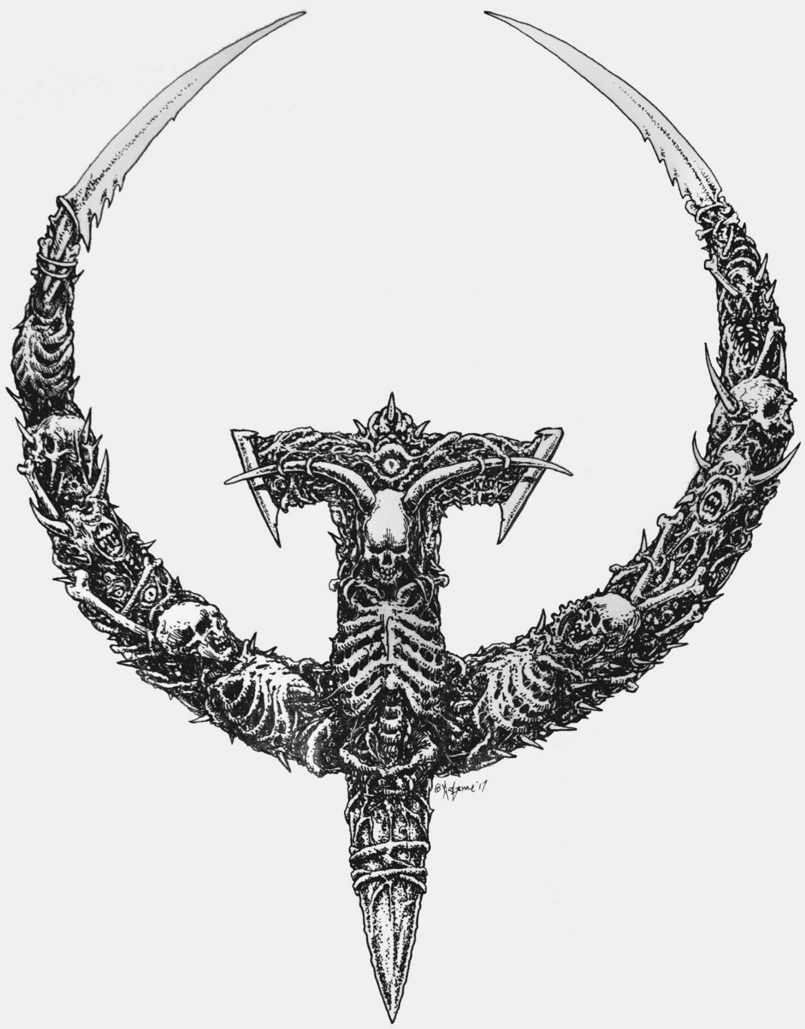


progs_dump

a devkit for QUAKE

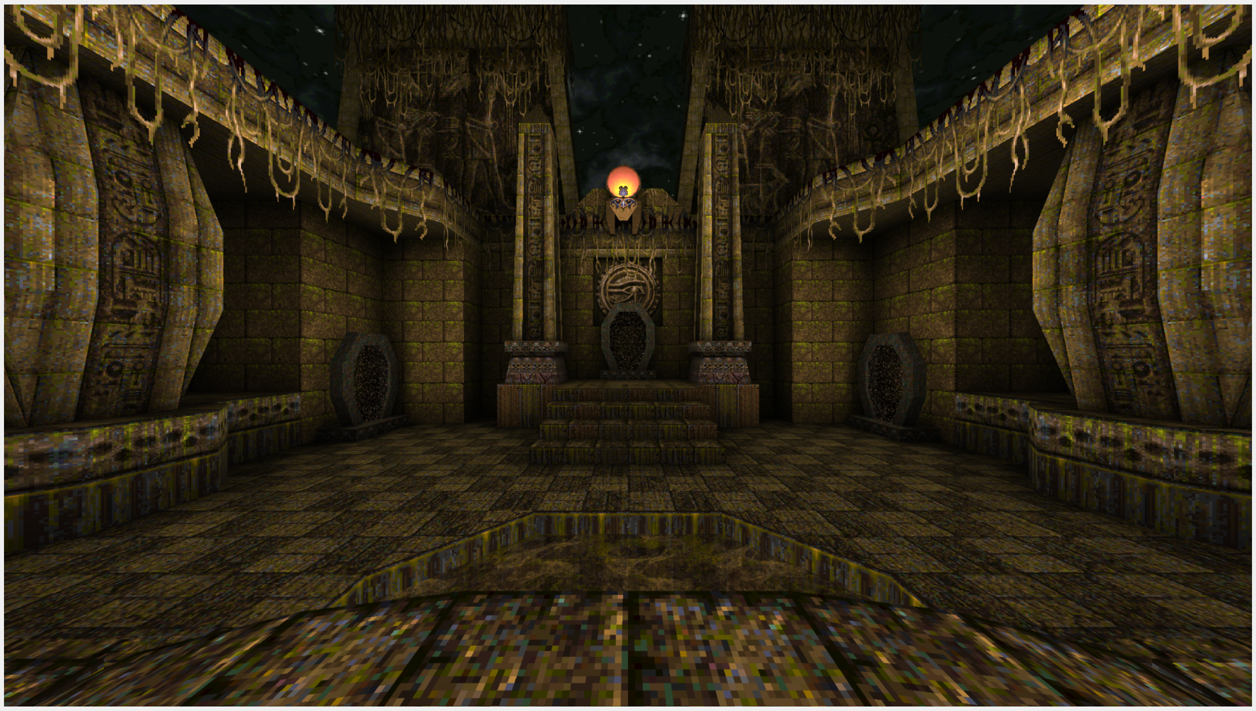
Version 3.0.0
27 September 2022

Contents

What is progs_dump?

progs_dump is a development kit for id software's Quake. Its purpose is to give mappers more creative options and "quality-of-life" improvements over the original "vanilla" version of the game. The *progs_dump* dev kit has dozens of unique and powerful features that are explained in this manual, in the FGDs and in the included sample maps.

The best way to get an overview of all these features is to load up the start map and play through the short sample levels. Only a few of these have monsters and not *every* feature is shown but the majority are.



