/maps folder.

Run Radiant and open tank.map. You will see the tank (smothered in trigger boxes), which is repairable by the allies, and a barrier, which is constructible by the axis.

If you want to immediately see what the tank will do in this sample map, compile it and have a go in ET. You should build the barrier, mend the tank and drive it to the barrier, destroy the barrier and then escort the tank (in an oval route back to the start) until it stops and fires the gun. Then come back to Radiant.

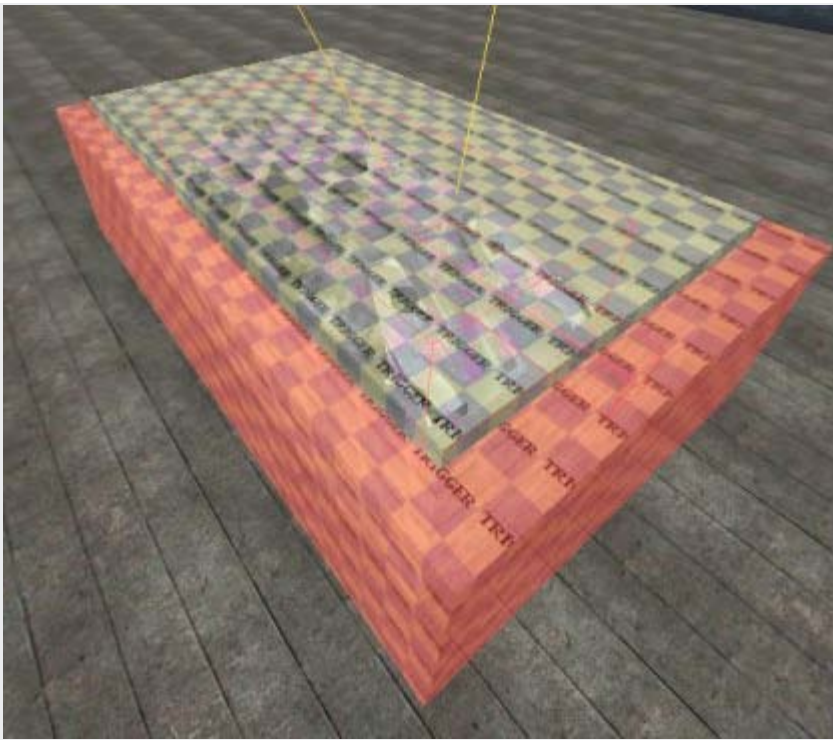
Hide the ceiling brush, it will make examining the tank easier. We'll look at each component - get a 3D view that shows all the tank elements, like this:



Note that it doesn't matter where the tank or any of its components is actually placed in the map - it will be immediately placed on the first point of the route it will follow, when the map starts.

Use shift+alt+click to select the largest trigger box. We need to use shift+alt+click quite a bit because many of these components have origin brushes inside and we want to include them when we examine and hide each element.

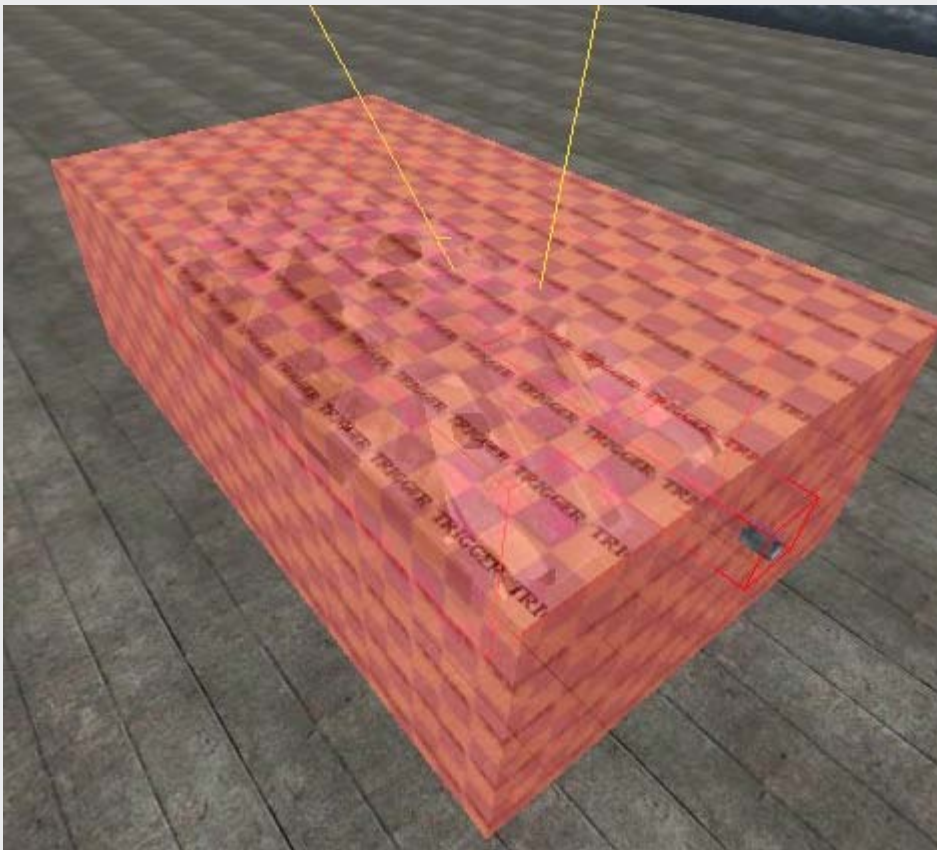




Press N, close the window, press N again.

This is the **tank\_trigger** - when an allied soldier stands within it, the tank is enabled to move (notwithstanding other reasons why it can't, such as damage or a barrier).

Just use a regular mouse click on the 3D pane and press H to hide the trigger while leaving the entities window open, and shift+alt+click the remaining big trigger box. We'll use the same technique for selecting and hiding each component.



\*\*\*\*\*

☐ axis\_objective    ☒ allied\_objective    ☒ mobile\_tank    ☐ is\_objective

☐ is\_healthammocabinet    ☐ is\_commandpost

---

classname	trigger_objective_info
objflags	7
shortname	Tank
infoAxis	This tank must be stopped from reaching the bank and allowing th
infoAllied	This tank must be stolen in order to blow open the doors of the ba
override	The Jagdpanther has been repaired
targetname	tank_build
scriptname	tank_build
spawnflags	10
track	the Tank
target	tank_construct
customalliesimage	gfx/limbo/cm_jagdpanther
customaxisimage	gfx/limbo/cm_jagdpanther

The **tank\_build** trigger is used to allow allied engineers to repair the tank.

As this tank is to be esorted by the Allies, the **allied\_objective** box is ticked.

The **shortname** will be shown on the command map.

**infoAxis**, **infoAllied** and **override** are redundant, and have never been changed since their use in Goldrush.

The **track** is shown along with "You are near".

This trigger is actually associated, via the **target**, to a func\_constructible called **tank\_construct**. To the player it appears that the plier-waving is for the tank, when in fact there is an invisible box in the air (the nodraw non-solid yellow box) which he is repairing (because the tank is not a func\_constructible).

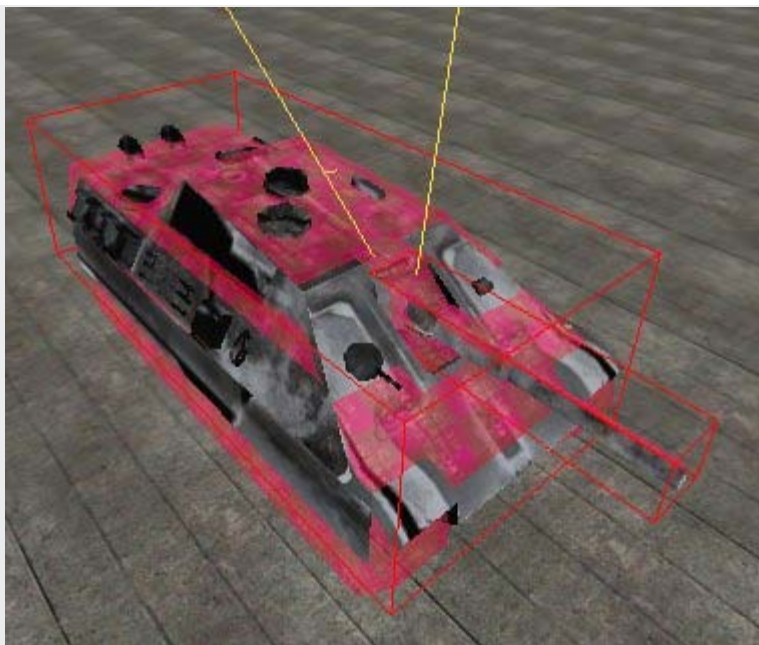
The **customimages** tell ET which icon to show on the command map.

Hide the trigger and shift+alt+click the yellow box.



It's shown as a func\_constructible. The health is a dummy value and doesn't matter what it is. The tank's health will be specified elsewhere.

Hide it, and shift+alt+click the tank brushes.



<input type="checkbox"/> triggerspawn	<input checked="" type="checkbox"/> solid	<input checked="" type="checkbox"/> explosivedamageonly	<input checked="" type="checkbox"/> resurectable
<input checked="" type="checkbox"/> compass	<input checked="" type="checkbox"/> allied	<input type="checkbox"/> axis	<input checked="" type="checkbox"/> mounted_gun
classname	script_mover		
model2	models/mapobjects/tanks_sd/jagdpanther_africa_tracks.md3		
targetname	tank		
scriptname	tank		
spawnflags	190		
health	1200		
description	Tank		
tagent	tank_shell		

This is the principal tank component. It is a **script\_mover** collection of brushes, because the brushes have to be capable of movement. The brushes are all **clip weapon metal** textured, to make the bullet ping sound when shot.

By default, script\_movers are non-solid, so that players would pass through them; so the **solid** box is ticked.

We only want explosives to harm this script\_mover, so the **explosivedamageonly** box is ticked.

When the script\_mover is destroyed, we want it to be able to start again with full health, so the **resurectable** box is ticked.

It should appear on the **compass**, and it can only be destroyed by the axis, so the **allied** box is ticked. Yes, a bit odd. Think of it as an allied tank.

The tank has a **mounted\_gun**.

The **model2** specifies that this tank-shaped collection of brushes should actually be visible as the jagdpanther tracks, which is a model.

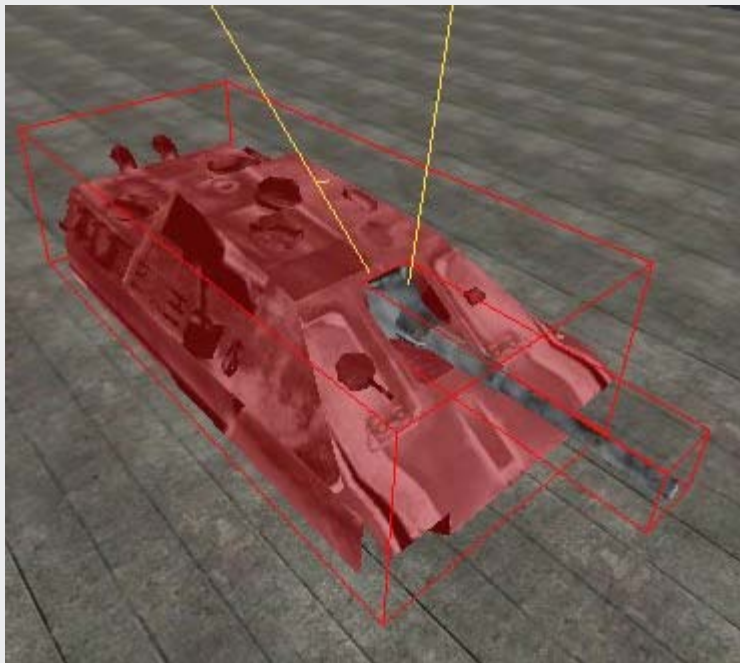
The **health** is 1200 which is the standard tank health value.

The **description** is what is shown when the player looks at the tank and sees its health bar.