Your project should be released as a stand-alone mod and installed into its own folder in the Quake directory NOT in the progs_dump folder.

The devkit consists of two elements:

First, there's the *progs_dump* "mod" folder. This holds all the sample maps, mapping assets, source code and the documentation you are reading now.

The second element is the *my_mod* folder inside the *mod_template.zip* file. This is a streamlined version of *progs_dump* with just the assets you need to make your mod.

The workflow is simple:

Use the *progs_dump* mod as a reference and learning tool, then create your own Quake mod with the *my_mod* folder as a base.

What can you do with progs_dump?

The devkit started as a simple project to add custom sounds and models to the game but has grown into a powerful toolkit aimed at beginner and intermediate mappers. Most features are from existing mods both old and recent, but there is a lot of new and unique code as well.

Features include:

Monster Customization:

No other Quake mod allows this amount of customization in such an easy way. Add custom sounds, skins, models, health, damage, names, obituaries and much more without any coding required. This includes customizing monsters' models, gibs and projectiles. Grunts, Enforcers, Shamblers, Deathknights and Ogres have multiple new attack options and we've added killable, gibbable versions of the original Quake bosses as well. Rotfish even gib.

Quality-of-Life Features:

Trigger spawned monsters, continuous monster spawning and random monster spawning. Respawn items and suspend them in mid-air. Add custom backpack pickups, drag and drop gore decorations and create visual effects like explosions and lightning effects. Custom models, sprites and sound effects. Multiple targets and targetnames, dormant triggers, enhanced platforms and more.

Unique new features like *trigger_look*, *sight_trigger*, *pain_target*, Doom-style door behaviors and *item_key_custom*.

Mission pack additions like custom gravity triggers, rotating entities, candles and proper elevators.

Enhanced teleporters with random destinations, monster only options, changeable destinations and more.

Popular requests like ladders, cutscenes and breakables are included. In fact, there are two styles of breakable. An "easy" method and a completely "custom" method.

Collisions for most objects are disabled in noclip making testing and reviewing your level a bit easier.

Bug fixes

Traditional fixes to the Shambler's collision during combat, the Rotfish "kill count" bug, door unlock sounds and many more "under-the-hood" code fixes. This includes fixes to the mission packs QuakeC as well. In version 3.0.0 we've included most of the Quake Info Pool fixes as well.

Credits & Acknowledgements

Programming:

iw
bmFbr
ILike80sRock
dumptruck_ds

Additional programming:

NullPointPalidin
TheRektafire
whirledtsar
TheSolipsist
JaycieErysdren
paavohuhtala
Zungrysoft
Inky

New models and new skins:

starshipwaters

Thanks to the following people for their assistance and generosity. We could not have compiled this mod without their guidance either directly, through tutorials, mapping, code, comments or forum posts:

Kebby, Jpal, Inky, ryanscissorhands, onetruepurple, Qmaster, RennyC, c0burn, ydrol, Preach, Joshua Skelton, Spike, Khreathor, Shamblernaut, ericw, metlslime, necros, negke, Baker, sock, G1ftmacher, NewHouse, Joel B, iJed, ionous, Yoder, Danz, thoth, vbs, Lunaran, Voidforce, NullPointPaladin, Twitchy, Paril, fairweather, shinola, SunkPer, KONair, xaGe, seven, Greenwood, hemebond, and many others on the Quake Mapping Discord and func_msgboard.

We also want to thank Pinchy, Mugwump, Len and PalmliX for their help with bug hunting in the early days.

A special thank you to Ian "iw" Walshaw for his excellent advice, coding, detailed comments and for fixing a massive list of bugs starting with version 1.1.1

Simply put, progs_dump would not have been as stable or ambitious without him.

Apologies if we're forgetting anyone else who assisted! Please let me know.

You can inquire about *progs_dump* on the progs_dump dev Discord. Check out dumptruck's Quake videos including the *progs_dump* playlist.

Recommended Quake Engines

progs_dump requires a modern Quake engine and was developed with Quakespasm

---

Recommended Quake Source Ports:

Quakespasm (0.95.0 and above)
https://sourceforge.net/projects/quakespasm/files/
http://quakespasm.sourceforge.net/download.htm

Ironwail (0.6.0 and above)
https://github.com/andrei-drexler/ironwail/releases

vkQuake (1.20.3 or above)
https://github.com/Novum/vkQuake/releases

Quakex
(a.k.a. Quake Enhanced, Quake Remastered and KEX Quake)

Steam
https://store.steampowered.com/app/2310/QUAKE/

Epic Games Store
https://store.epicgames.com/en-US/p/quake

Other source ports:

Quakespasm-Spiked (22-August-11 and above)
http://triptohell.info/moodles/qss/

FTEQW (Revision 6282 and above)
https://www.fteqw.org/

We no longer recommend Darkplaces or MarkV for progs_dump although many features will still work. These engines are untested and unsupported.

Installation

Do not copy new versions of progs_dump over existing installations. It's always best to make a new folder and move any work-in-progress maps and assets there.

1. Unzip the *progs_dump* archive into your Quake folder. This will create a *pd_300* folder inside. This directory will be a learning tool and reference for the features of the dev kit. Play it like any other Quake mod using the start map to explore a hub with sample maps.

   The development folder contains the FGD and DEF files that allow JACK, TrenchBroom and other editors to use the features of the devkit. Please refer to your map editor documentation for information on how to load mods and FGD files. In addition, there is a wad file that you can use to load the textures used in the sample maps. The QuakeC source code is included as well.

   Please read *progs_dump-3.0.0-README.txt* for important info and any last minute changes.

2. When you are ready to create your own mod, unzip the *mod_template.zip* into your Quake directory and rename the *my_mod* folder to the name of your mod (lowercase with no spaces). This folder is a stripped down version of *progs_dump* without the sample maps and other files. However, the new models, sounds, sprites, progs.dat and QuakeC source code are included.

3. When you are ready to release your mod, zip up your mod directory and make sure to include the *progs_dump-devkit-readme.txt* file and the QuakeC source folders in your release. If you modify the QuakeC code, make sure and include that version in your zip.

   Remove any cfg files, screenshots or save game files before zipping up your mod folder!

4. Please **do not** include the original *progs_dump* sample maps in your mod. But feel free to use the entity setups from the samples and prefab maps in your own projects. e.g. particle effects or custom monster entities.

5. Make sure and share your work on the progs_dump dev Discord.

   Good luck and happy modding!

Development Notes

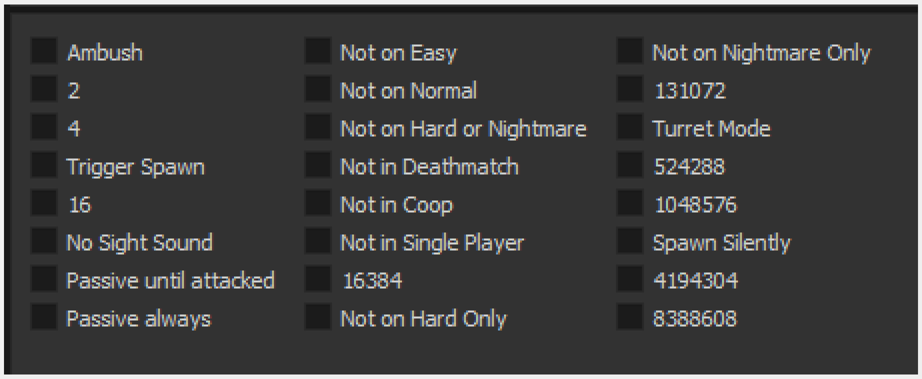
- You can launch Quake with *-nomonsters* on the command line to disable monsters in your map for testing. You can also set *nomonsters* to 1 in the console and enter *restart* to relaunch your current map without monsters.
- To check what version of the dev kit you are running, enter *impulse 100* in the console.
- While using the *noclip* command, you will not collide with item pickups and many triggers.
- The cheat code *impulse 9* will give the player any custom keys used in the current map in addition to all ammo, weapons and standard keys.
- Triggers that feature the *message* key have a new spawnflag *Message All Players*. This is useful for co-op gameplay when it's important that every player get the message.
- The FGDs are not included in the *my_mod* folder since they aren't required to play your mod. You can find the FGD and DEF files in the development folder. For more info on loading FGDs, refer to the documentation for your editor of choice.
- The included maps prepended with *pd_* and *prefab_* should be enough to demonstrate the features of the devkit. However, I have zipped up most of the other test maps made during development. These can be downloaded here. Please refer to the readme for more info.

Spawnflags

Trigger Spawned Monsters

The most requested feature of any general purpose Quake mod is trigger spawned monsters. This makes spawning monsters much easier than in the original game. All you need to do is select the *Trigger Spawn* flag and target the monster with any trigger when you want them to appear. *Ambush* (from vanilla Quake), will prevent the selected monster from being “awakened” by other monsters nearby. Errant gunfire or seeing the player will wake them up. *No Sight Sound* will suppress the monster’s “wake up” sound. (e.g. a Shambler will not roar when it sees the player.) You can make passive monsters that will never respond to the player’s actions by selecting *Passive always* or allow them to wake up and fight with the *Passive until attacked* spawnflag.

Starting in version 3.0.0 use the *Spawn Silently* flag to suppress visual and sound effects when a monster trigger spawns. In previous versions, this was enabled with the *wait 1* key | value pair. This will still work but has been removed from the FGDs.



☐ Ambush	☐ Not on Easy	☐ Not on Nightmare Only
☐ 2	☐ Not on Normal	☐ 131072
☐ 4	☐ Not on Hard or Nightmare	☐ Turret Mode
☐ Trigger Spawn	☐ Not in Deathmatch	☐ 524288
☐ 16	☐ Not in Coop	☐ 1048576
☐ No Sight Sound	☐ Not in Single Player	☐ Spawn Silently
☐ Passive until attacked	☐ 16384	☐ 4194304
☐ Passive always	☐ Not on Hard Only	☐ 8388608

Appearance Flags

Nearly every entity in the devkit has an expanded set of “Appearflags” compared to vanilla Quake. These new flags allow you to customize what shows up in a specific mode of the game.

4096	Not in Coop
8192	Not in Single Player
32768	Not on Hard Only
65536	Not on Nightmare Only

Spawnflag 16384 is not used here because it's already used for something else in *progs_dump*.

The new spawnflags complement and complete the set of built-in spawnflags provided by the engine, which of course are:

256	Not on Easy
512	Not on Normal
1024	Not on Hard or Nightmare
2048	Not in Deathmatch

In conjunction with the old spawnflags, the new spawnflags make it possible to exclude any entity from any combination of game modes and/or skill levels.

Not in Coop* and *Not in Single Player

These spawnflags were inspired by Quoth 2 (Kell and Necros, 2008), which included two additional spawnflags for all entities: *Not in Coop* and *Coop Only*. In contrast to Quoth 2, the spawnflags implemented here are *Not in Coop* and *Not in Single Player*, for symmetry with the built-in Not in Deathmatch spawnflag.

Not on Hard Only* and *Not on Nightmare Only

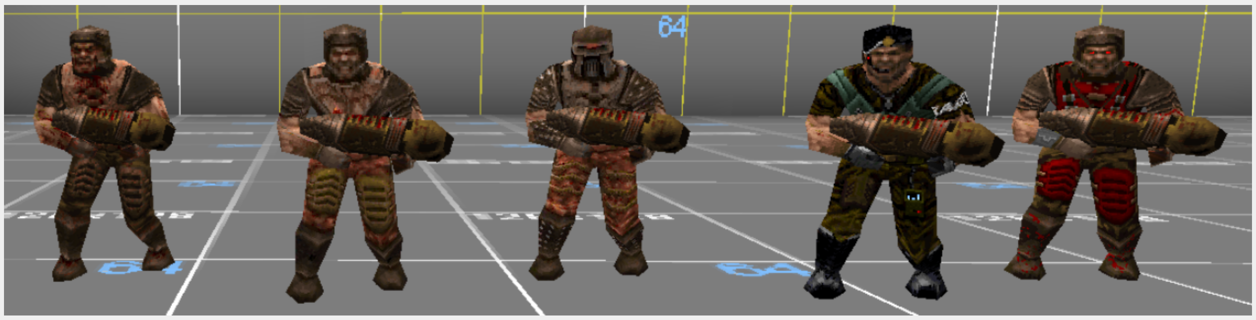
The set of built-in spawnflags doesn't allow a mapper to treat the Hard and Nightmare skill levels differently, because it only includes one spawnflag, 1024, which excludes an entity from both Hard and Nightmare. The new *Not on Hard Only* and *Not on Nightmare Only* spawnflags allow the mapper to exclude an entity from one of these skill levels without affecting the other. The original spawnflag will supersede the new flags.