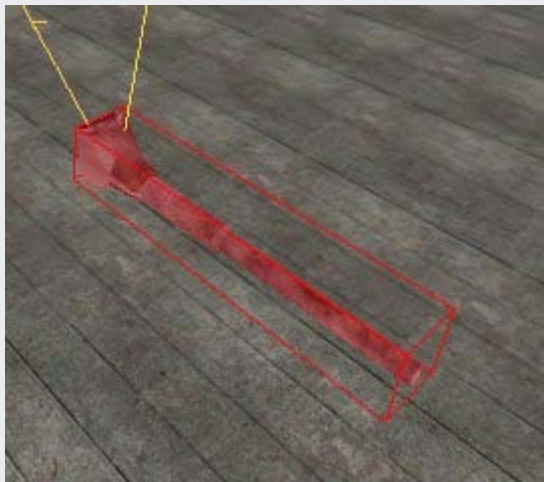
The **tagent** is used to tell ET where the player using the mounted gun should sit, ie in another entity called the tank\_shell.



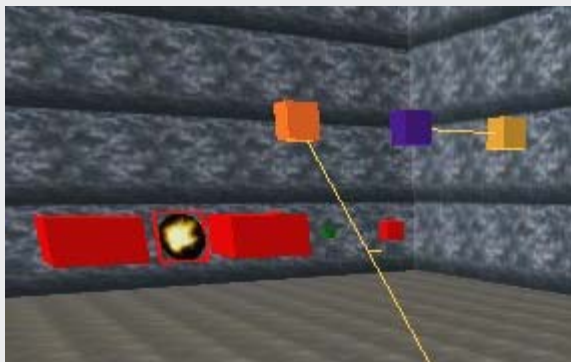
Hide it and select the tank model:



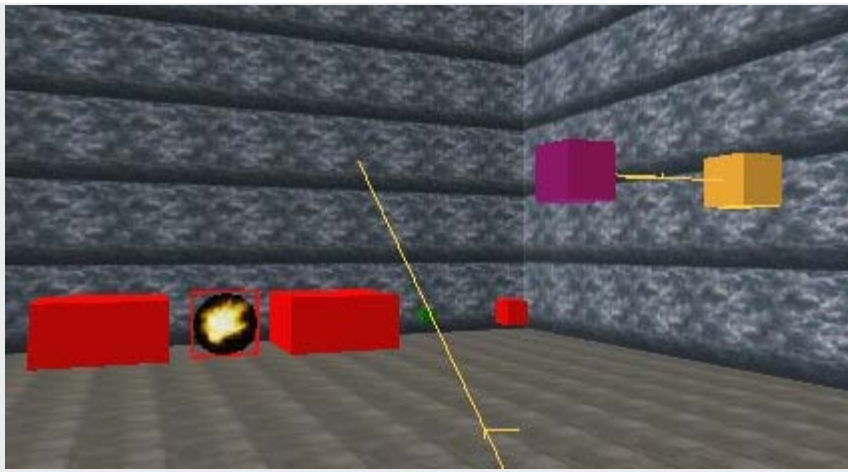
This is the **tank\_shell**, ie the hull of the tank. The model sits on the tracks. Hide it and select the gun barrel.



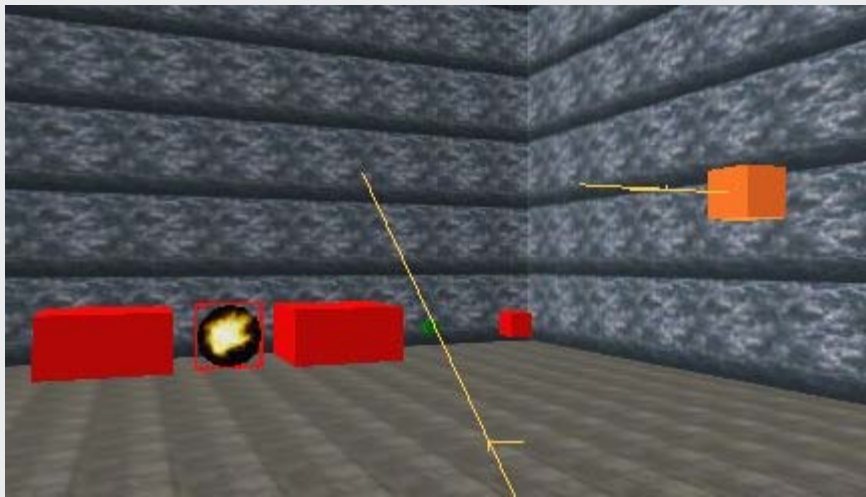
The tank turret is a separate entity, as it has to be able to turn to the side. Hide it and select the yellow target\_script\_trigger on the left.



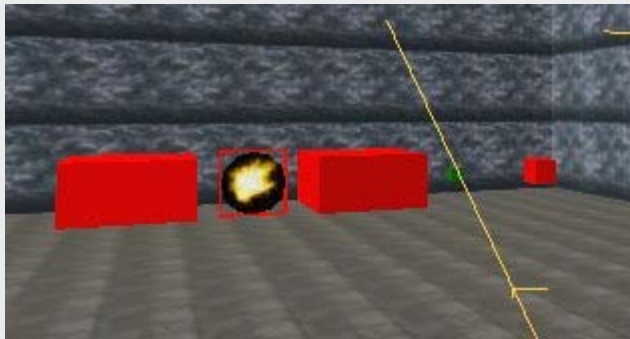
This is the tank\_trigger target\_script\_trigger. It tells ET which script element to run when someone stands near the tank. Hide it and select the mauve func\_timer box.



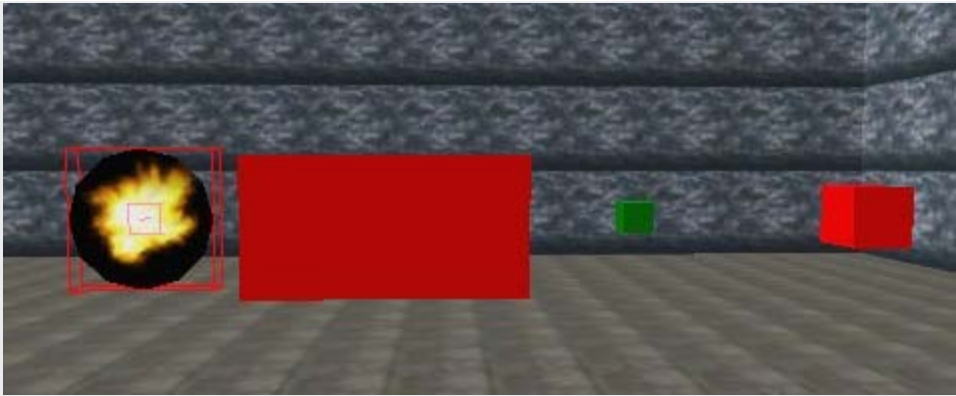
This is a timer, which executes every second (wait = 1). Each time it executes, it causes the script element specified in the yellow target\_script\_trigger shown on its right, to run. Its purpose is to halt the tank if no-one has been near it for a few seconds. Hide it and select the yellow box.



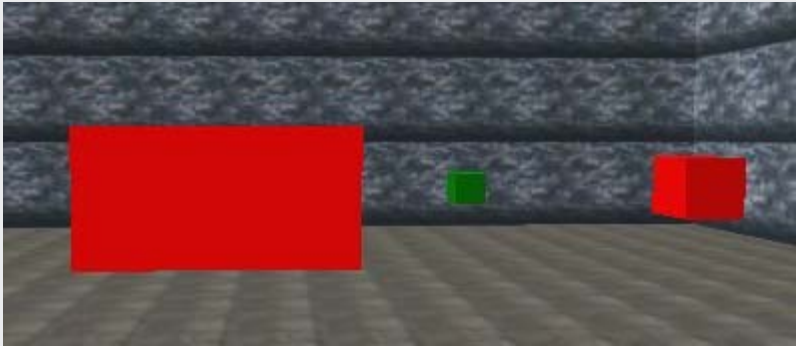
You can see this is the **tank\_disabler**. Hide it and select the big red box on the left.



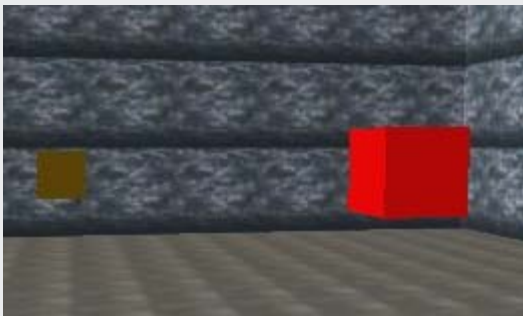
This is the smoke to be shown when the tank is damaged. It will show black smoke. Hide it and select the gun flash model. You will probably need to view the model from the back to select it.



This is shown for a moment at the gun tip when the gun fires. Hide it and select the big red box next to it.



This is something I've added. I thought it would be nice to have a puff of smoke at the tip of the gun barrel when it fires, so I added this smoke entity and called it `tank_gunsmoke`. Hide it and select the little green box.



This is just a placeholder entity to allow the script to manipulate sounds associated with the tank. It gives the script a named procedure it can use, called **`tank_sound`**.

Hide it and select the last red box. This is a sound entity. I used a different **`noise`** to the usual - I don't remember why now, it was while I was getting 2 tanks to work in the 2tanks map - so you could put this back to the standard gunfire sound: `sound/vehicles/tank/tank_fire.wav`

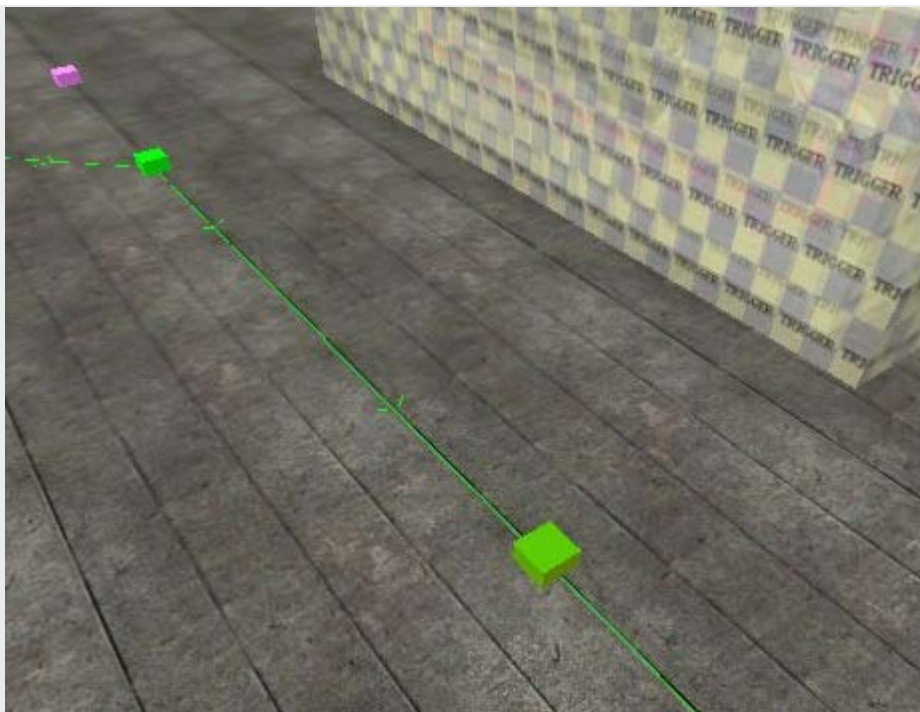
Unhide everything again. That completes the examination of all the parts that make up the tank.

## Plotting the route

[\[Top\]](#)

You control where the tank goes by plotting a route for it. Select the green box near the front of the tank.





target	spln1
targetname	spln0
origin	2496 3008 0
classname	info_train_spline_main

The boxes are splines, an entity called **info\_train\_spline\_main**. Each spline points to the next spline along the track, by using the **target** key.

I have made an oval circuit. Normally the splines define a winding route, not a circuit, and the last spline does not have a **target** key, as there is no next spline.

The origin of the tank tracks will follow the origins of the splines.

When you're making the route, the easiest way is to copy the last spline in the track so far, move it into place, and rename its **targetname** and **target**. Repeat as necessary.

These example splines are actually too far apart. As a rule of thumb, they should be about 3 intersections apart at grid scale 7. This is because when the tank is damaged, it won't actually halt and smoke until it reaches the next spline. If you put them too far apart, a wrecked tank will go on too far.

The pink boxes are **info\_train\_spline\_controls**, and they are used to smooth the turning from one spline to the next. Without them, the tank will turn quite abruptly. Don't add them until you've laid out your route with some confidence that it won't change, because re-doing the spline controls is really tiresome and best avoided - so leave them to last.

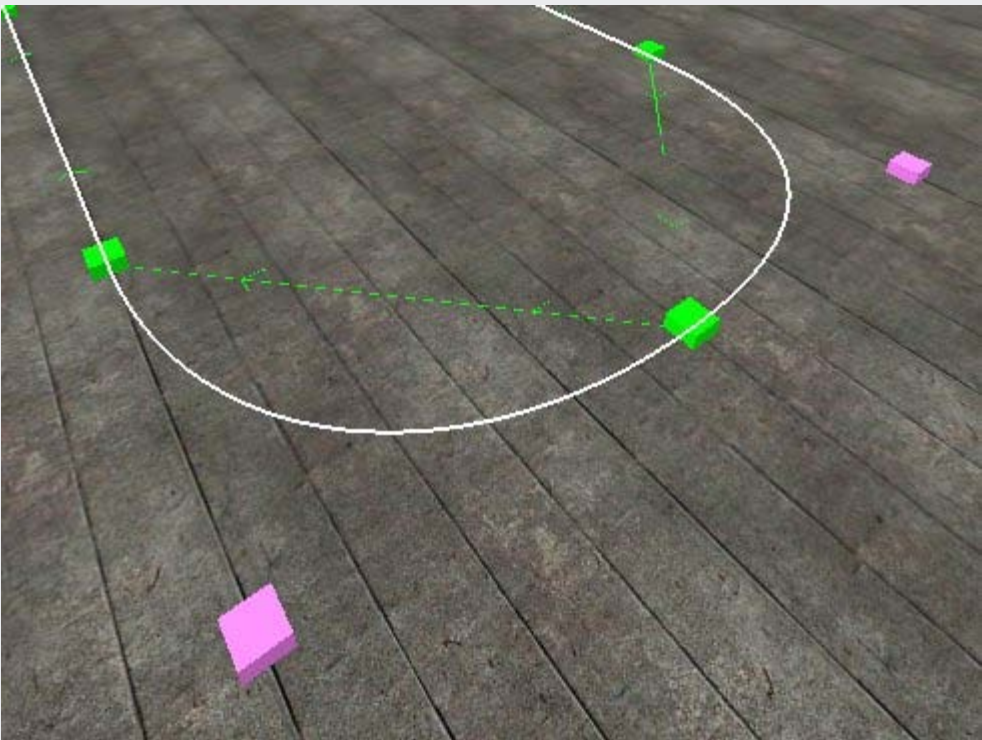


Select one and press N - you will see the only thing of interest is its name. I name them with the same name as the spline they are controlling, plus "\_c". Select a spline, like one of those at the pointy end of the circuit, and press N. You will see it has a **control** key. Splines only need a control key when you decide the tank movement from this spline to the next needs smoothing.

In the picture above, the centre spline leads diagonally up to the left to the next spline. It has a control, the pink box to the left. So the tank won't go straight along the green arrow, it will bow out towards the pink box and then back to the target spline.



This is best seen when you activate the **plot splines** feature. Click this button: Nothing seems to happen. Press ESC.



This shows with the white line, the path that the tank will actually follow. Handy to spot lumpy plots that need smoothing out with control splines.

**Making a tank barrier**

[\[Top\]](#)

I have put a fairly standard tank barrier in this map - it's just a func\_constructible, like any other. This one has been made an axis constructible. It's the script that will make the tank react to the barrier - without the script barrier detection, the tank would otherwise just drive straight through it.

**Writing the script**

[\[Top\]](#)

Open the tank.script with Wordpad or similar. I've arranged the scripting elements alphabetically. I'll explain just those elements of note or those you'll need to amend to make the tank script work in your own map.

Look at the **barrier** procedure. There are a number of **teamvoiceannounce** commands - remember you will need to include their definition in your **etmain/sounds/scripts/<yourmapname>.sounds** file. You can copy them straight out of the goldrush.sounds file.

If you made the barrier an explicit objective, you'd need to add a limbo camera and the objective ticking/crossing code.

Scroll down to the **tank** procedure.

All the **accum** comments are pretty much what came with the goldrush script, and you can more or less ignore them until you get more proficient with scripting.

Scroll down to see each of the next sections as they get described:

**spawn** : (for your info) This puts the tank onto the start of the route, and gets it facing the right way (ie the way to go from spln0 to spln1) by running the tank forwards and then backwards along the first spline.

<b>followspline</b> 0 spln0 50000 length 32 wait	Move the tank along a spline route
followspline <b>0</b> spln0 50000 length 32 wait	Go forwards along this spline. 1 means go to the next spline and move backwards to this one.
followspline 0 <b>spln0</b> 50000 length 32 wait	Identifies the spline to follow.
followspline 0 spln0 <b>50000</b> length 32 wait	The speed to move at. Normally speeds are 50-80. 50 is the speed used in Goldrush. 80 is used in Fuedump. 50000 is extremely fast and is used just for positioning at game start.
followspline 0 spln0 50000 <b>length 32</b> wait	Tells ET that the tank origin is not at its actual centre, and to compensate for this during turning.
followspline 0 spln0 50000 length 32 <b>wait</b>	Tells ET to wait until the movement is completed before executing the next script line.

Scroll down to **trigger dispatch**.

You'll need one of these lines per spline. Make sure the numbers match, eg accum 3 trigger\_if\_equal **7** tank run\_**7**

Scroll down to **trigger run\_0**.

You'll need one like this for each normal spline, ie a spline that isn't the last spline or a point at which a barrier may impede tank progress. Change run\_**0** to run\_<the next number>. Changle spln**0** to spln<the next number>.

Look at **trigger run\_3**.

If the tank starts along spline 3, it means it has passed the barrier - so remove the barrier.

Look at **trigger run\_7**.

This would be your last spline. You can see the tracks stop, the engine stops, the tank shudders to a stop (the animations) and then the turret will turn to fire. If you don't want the tank turret to turn and fire, delete the line **trigger tank\_turret turn**.

Scroll down to **trigger stuck\_check\_finished**.

The **8** is the value of the last spln number + 1. Change it to your last spln number + 1.

Scroll down to **trigger stuck\_check\_barrier**

The **abort\_if\_not\_equal 3** is being compared to the spln number in front of the barrier, ie the point at which the tank should stop if the barrier is built. Change the 3 to wherever your barrier is. If you have more than one barrier, clone the **trigger stuck\_check\_barrier** procedure with a different name, and set the abort\_if\_not\_equal value accordingly. You'll also need to clone the line **trigger self stuck\_check\_barrier** into the name of the new procedure, in the **trigger stuck\_check** routine.

You can ignore the rest of the script, it will work just fine as it is.

Now go play with your tank - experiment with changing the route and get the hang of control splines.

Next lesson - not yet ready.



We are a small website-development company, specialising in providing a personal service at affordable prices.

On your behalf we register your domain name and organize the hosting of your new website with the UK's largest web hosting company. We then design and create a professional custom website to meet your needs. In addition we can provide graphic design for your corporate identity to complement the style of the website.

You are welcome to [contact us](#) to chat about your requirement, even if you are not quite sure what it is! We avoid tech-speak and talk in plain English to make the whole process straightforward for both parties.





- Home
- About us
- Our services
- Our customers
- Contact us

PythonOnline's principal website developer is [Simon](#), who has over 30 years' experience in designing and programming a variety of computer applications. He has worked in senior programming, project-leading and consultancy roles for a number of corporations and major companies including Lloyds Bank, Nat West Bank, HSBC, SocGen Bank, British Telecom and Logica Systems. Simon also spent some years as a director of a small software house developing applications for treasury and dealing systems of international banks.

Simon has been developing websites for small businesses since 2001, and understands the advantages in discussing the projects with clients using plain English instead of baffling technical terms.







## Our services

- Home
- About us
- Our services
- Our customers
- Contact us

PythonOnline offers the following services:

- A free no-obligation discussion of your requirement, be it a revamp of an existing website or the creation of something brand new
- Registration of your domain name and reservation of server web space (which comes with unlimited email addresses)
- Design and creation of the website using text and images supplied by the client, supplemented with custom and independently sourced images
- Placement of the developed website onto the web server
- We can also create graphic designs to match the style of the website for business stationery such as letterheads, leaflets and business cards

In addition we can optionally provide a simple, non-technical facility to enable you to update the content of the web pages we craft for you. With this invaluable system you can change the images and text whenever you want and without incurring delays or extra costs.

On completion of the project we advise you of all the technical magic numbers that are associated with the new website, so that if you choose to use another web host or the services of another web developer, you are free to do so.

Websites developed for our clients will be written to modern standards which are intended to improve the quality and consistency of the web pages when viewed by different browsers.

Prices start from as low as a few hundred pounds. [Contact us](#) to discuss your own project without obligation.







Home About us Our services Our customers Contact us

This page shows some of the websites we have created, covering such diverse subjects as a hotel, a college, an adult education organization, a hairdressers, a yoga centre, a clairvoyant, a museum attraction, a holiday home let, a village society and a designer clothing retailer.

We have done a number of websites for Parish Councils, one of which won the Local Council Review / Co-Op Bank Website of the Year award, while another came second in the following year's competition.



Metropolitan Home Improvements - loft conversions in the South East



(In development) Sarlton Facilities - mechanical and electrical engineers



Piper for Weddings - bagpipers professionally played on special occasions



Andromeda Vehicles - car leasing company



Universal Wisdom - multi-language site presenting the letters of Gurudev Shri Ojaswi Sharma



The Brickwall Hotel - 16th century Tudor mansion hotel and restaurant in Sussex



ACRES - Adult College for Rural East Sussex



Casa Carlino - Holiday home near Mount Vesuvius, Italy



David Rae Hair Spa - The London Hairdressing Salon in Petts Wood



Robertsbridge Community College - A specialist mathematics and computing college



The Hurstpierpoint Society - Preserving the rural village environment of Hurstpierpoint



Stephen Austen - Clairvoyant medium, healer, channel, author and metaphysician



Yesterday's World - The award-winning attraction in Battle, East Sussex



Yoga Universal - Yoga centre in the heart of the Sussex countryside



Cheeky2 Kidz - International retailer of children's designer clothing



Salehurst Parish Council - Winner of the Co-op Bank Website of the Year Award 2004



JumpCo - Supplier of quality portable cross country fences



George Cross Island Association - Commemorating the Siege of Malta 1940-1943



Sussex Associations of Local Councils - Providing support to parish councils



Tiny Fingers Tiny Toes - Hand crafted ceramic impressions of babies' hands and feet



Lindfield Rural Parish Council



Hurstpierpoint Parish Council - Second prize in the Co-op Bank Website of the Year Award 2005



Eye Can Help - Dyslexia specialists



## Contact us



- Home
- About us
- Our services
- Our customers
- Contact us

PythonOnline is based in Robertsbridge, East Sussex.

Please click here to [contact us](#) by email, or you can complete the form below if you prefer. In either case we will respond as soon as possible.

Fields marked \* are required

Name \*

Telephone

Email address \*

Repeat email address \*

Enquiry \*





# Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Jump To: [Congratulations](#) · [Icons](#)

This document was successfully checked as XHTML 1.1!

Result:	Passed	
Address :	<input type="text"/>	
Encoding :	windows-1252	(detect automatically)
Doctype :	XHTML 1.1	(detect automatically)
Root Element:	html	
Root Namespace:	<a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a>	



The W3C validators are hosted on server technology donated by HP, and supported by community donations.

[Donate](#) and help us build better tools for a better web.

## Options

Show Source

Show Outline

List Messages Sequentially  
Type

Group Error Messages by

Validate error pages

Verbose Output

Clean up Markup with HTML Tidy

[Help](#) on the options is available.

## Congratulations

The document located at <http://www.pythononline.co.uk/uk/index.asp> was successfully checked as XHTML 1.1. This means that the resource in question identified itself as "XHTML 1.1" and that we successfully performed a formal validation using an SGML, HTML5 and/or XML Parser(s) (depending on the markup language used).

### *"valid" Icon(s) on your Web page*

To show your readers that you have taken the care to create an interoperable Web page, you may display this icon on any page that validates. Here is the HTML you could use to add this icon to your

## Web page:



```
<p>
<a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```



```
<p>
<a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```

A [full list](#) of icons, with links to alternate formats and colors, is available: If you like, you can download a copy of the icons to keep in your local web directory, and change the HTML fragment above to reference your local image rather than the one on this server.