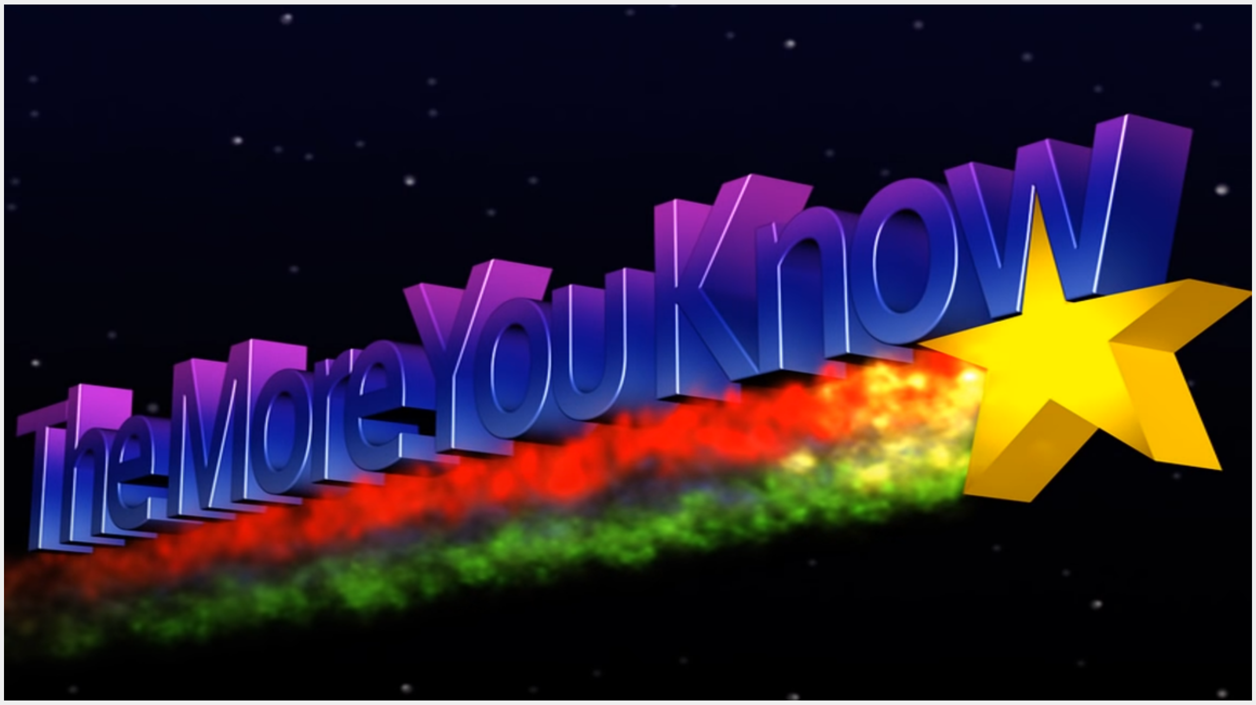
Monsters

Some of the new features in progs_dump can drastically change the way the game plays. Always use proper game design principles and communicate to the player that there is something different than they might expect.

Starting with version 2.x.x, *progs_dump* focuses heavily on monster customization. There are a lot of new key | values that can seem overwhelming at first glance. To make things easier to digest, the new features can be broken down into three categories: behavior mods, models and sounds.

When creating a custom monster, think of it as giving that monster a costume. You change their appearance with a compatible model and / or a skin. Change their "voice" with new sounds. Then disguise their attacks with projectile models, sound effects and behavior modifiers like custom projectile speeds and damage modifiers. There are dozens of replacement monster models, skins and sounds from various Quake mods dating back nearly 25 years. Info on where to find them is in Appendix B.





It’s important to note the limitations of monster customization before getting started:

1. To use a custom monster model, it must include the same number of animation frames in the same order as the monster you are using as a base. Don’t worry though, there are many “replacement” models and skins available. See Appendix B for info.
2. Custom monster sounds should be roughly the same duration as the original monster sounds you are using as a base.
3. The “rate of fire” for a custom monster cannot be changed but the damage dealt and projectile speeds can be! Also, all monster projectiles can be set to *homing* to behave like Vore Balls.
4. The Shambler’s lightning bolts cannot be replaced.
5. *progs_dump* only comes with a handful of built-in models to keep the distribution size to a minimum. You will have to provide custom models yourself.

For more info, check out our sections on built-in assets and where to find custom models.

Even with these limitations, you can create a large variety of monsters that feel unique. Also, *progs_dump* has a simple “plug-in” system where you will be able to download pre-made monsters to add to your mods.

Behavior Modifiers

Use the following key | values to change health, damage, spawn times, visual effects and more.