### Linking to this result

If you would like to create a link to *this* page (i.e., this validation result) to make it easier to revalidate this page in the future or to allow others to validate your page, the URI is

[<http://validator.w3.org/check?uri=http%3A%2F%2Fwww.pythononline.co.uk%2Fuk%2Findex.asp>](http://validator.w3.org/check?uri=http%3A%2F%2Fwww.pythononline.co.uk%2Fuk%2Findex.asp)

(or you can just add the current page to your bookmarks or hotlist).

### Validating CSS Style Sheets

If you use [CSS](#) in your document, you can [check it](#) using the W3C [CSS Validation Service](#).

↑ TOP

[Home](#) [About...](#) [News](#) [Docs](#) [Help & FAQ](#) [Feedback](#) [Contribute](#)



This service runs the W3C Markup Validator, [v0.8.5](#).  
COPYRIGHT © 1994-2009 W3C® (MIT, ERCIM, KEIO), ALL RIGHTS RESERVED. W3C  
LIABILITY, TRADEMARK, DOCUMENT USE AND SOFTWARE LICENSING RULES  
APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR  
PUBLIC AND MEMBER PRIVACY STATEMENTS.





# ET Mapping Tutorial

## Introduction

This tutorial has been set up to help would-be mappers looking for a step-by-step guide from the beginning. It assumes the reader has no idea how to use GtkRadiant and knows nothing about scripting and the like.

These pages were originally created for members of the TibeT clan and regular players on the TibeT servers, but anyone finding this tutorial is welcome to use it :)

I don't claim to know everything about mapping but I've learnt enough from various sources to be able to make [maps](#) of reasonable complexity. I've had a number of people asking various mapping questions and I thought it was a better idea to explain it all once to anyone interested. The tutorial is incomplete but I continue to add pages as I get the time.

Be aware that the subject is very large so there's a lot to explain; and that to make a decent fair-sized map can easily take over 200 hours, so with this as a hobby you need a lot of patience and spare time, but at least the tools to do the job are free. The best maps out there might have taken their authors 18 months to create, depending on how much free time they had.

Disclaimer: I may be wrong in my understanding of any aspect of the subject, but hey, I'm doing my best :)

## Introductory topics

To get from knowing nothing about mapping to being able to run around in your own first little map, follow these introductory topics in order from start to finish.

- [Getting Started](#)
- [Making your first brush](#)
- [Making your first map](#)

## First set of intermediate topics

Create a building inside a landscape, with a destructible window and working door.

- [Creating an environment](#)
- [Making a building outline in your environment](#)
- [Making a door](#)
- [Making a destructible window](#)



## Second set of intermediate topics

A break from brushes: a quick delve into *some* of the other mapping elements.

- [Creating the initial script](#)
- [Creating the mission text to be shown as the map loads](#)
- [Adding ambient sounds](#)

## Third set of intermediate topics

Back to Radiant again, for models, simple destructibles and constructibles. Starting to get more interesting now. Don't attempt these unless you've completed the previous topics.

- [Planting a tree](#)
- [Making stuff you can shoot up](#)
- [Making barbed wire](#)
- [Making a ladder](#)
- [Making a constructible MG42](#)

## Fourth set of intermediate topics

A set of topics with some variety, helping you ease into some of the more complicated subjects..

- [Secure doors](#)
- [Lighting](#)
- [Detail brushes vs structural brushes](#)
- [Health and ammo cabinets](#)

## Fifth set of intermediate topics

Some of the principal game features, the CP, bendy shapes and terrain.

- [Command Posts](#)
- [Curved walls and arches](#)
- [Cylinders, cones and curved roads](#)
- [Making terrain using GtkGenSurf](#)
- [Fine tuning the terrain by dragging vertices](#)
- [Skyboxes](#)

## Sixth set of intermediate topics

Some topics that help to bring interest to your map.

- [Bespoke graphics](#)
- [Making a constructible object like a ramp](#)
- [Making a destructible object like a gate](#)
- [Forward spawn flags](#)
- [Water](#)
- [Team speech](#)

## Final set of intermediate topics

This last set of intermediate topics cover the remaining components you'll want to make a distributable PK3 package.

- [Limbo camera and objectives narrative](#)
- [Making the game end](#)
- [Generating a tracemap](#)
- [Making the command map](#)
- [Making the picture to be shown while the map loads](#)
- [Making a PK3 file](#)

## Advanced topics

This final set of topics covers some of the more complicated stuff that you may never want to use - but if you've made it this far with the tutorial, you probably will :)

- [The Tank](#)
- Scripting in detail
- Grabbing the gold or radar parts, etc
- Making constructible/destructible doors
- Bespoke sound
- Making something simple move
- Trucks
- Shaders
- Custom command map icons



## Berlin preview

### Berlin screen shots









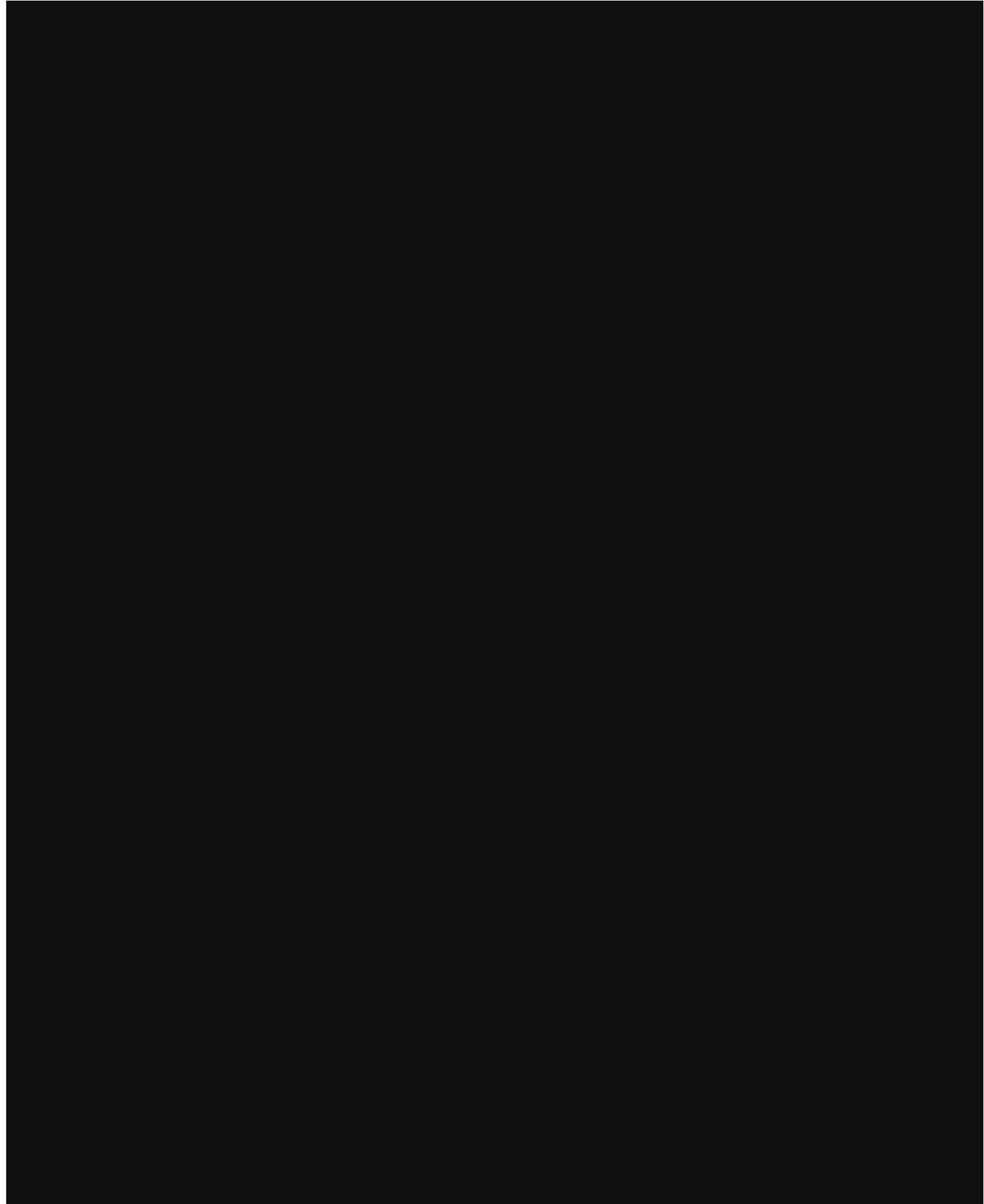




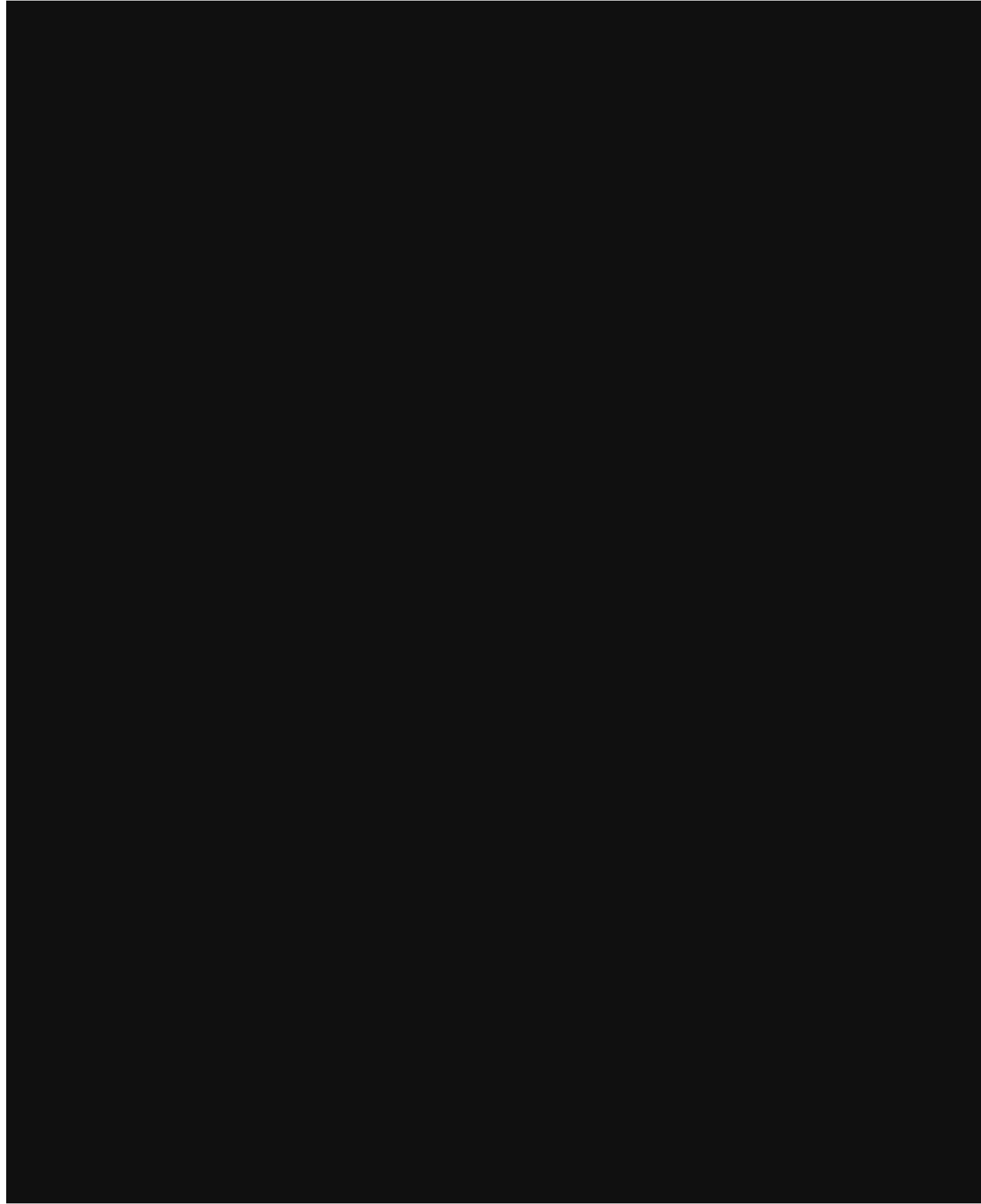






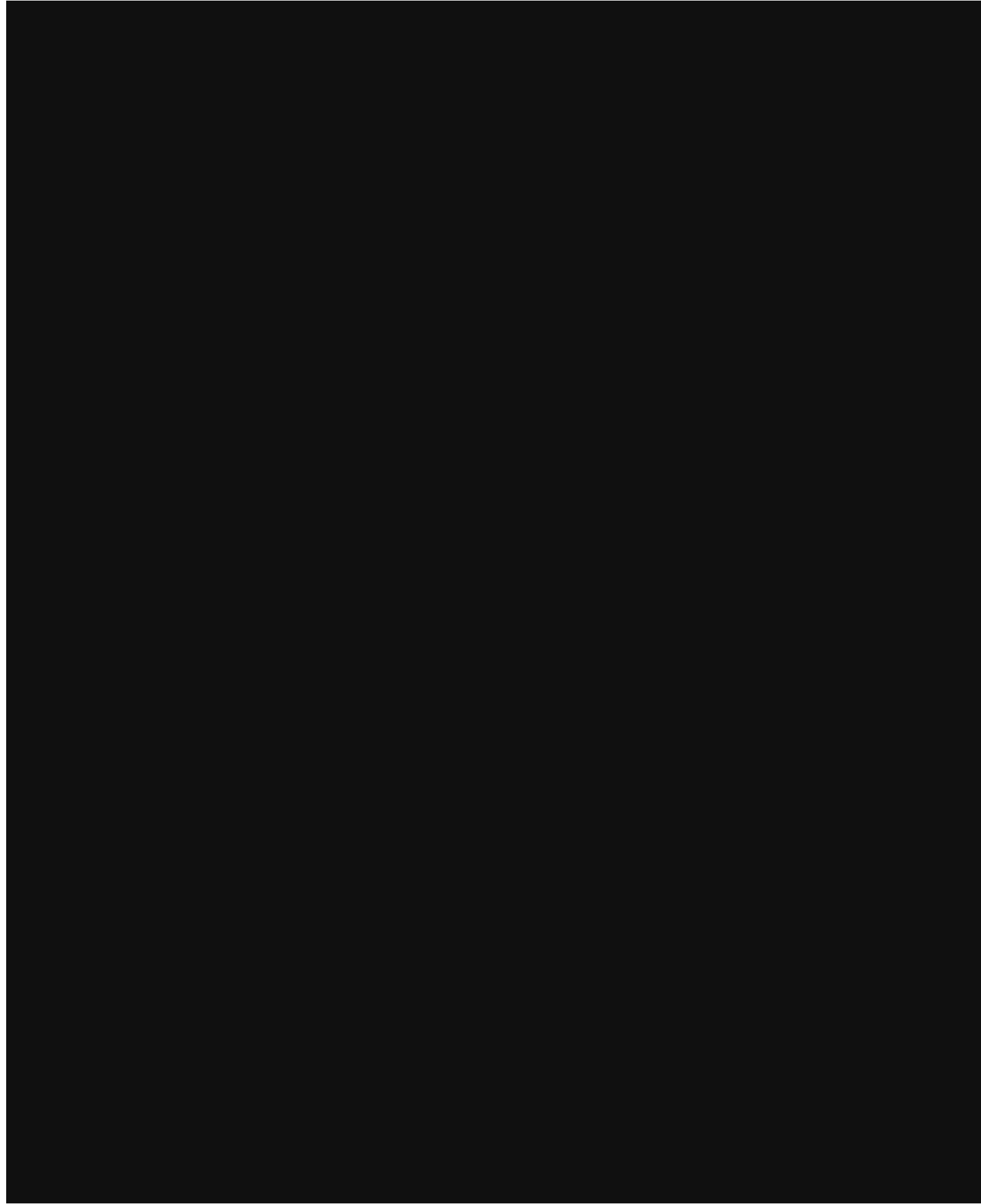




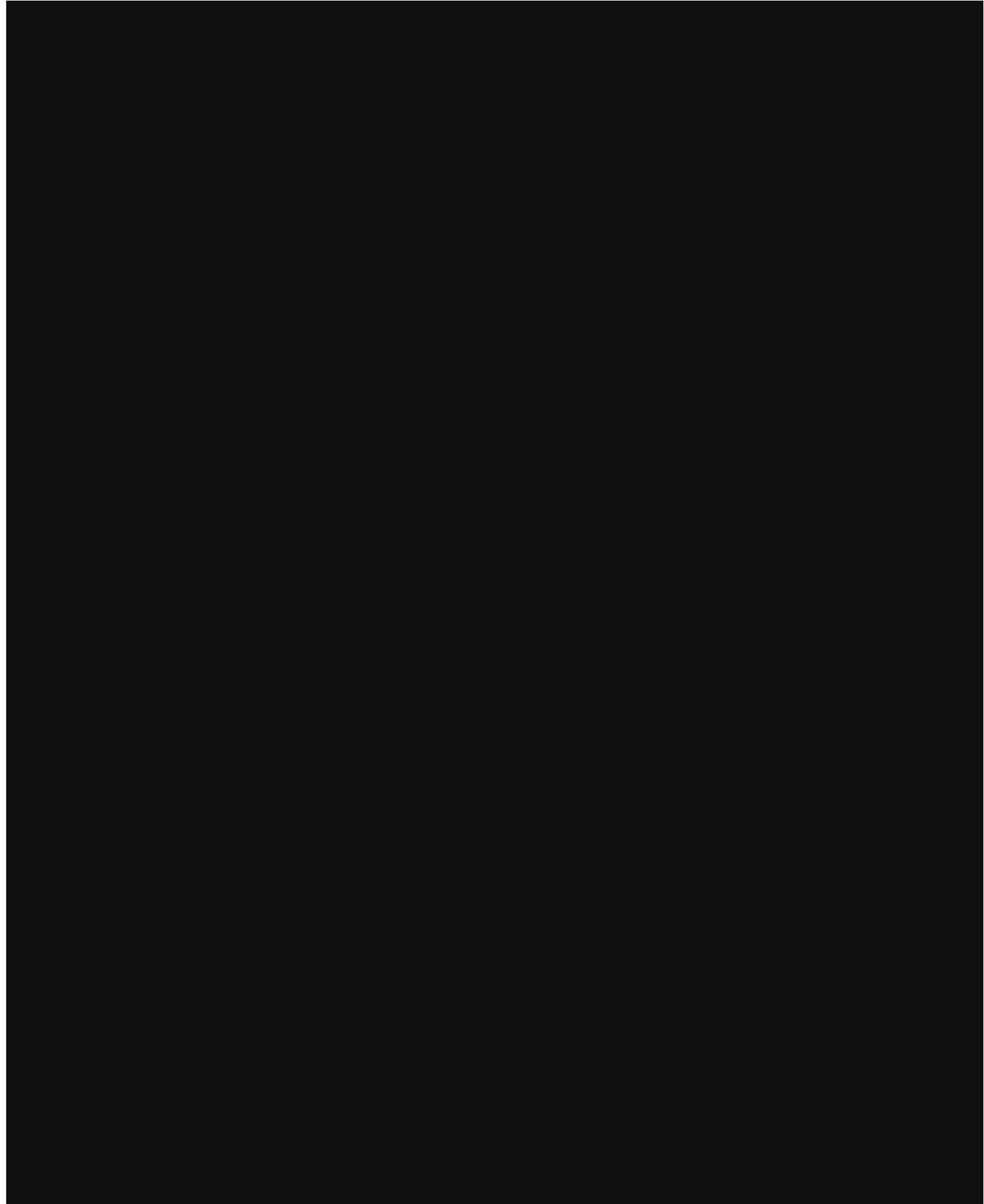




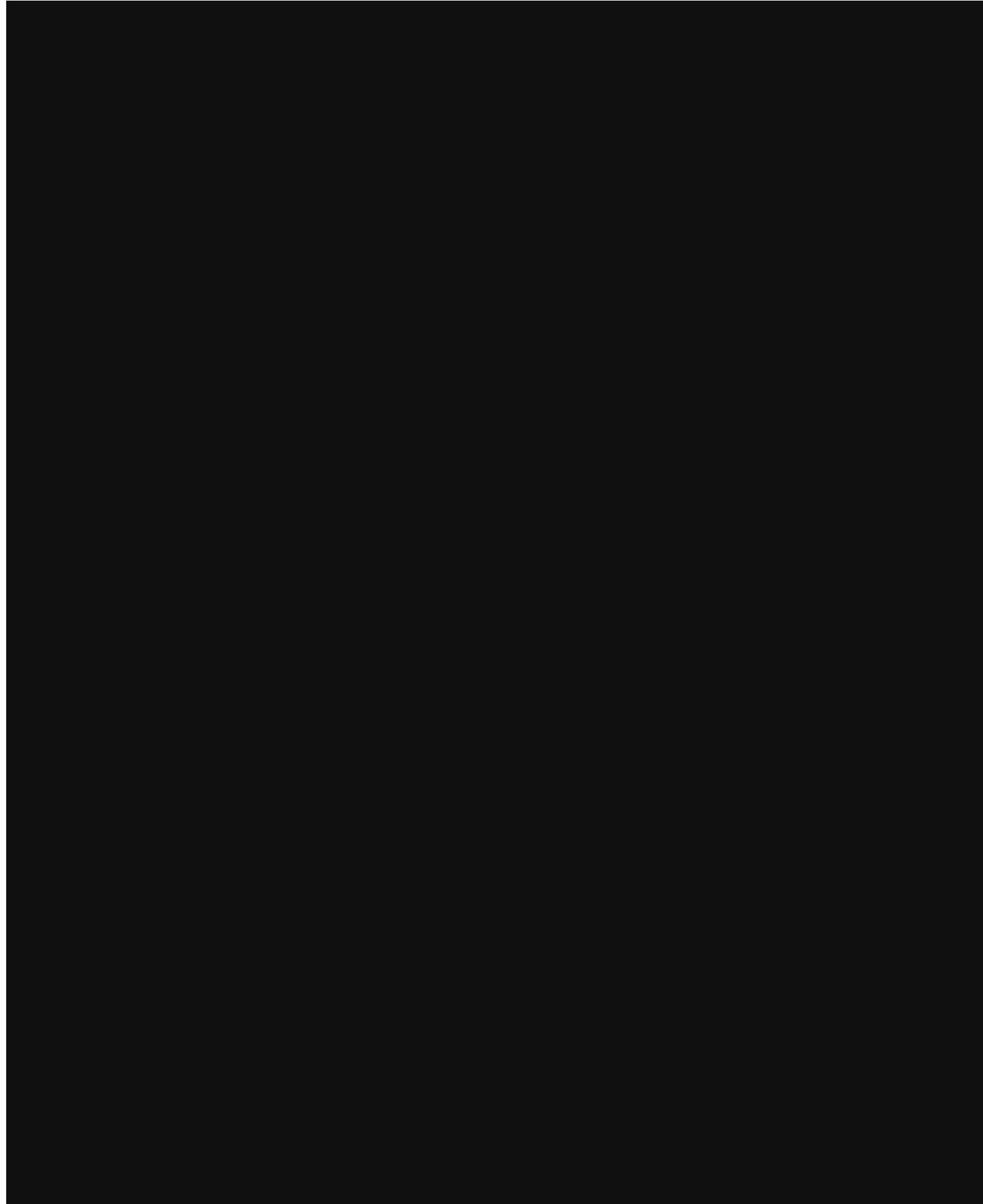


















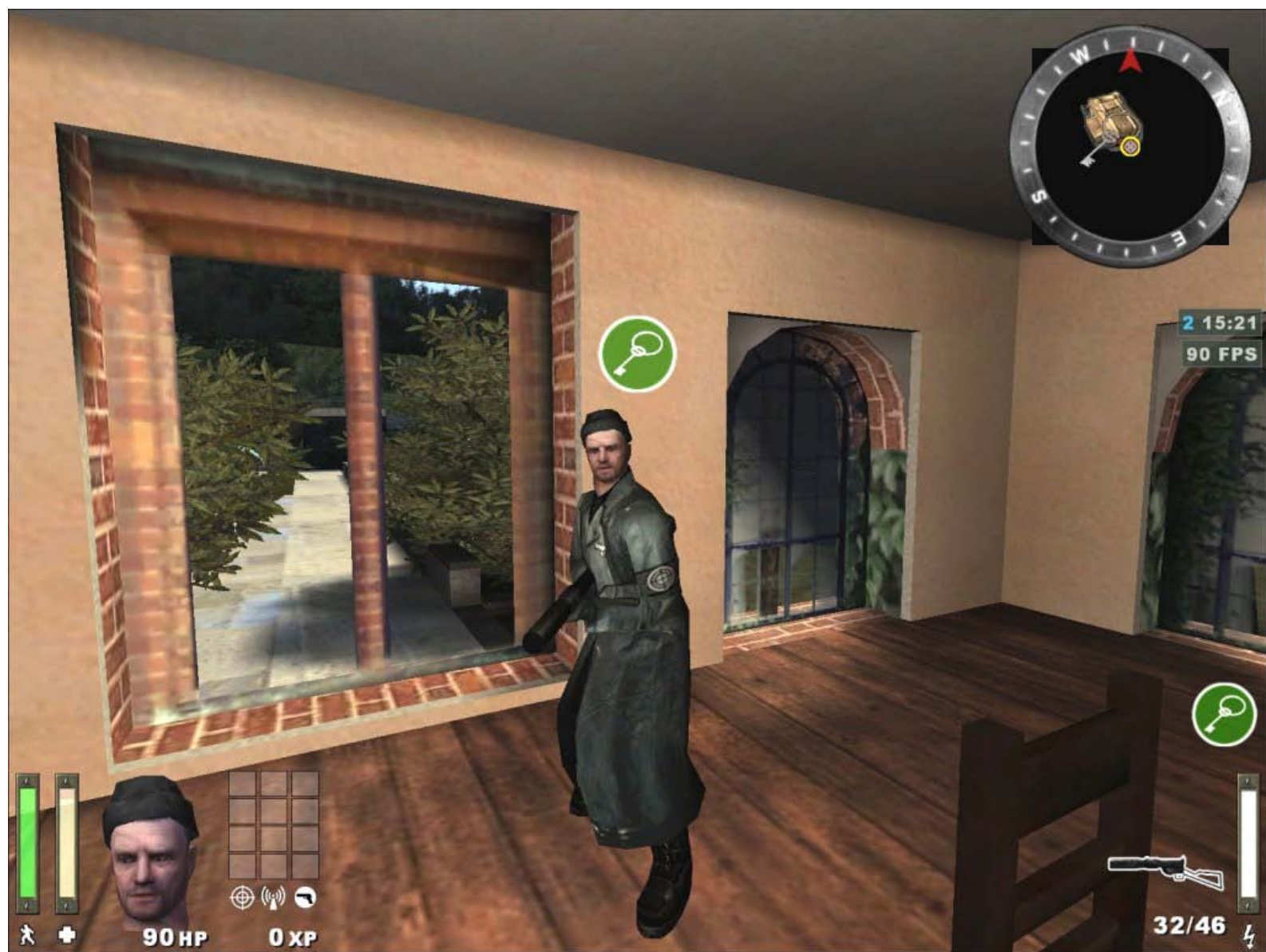




























# Ludendorff Bridge

## Gameplay

This webpage has been created to provide a description of the gameplay in Ludendorff Bridge (the bridge at Remagen). There are enough novel gameplay features that I thought it worth explaining them here, so that readers will immediately know how to play the map if they encounter it. Please visit [www.tibetclan.com](http://www.tibetclan.com) for details of TibeT servers that the map may be playing on.

ET elements used in a non-standard way are highlighted with this:



### Scenario



The Allies wish to capture the bridge intact to allow their armour to cross the Rhine. The Axis must destroy the bridge to prevent this, but not so soon that their own troops cannot retreat across it. The action takes place mostly on the bridge, which has been built to scale.



Click for a larger image

## Winning the game



The game only ends when the timer (20 mins) expires - no sooner and no later.

**AXIS WIN:** if at game end, they have planted explosives on both bridge supports.

**ALLIES WIN:** if at game end, there are no explosives planted on the bridge supports.



If at game end there are explosives planted on just one bridge support, the winning team is **chosen randomly**. This is because neither side knows whether one set of explosives is enough to destroy the bridge. It also ensures both teams don't camp defensively - they need to get *both* supports secure to ensure a victory.

When explosives are planted on a bridge support, a large flag is erected on one of the Axis towers, and the corresponding flag on the Allied tower is taken down. Similarly the opposite happens when the Allies defuse the explosives. The game starts with one set of explosives already planted.



Click for a larger image

## Spawn flags

There are 3 spawn flags along the bridge.



Your forces can only spawn at the farthest flag in an **unbroken line** of friendly-controlled flags. For example, if your team controls all 3 flags, you can spawn at any of them, including the farthest from your base. If the enemy then takes the flag nearest to your base, the chain is broken and the two flags further forward are cut off. You would then only be able to spawn at your base.

Starting respawn times are 10 seconds for each team.



When your team captures a flag, you may or may not be able to spawn there (see above). However your respawn time increases by 5 seconds for each flag captured. This represents the problems of supply lines, and has the effect of aiding the defender who has been beaten back to his starting point by making it progressively harder for the aggressive team, the further they advance. When a team loses a flag, their respawn time drops again by 5 seconds.

## The {TibeT} Tank



The tank on the bridge can be damaged, repaired and driven by both teams. If accompanied by Allies, it will advance West. If accompanied by Axis, it will advance East. If players from both teams are near the tank, it will stop.



The tank has a number of uses.

- It provides mobile fire support with its MG42.