Key	**Details**
berserk	Skips certain pain animations similar to skill 3 Makes a semi-nightmare monster! e.g. The Enforcer will not stumble after taking damage. • 0 Off (Default) • 1 Berserk (skip pain animations) Excludes Bosses, Zombies and Spawn.
damage_mod	Multiply all damage from this monster by this number (e.g. 4 = Quad damage, 0.5 = half damage)
delay	The *delay* key allows you to add a custom delay to each trigger spawn. Normally, multiple targets will spawn simultaneously. If you want to stagger the time each monster enters the map, add a delay. Use the drop down menu to select some predefined values or enter a custom value in seconds if you need a specific time set.
drop_item	• 0 (Default) Disabled • 1 Drop a Silver Key upon death • 2 Drop a Gold Key upon death • 3 Drop a Health Vial upon death • 4 Drop a Armor Shard upon death • 5 Drop one vial and one shard • 6 Drop a random combination of 3 vials and/or shards Optional: Use *keep_ammo* 1 on Grunts, Enforcers or Ogres when this is enabled.
effects	Add a visual effect to an entity • 0 None (Default) • 1 Brightfield (yellow particles) • 4 Bright light • 8 Dim light
health	Monsters can have custom *health* levels.
homing	Enables and acts as a multiplier for homing on non-grenade projectiles. 1 is the maximum (see below for details)
waitmin	When set, homing factor will begin to increase after this many seconds (see below for details)
infight_mode	See more info below

keep_ammo	Stop Ogres, Grunts and Enforcers from dropping backpack ammo by setting to 1. Covered in this video.
obit_name	Custom description of WHO killed the player. When used with obit_method, this will set part of the text for a custom obituary. e.g. a Super Soldier!
obit_method	Custom description of HOW this monster killed the player. When used with obit_name, will set part of the text for a custom obituary. e.g. eviscerated - If empty, defaults to killed. Using the examples above, the obituary would read: “Player was eviscerated by a Super Soldier!”
pain_target	see description below
pain_threshold	see description below
proj_speed_mod	Multiplier for custom projectile speed (see below)
sight_trigger	Set *sight_trigger* to 1 to have monsters trigger targets when they see the player. This means they can not trigger an event upon their death. You can still use *pain_targets*. Covered in this video.
spawn_angry	Only when trigger spawned: • 0 default behavior - not angry • 1 set to 1 to spawn angry at player

When using *drop_items* with keys, take care that the key is accessible by the player when it spawns. Avoid placing monsters near lava or a void where the key could be lost or break the player’s progression in other ways.

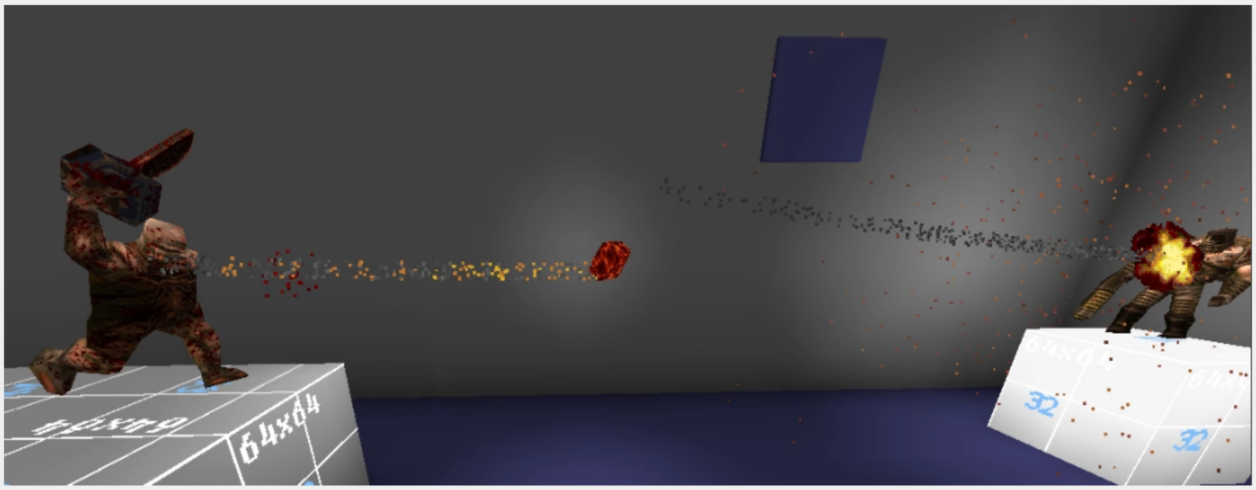
Infighting System

Monsters have an *infight_mode* key that affects how they will act toward other monsters. Now mappers have much more control and can even trigger specific fights using a *misc_infight* entity. Keep in mind that in the original game Grunts (monster_army) always infight and that is still the default in *progs_dump*.

Selection	Details
-1	Never infight
0	Default behavior, only infight with different classnames
1	Infight with monsters with the same classname but different *mdl_body*
2	Infight with monsters with the same classname but different *skin*
3	Infight no matter what

misc_infight

This point entity allows a mapper to pit one specific monster against another. You can only use this with a pair of monsters.



Key	Details
target	The monster who will get angry
target2	Who *target* will get angry at
Spawnflag	**Details**
1: Mutual hate	By default, the infighting does not start mutually, that is, *target2* will only get mad at *target* after it has been attacked. If you want them both angry when the *misc_infight* is triggered then use this spawnflag.
2: Keep player as activator	Allows the player to be the activator of anything the fighting monsters do. Useful for cutscenes.

Custom Projectile Speed

Starting in version 3.0.0, the speed of a monster's projectile (not including grenades) can be modified. Also, these projectiles can be set to *homing* and the turning rate can be changed.

Similar to *damage_mod*, the *proj_speed_mod* key will change the speed of a monster's projectile attack. For example, you can increase the speed of the Enforcer's lasers by setting *proj_speed_mod* to 2. Likewise, you can slow it down by setting this to 0.5. The upper limit differs for each projectile type. The max speed is 2000 Quake units per second. So for instance, Voreballs max out at 8 and nails at 2.

Homing Projectiles

Also, starting in version 3.0.0, **most** monster projectiles (excluding grenades) can be set to *homing*. Any non-zero value from 0.1 to 1 will enable *homing* for the applicable projectiles. In the QuakeC weapons.qc file, ILike80sRock explains how homing works:

"This finds a vector somewhere between the vector the projectile is currently traveling on and the vector it would normally snap to for homing. If you set homing to .25 it will go 25% toward the new direction, but keep 75% of the original vector, resulting in a wider turning range."

Lower numbers will mean a wider homing range but *homing* set to 1 would mean traveling in a straight line. Additionally, setting the *waitmin* key will increase the *homing* factor by .05 over time after that many seconds. So for example, consider a style 1 Grunt that shoots rockets with its *homing* key set to .25. If you set *waitmin* to 3 seconds, after that amount of time the rocket's homing range will increase by .05 every time it recalculates the direction. This means it will increase the accuracy over time after that delay, and reach its target more quickly.