- It provides mobile cover for advancing troops.
- When it is halted by a tank barrier near the ends of the bridge, it turns sideways to give following troops maximum cover from enemy fire.
- It acts as a minesweeper to clear enemy mines from the gravel paths alongside the railway tracks.

## Gameplay

This is intended to be a brief, fierce battle on a long bridge, in which teamwork will be vital for success. There is a lot of cover provided in the shape of wrecked vehicles, sandbags, crates and improvised defensive barriers built by engineers, all of which makes the snipers' job harder. The constructed barriers can be destroyed by satchel charges. The crates provide good cover but can eventually be destroyed by gunfire.

I hope you'll enjoy the map :)

## Screenshots

[Click for a larger image](#)





# ET Mapping Tutorial

## Lesson 5b

### Topics

#### Making a building outline in your environment

[Floor and ceiling](#)

[Back to main menu](#)

### Floor and Ceiling

[\[Top\]](#)

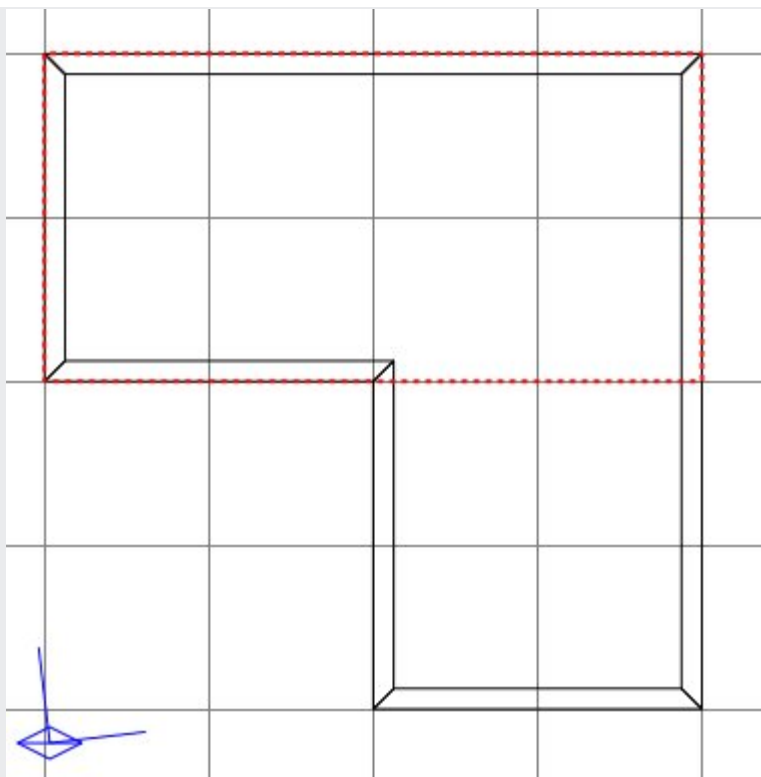
Run Radiant. Open the tutorial map. Adjust zoom and scroll (right-click and drag) in the 2D window so you can see your building outline. You might want to move your 3D view too so that you can see what you are building in the 2D window.

A quick way to find the area you want in your 2D window, which happens more often as the map gets bigger and more complicated, is to select one of the brushes of interest in the 3D view, and then press ctrl+tab. The selected brush is now centred in the 2D window. Usually you would press ctrl+tab twice more to return to the top down view.

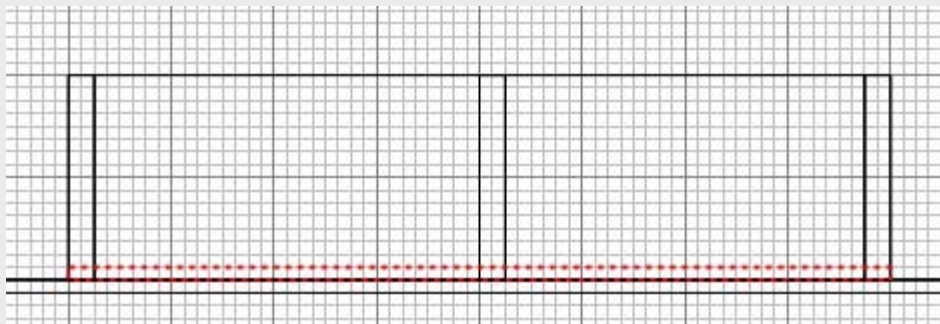
Press 8 to make the floor brush creation very easy.

We're going to want to make a couple of caulk brushes - rather than create them with a random texture then caulk them, we'll set the texture before creating the brushes: click on the Caulk texture in the textures window.

Make a brush as shown in this picture.

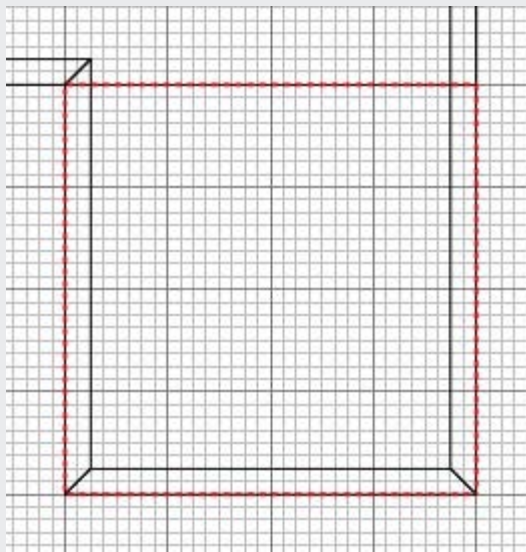


Press ctrl+tab to get a side view. Press 4 to reduce the grid scale, and then reduce the height of your floor down to very flat:



Press ctrl+tab twice to get back to the normal 2D view. Press ESC.

Draw another brush as shown. This one will be of the right height already as you will immediately see in the 3D window :)



Press ESC. Select the environment ceiling brush and Hide it - you will only accidentally keep selecting it otherwise. You can reveal hidden brushes with shift+H.

With floors and ceilings it is often the case that you accept some overlap with the walls, and don't worry about the strip of texture "wasted". It can become just too grim trying to angle everything to prevent this. However, for the purpose of making quick progress we are creating this building sitting on the main exterior hull-caulked wall (which we've given a snow texture to on one side) and ordinarily you don't do that. Generally we will have created terrain and then set the building into it; or the building will be contained in its own volume of space, like our original tiny room.

In both of those cases you can usually create an efficient floor space, in the knowledge of what the player will actually be able to see. We'll get to this later.

So for now we're being a little quick and dirty, but it will have a negligible effect on FPS so don't worry.

Let's move our walls up a notch so that they sit on the new floor.

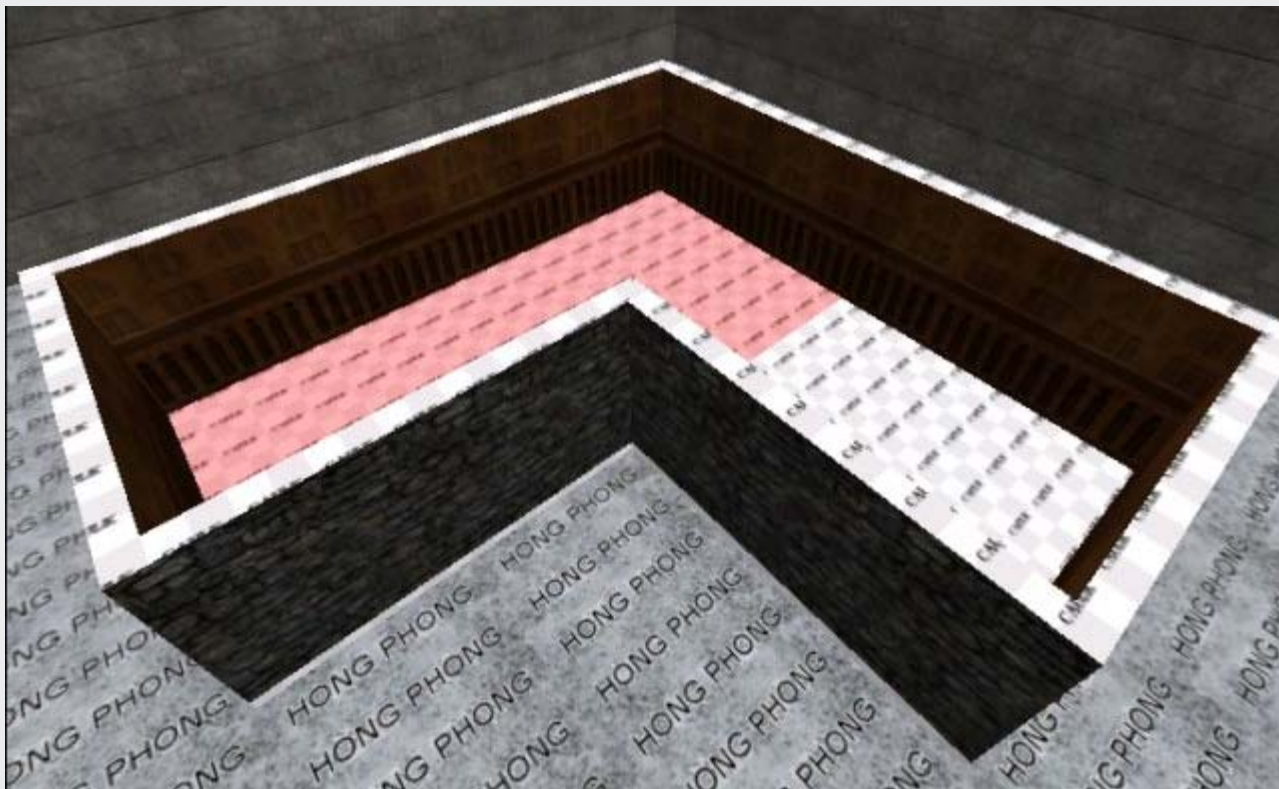
Select all the walls in the 3D window by shift+clicking them all one after the other.

Press ctrl+tab to get a side view.

Move the walls up one notch so they rest on the floor brushes.

Press ESC, ctrl+tab twice, and right-click/drag the building back into view if it has wandered off centre.

Let's texture the floor. In the 3D view, ctrl+shift+click a floor brush face.



Then ctrl+shift+**alt**+click the other brush's floor face.

Then click Textures/egypt/egypt\_floor\_sd and click the block-16sq texture.

Remember you can adjust the relative sizes of the Radiant windows so you can access more easily what you want at any given time. So I made the textures window bigger, clicked the texture I wanted, and then made it back to smaller again. Also I find the Texture Window Scale of 50% to usually be the most workable.

Press ESC to deselect the faces and press **8** to go back to a big grid scale. Always worth doing as a habit.

You may have noticed that with a small grid scale, when you zoom out, the grid display loses the



finer lines and only draws major lines, giving the impression of a larger grid scale. Mind you don't get caught out by this.

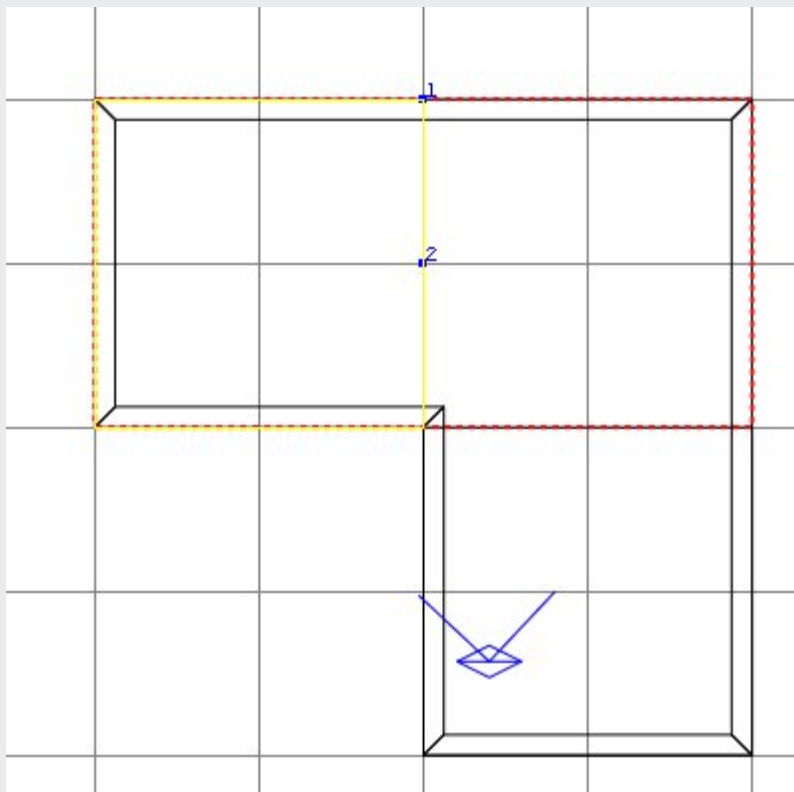
As the walls are now sitting on the floor, we can see a thin rim around the bottom which is the caulked sides of the floor brushes. We'll need to texture them too.

You can also see that by texturing the lower edge of the longer floor brush, half of its length will be obscured by the square floor brush, which is something we can easily avoid and it demonstrates another option available to the mapper, the Clipper Tool:



Select the large floor brush, and either press **X** or click the button shown:

Click on the vertices shown in this picture, starting with the uppermost one. When you click on the first one, the number 1 will appear next to it. When you click on the second, the number 2 will appear, and the brush will be shown with a dividing line defined by where you clicked 1 and 2, with one chunk shown yellow and the other still red.



The Clipper Tool has two functions. It can...

1. Crop an unwanted chunk of a brush: if you were to press Return (don't in this instance), the yellow section would remain and the red would be deleted.
2. Split a brush into two: if you press shift+Return (yes please, do it now) the brush is merely split along the line defined.

Both chunks remain selected and the cropper tool remains active. Press ESC (or X or click the button again) to turn off the cropper.

Press ESC to deselect the brushes.

Now we can texture all the wall rim without having some texture wastefully drawn but not visible.

Select all the wall rims by ctrl+shift+clicking one followed by ctrl+shift+**alt**+clicking all the others (in the 3D view). You'll need to use the right-click in the 3D window to allow free movement/turning in that view so you can see all the rims. Don't forget the ones round the back.





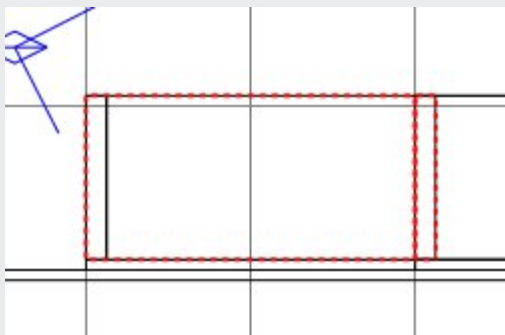
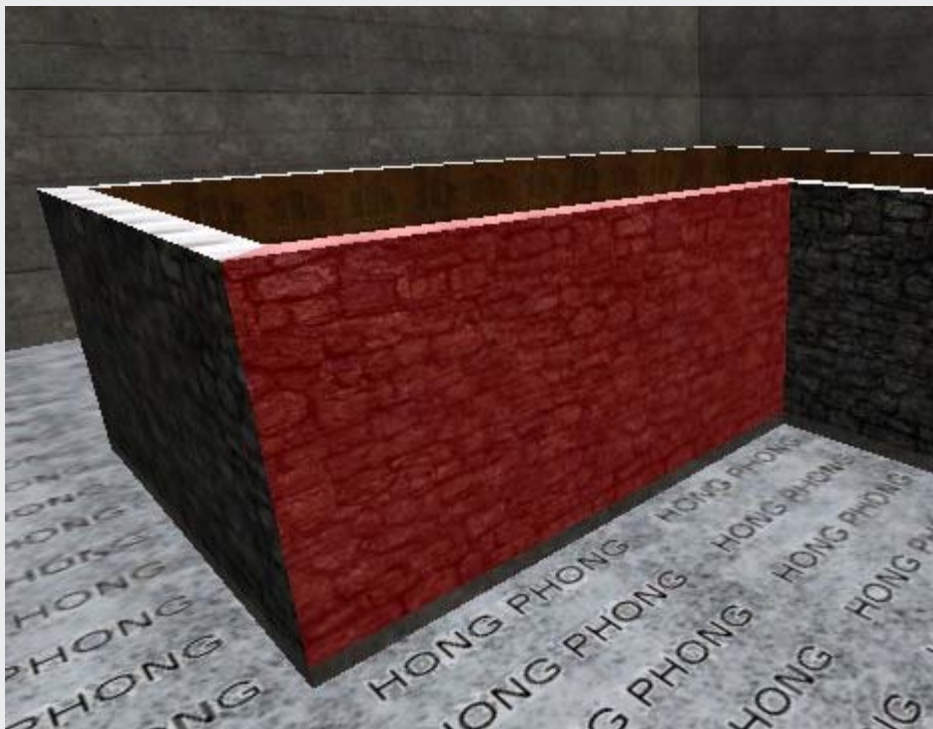
Apply a suitable texture, say Textures/town/town\_wall town\_c61a. Press ESC.



Another good habit to cultivate is to deselect anything you're done with. A **very** common error is to leave something selected, then select and work on something else, then wonder wtf mangled your first brush into such a bizarre shape :(

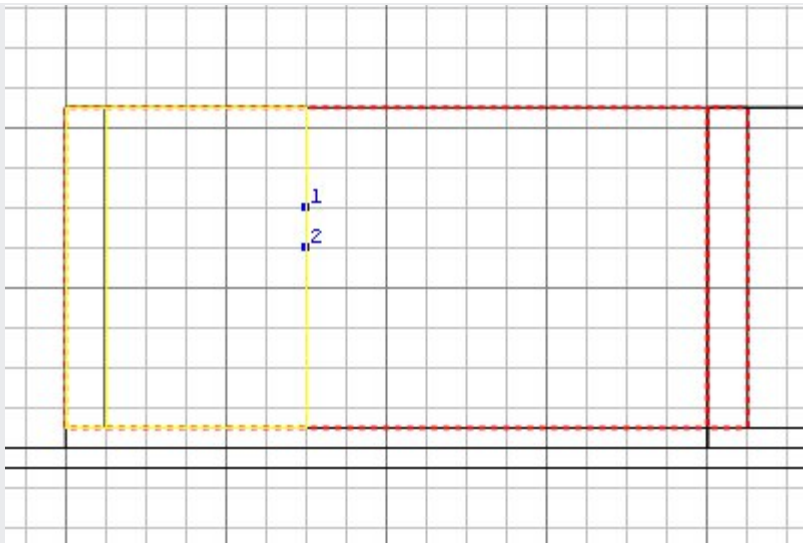
We'll hold off from putting a ceiling on for a minute. Let's make an opening in a wall so we can walk through it. If you make a mistake during this, use ctrl+z to undo. Also might be worth doing a quick file save before you start, for good measure :)

Select the indicated wall in the 3D window, and press ctrl+tab to see it side on.