To disable homing on a Shalrath, set its *homing* key to *-1.* This is *only* applicable to the Sharath.

Homing and speed modifiers can change the core gameplay of Quake. As always, use proper game design theory to communicate any drastic changes to the player so these unique attacks aren't a total surprise. You can do this with on-screen text or by isolating the modified monster in a mini-boss arena or some other unique setup.

Here's a list of monsters that can use these speed and homing features:

monster_army (rockets)
monster_army (lasers)
monster_army (nails)
monster_enforcer (rockets)
monster_enforcer (lasers)
monster_enforcer (nails)
monster_hellknight
monster_shalrath
monster_wizard
monster_ogre (flak)
monster_ogre (sniper)
monster_ogre (lava balls)
monster_boss (Chthon)
monster_boss2 (killable Chthon)
monster_oldone (Shub-Niggurath)
monster_oldone2 (killableShub-Niggurath)

Custom Monster Models

In Quake it’s easy to replace a monster model. Simply place a different model in the correct directory with the same name as the monster you want to replace. The drawback is that *every* iteration of that monster will have that model in the game. Same with sounds and other assets.

In *progs_dump* you can use the key | value pairs below to load any compatible model for a given monster entity in your map. This allows you to mix and match monster types in the same project. Using this with custom health, damage, sounds and other behaviors, allows you to have a variety of monsters in your project without any coding required.

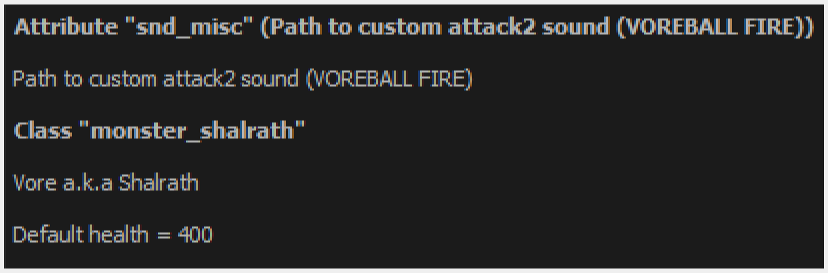
Not all of these key | values appear on each monster. For example, the Spawn has no head and therefore, no head model!

Key	Details
mdl_body	Path to custom body model e.g. dev/super_soldier.mdl
mdl_head	Path to custom head model
mdl_proj	Path to custom projectile model
skin	Skin index number of the body model if multiple built-in skins are included. Defaults to 0. NOTE: If the wrong number is entered here the model will not display in TrenchBroom
skin_head	Skin index number for head model if multiple built-in skins are included. Defaults to 0
skin_proj	Skin index number for projectile model if multiple built-in skins are included. Defaults to 0
mdl_gib1	Path to custom 1st gib model
mdl_gib2	Path to custom 2nd gib model
mdl_gib3	Path to custom 3rd gib model

You can use Quake compatible sprites and BSPs in addition to models!

Custom Monster Sounds

As with models, *progs_dump* allows you to replace the sounds a monster makes with custom audio. Most sounds can be replaced with some exceptions. Each monster entry in the FGD and DEF details any sounds that are not obvious with a hint in ALL CAPS. e.g. *snd_misc2* will replace the Enforcer's “HALT!” sight sound. You can see tips on how to “audition” existing sounds and models in the Custom Monster Example section below.



Attribute "snd_misc" (Path to custom attack2 sound (VOREBALL FIRE))

Path to custom attack2 sound (VOREBALL FIRE)

Class "monster_shalrath"

Vore a.k.a Shalrath

Default health = 400

Custom sound files used with these entities must be in the SOUND folder of your mod (or a sub folder under that SOUND folder.) There is no need to add “sound” in the path. (e.g. *boss2/sight.wav)* Most Quake source ports require a mono sound file for custom sounds. Do not use stereo files in your mod except for music.

Key	Details
snd_attack	Path to custom attack sound.
snd_death	Path to custom death sound.
snd_hit	Path to custom hit sound. e.g. laser hits wall
snd_idle	Path to custom idle sound.
snd_misc	Path to custom sound. Context will be different for various monsters or items. e.g. Enforcer’s “FREEZE” sight sound.
snd_misc1	same as above
snd_misc2	same as above
snd_misc3	same as above
snd_move	Path to custom sound for Chthon rising from lava.
snd_pain	Path to custom pain sound.
snd_sight	Path to custom sight sound.

pain_threshold
pain_target

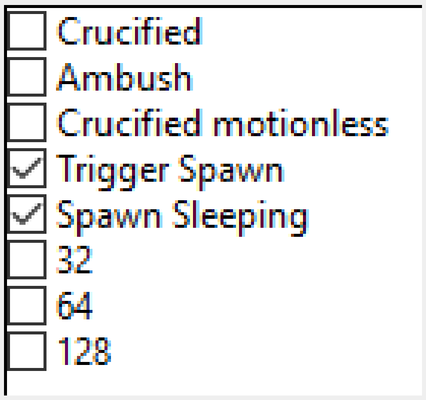
When a monster's health drops below its *pain_threshold*, its *pain_targets* are triggered. You can use this to call in reinforcements mid-battle or spawn items or fire other triggers when a monster reaches a certain level of health. You can also target things upon a monster's death, as always. Default values for monster health have been added to the FGD for reference. Check out this tutorial video to see this and other features in action.

monster_boss2
monster_oldone2

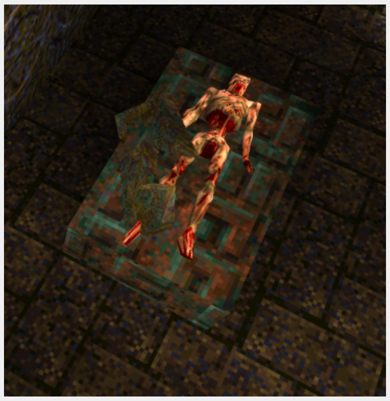
These are killable variants of the original boss monsters. On Skill 1 (Easy) these both have 1000 HP. On Normal, Hard and Nightmare the HP is set to 3000. You can also set a custom *heath* value, as with other monsters. Upon death, Shub will always gib but Chthon will only gib if his HP drops below -50 with one hit (Quad Damage, etc.)

Enhanced Zombies

Zombies have more options in *progs_dump*. First off, there are motionless, silent versions of the crucified "decorative" zombie. You can also create a *sleeping* zombie that will not awaken until triggered. You must target these zombies if the *Spawn Sleeping* spawnflag is set. If you trigger spawn a sleeping zombie into a map, you will have to target them a second time to "wake" them up. You can see examples of the new features in the *pd_zombies* sample map. Spawnflag examples:



- ☐ Crucified
- ☐ Ambush
- ☐ Crucified motionless
- ☑ Trigger Spawn
- ☑ Spawn Sleeping
- ☐ 32
- ☐ 64
- ☐ 128



Enhanced Hellknights

KA-BOOM! As of version 3.0.0, Hellknights now have spikes that can explode on contact using the *projexpl* key. There is also a new lightning style for Hellknights detailed below.

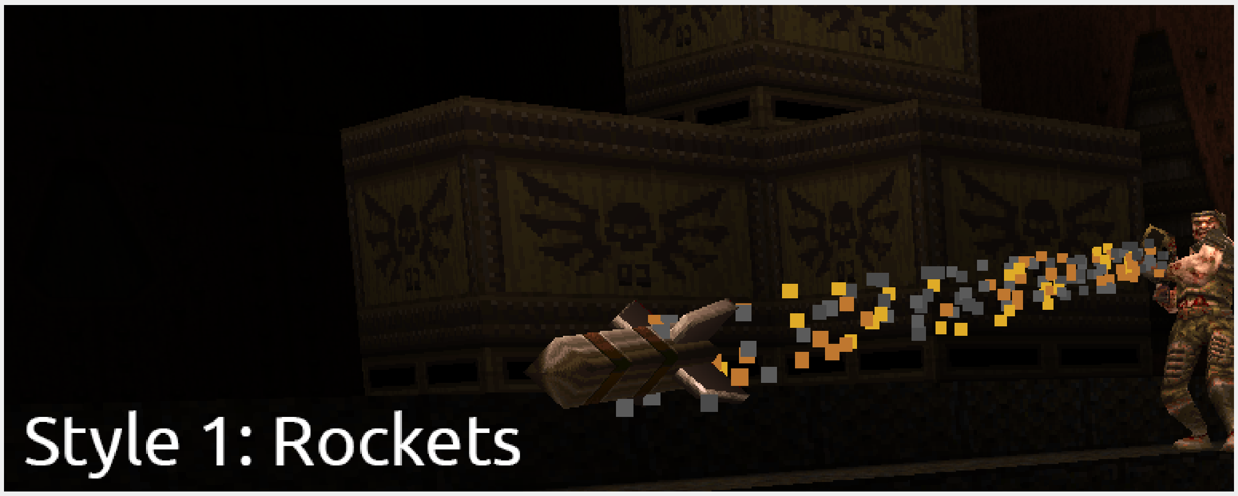
***projexpl* Selection**	**Details**
0	Default (no explosions)
1	All projectiles explode
2	Alternate between exploding and not
3	Projectiles randomly explode

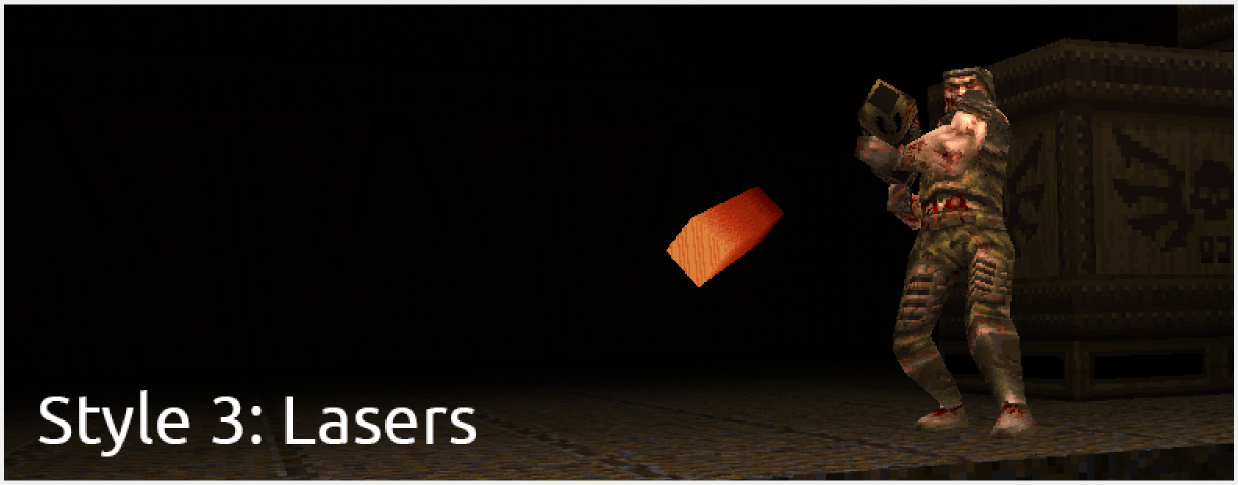
Monster Styles

This started out as a coding exercise but we decided to include it in *progs_dump* as part of monster customization. Grunts, Ogres, Shamblers, Hellknights and Enforcers have additional attacks that can be set via the *style* key. Of course, you can change their appearance and behaviors as explained above to make variants. e.g. You can replace the rocket projectiles with a custom sprite and add new sound effects to make a Grunt fire an explosive blast of energy.