Press **5** for a smaller grid scale, then use the clipper tool and define a cut line as shown:





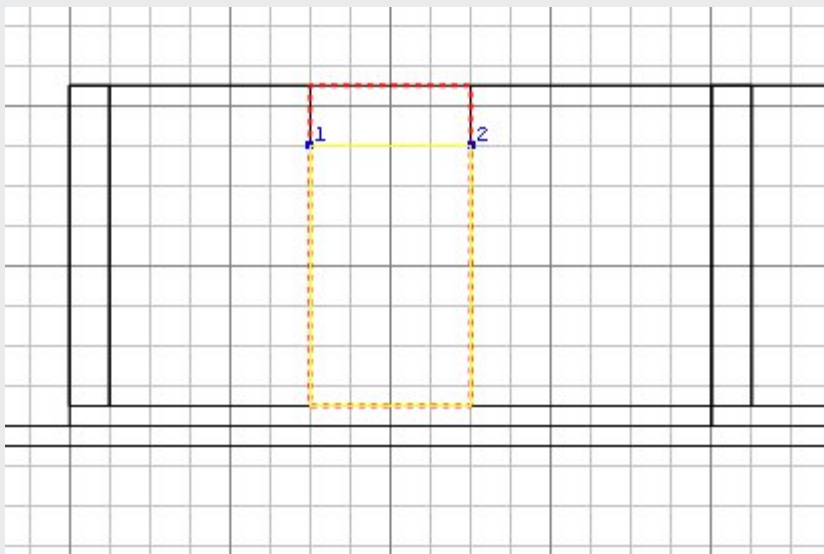
Press shift+return to divide the brush into 2.

In the 3D window, click on the **smaller** chunk to deselect it, leaving the other selected and the clipper active. Then define another cut as shown.



Press shift+return to divide the brush into 2.

In the 3D window, click on the **larger** chunk to deselect it, leaving the other selected and the clipper active. Then define another cut as shown.



This time we want to make an empty door space, but the yellow chunk is the wrong one to keep - so press ctrl+return to swap the yellow marker over, then press Return.

Press ESC twice now, as we are done clipping.

We have an opening. The faces of the opening are caulk so we must do something about it.

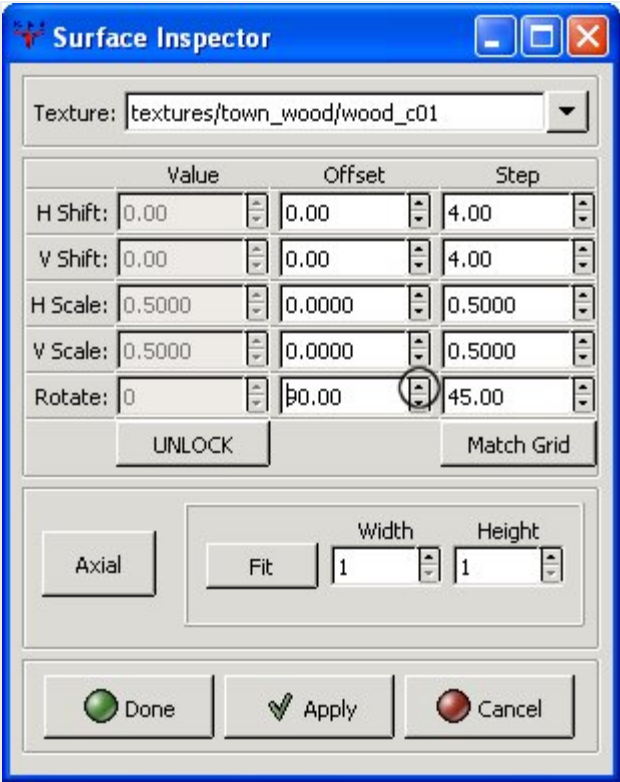
For now, we won't bother putting in a door frame, we'll be a bit quick and dirty and texture the visible caulk. This is slightly wasteful, because the two side faces extend up beside the small chunk of wall over the doorway. Later when we make the door, we'll fix this. As I said, sometimes it won't be worth the grief in trying to stop everything from being drawn if it won't be seen - but if you do this where practicable, you'll keep the FPS up and everyone playing will be grateful that your map doesn't play like running through custard :)

Select the 3 caulked faces that surround the door opening. Apply Textures/wood wood\_c01 and press ESC.



This leads us to another useful technique, that of being able to orientate a texture. We can see that the upright wooden texture looks fine, but the crossmember texture is aligned 90 degrees to what we really wanted.

Select the face with the wrongly aligned texture. Press **S** to get access to the Surface Inspector. Click the indicated up arrow twice to rotate the texture through 90 degrees and click Done.

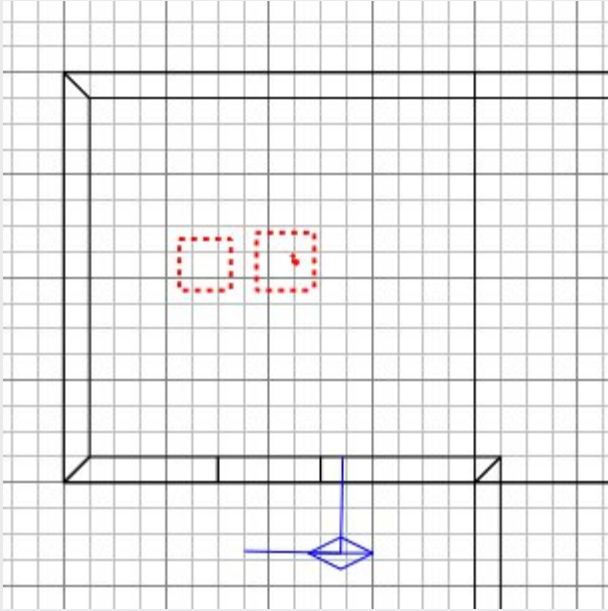


Press ESC to deselect the face. Looks good now :)

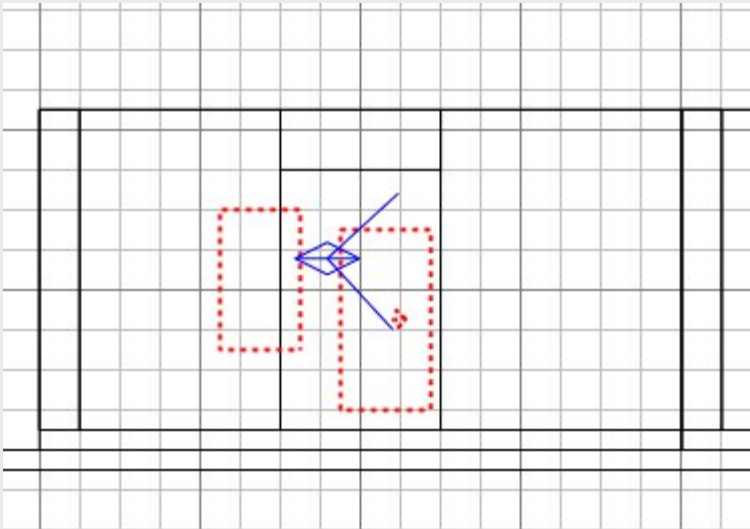


Probably about time for you to trial your creation so far. Let's put the allied start point indoors.  
Press ctrl+tab until you get the overhead view. Shift+click on the blue box and the yellow box behind it.

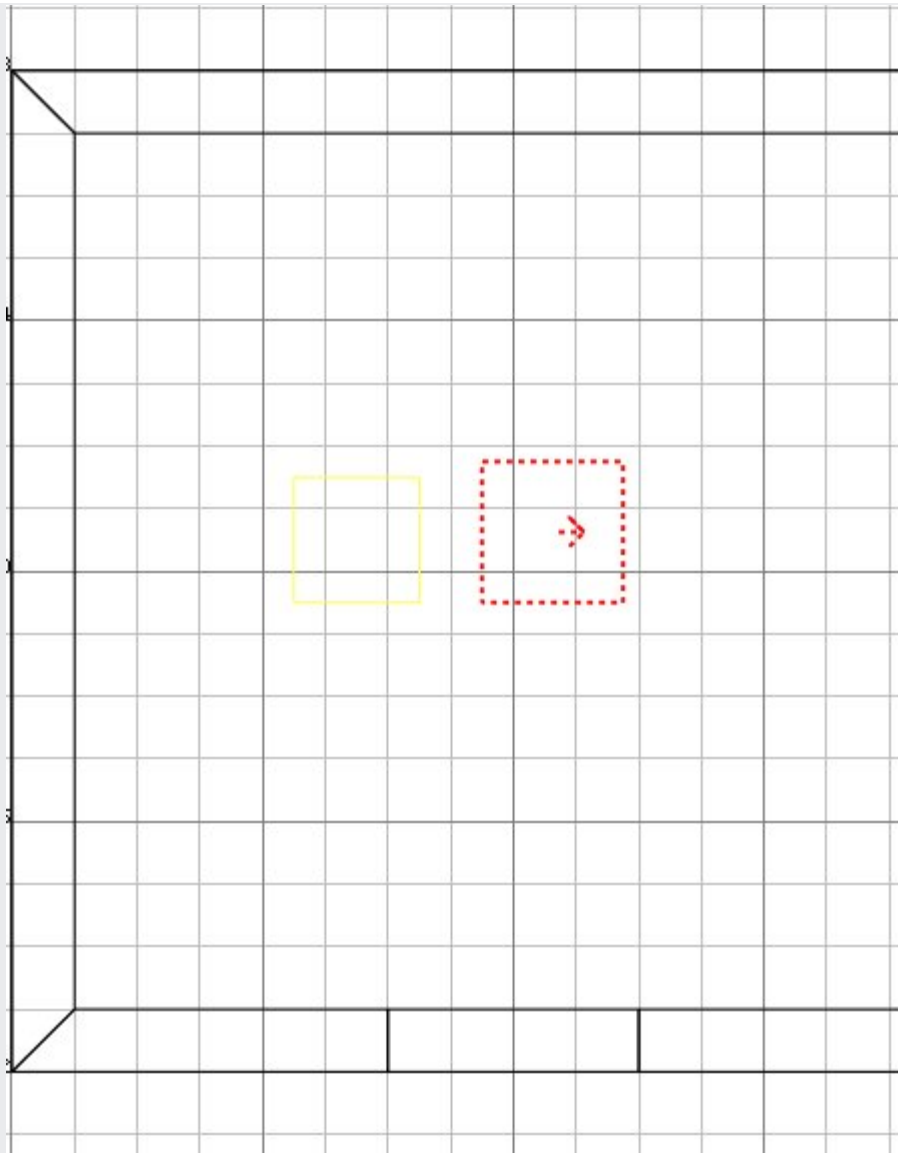
You can actually select entities this way even through intervening regular brushes, which is quite handy. Drag them into the room.



Press ctrl+tab and you will see the player start is buried in the ground. Lift the entities up a notch until the player is on or just above the floor.



Press ESC and then select the player start point alone. Press ctrl+tab to get the overhead view. If you zoom in you will see an arrow in the blue box. It indicates the direction the player will be looking in when he spawns.



Let's get him to face the door. Press **N**. At bottom left of the Entities window is a little cluster of boxes with degree angle numbers written in them. Their arrangement in the box indicates the direction the number represents. Click the 270 button.

Close the Entities window and press ESC - you can see now that the player will arrive facing the door.

Save your work, compile it and give it a little playtest. You should see something like this:



Notice that the indoor footsteps sound is different to the snowy. The tiles used don't actually have any special sound properties given to them (in the way that the snow texture has). Instead ET uses a default footstep sound for anything not coming with its own sound defined - and the default is fine for this sort of texture when walked on.

Well done if you've reached this far successfully. You've actually used many of the viewing/selecting/editing techniques and tools that will be your mainstay for much of your mapping career. We'll add more to them later on. In the next lesson we'll put in a ceiling then go on to put some lights inside the room, make a door and some windows we can shoot.

[Next lesson](#)



## Welcome to **thetibetclan.com**

- [Human Rights Tibet](#)
- [Free Tibet Shirt](#)
- [Tibet Chat](#)
- [Clan](#)
- [Tara Tibet](#)
- [Tibet Europe](#)
- [1959 Tibetan](#)
- [Tibetan News](#)
- [Chinese Authority](#)
- [Chinese Crackdown](#)
- [Chinese Military Force](#)



SEARCH

### Finance

- [Free Credit Report](#)
- [Car Insurance](#)
- [Credit Card](#)
- [Application](#)

### Dating

- [Online Personals](#)
- [Christian Singles](#)
- [Jewish Singles](#)

### Travel

- [Airline tickets](#)
- [Hotels](#)
- [Car rental](#)

### Home

- [Foreclosures](#)
- [Houses For Sale](#)
- [Mortgage](#)

**This domain may be for sale. [Buy this Domain](#)**