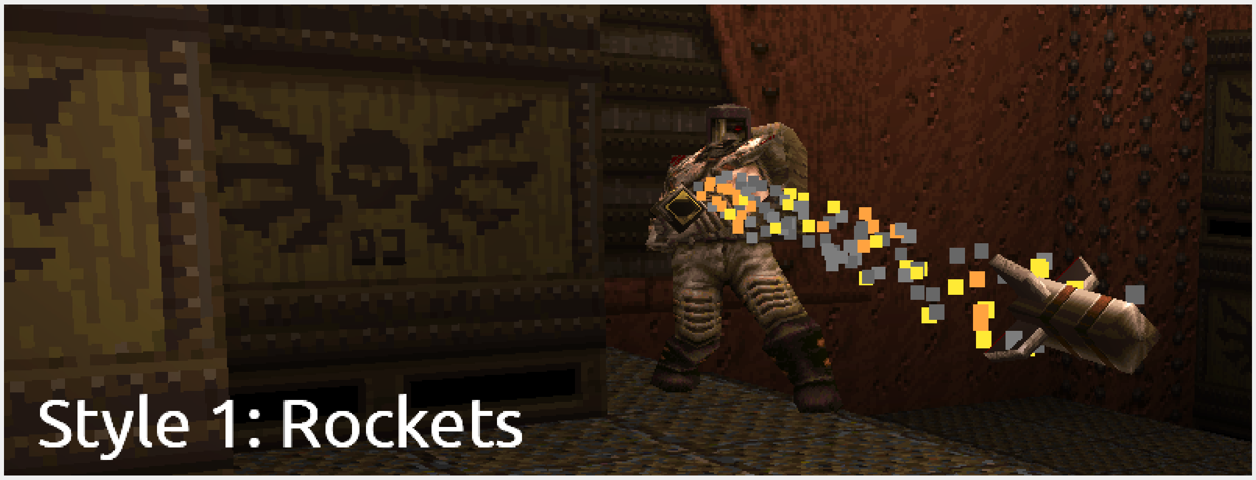
Grunt styles

Style	Description	Projectile Homing
0	Shotgun (default)	
1	Rockets	yes
2	Grenades	
3	Lasers	yes
4	Nails	yes
5	Nails (rapid fire)	yes





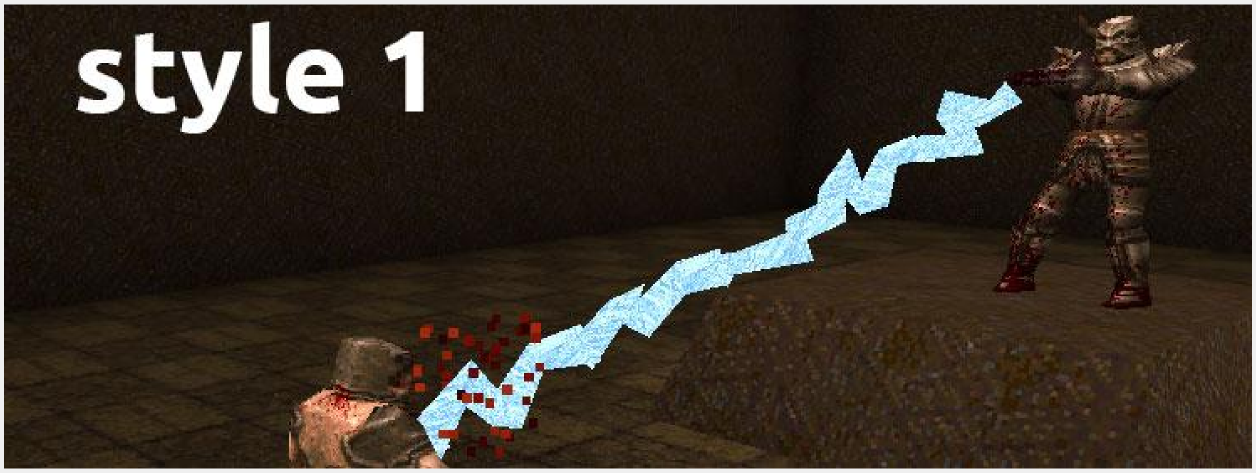
Enforcer Styles



Style	Description	Projectile Homing
0	Lasers (default)	yes
1	Rockets	yes
2	Grenades	
3	Nails (single fire)	yes
4	Lightning	
5	Nails (rapid fire)	yes

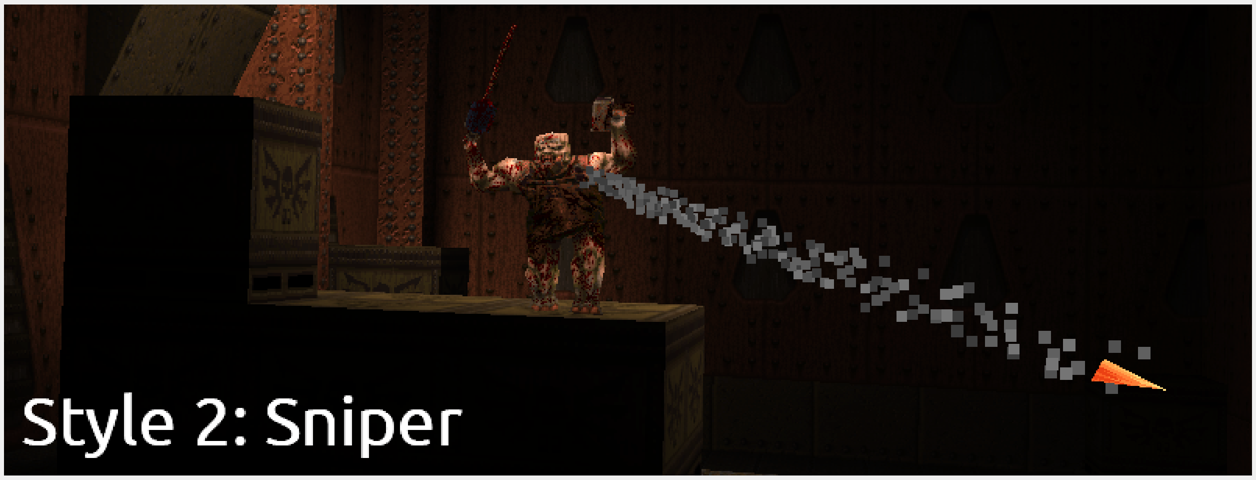
As with the Grunt and Ogre you can use the *mdl_* and *snd_* keys to replace projectiles, head, body, sounds and skins to create variations of the Enforcer.

Hellknight Styles

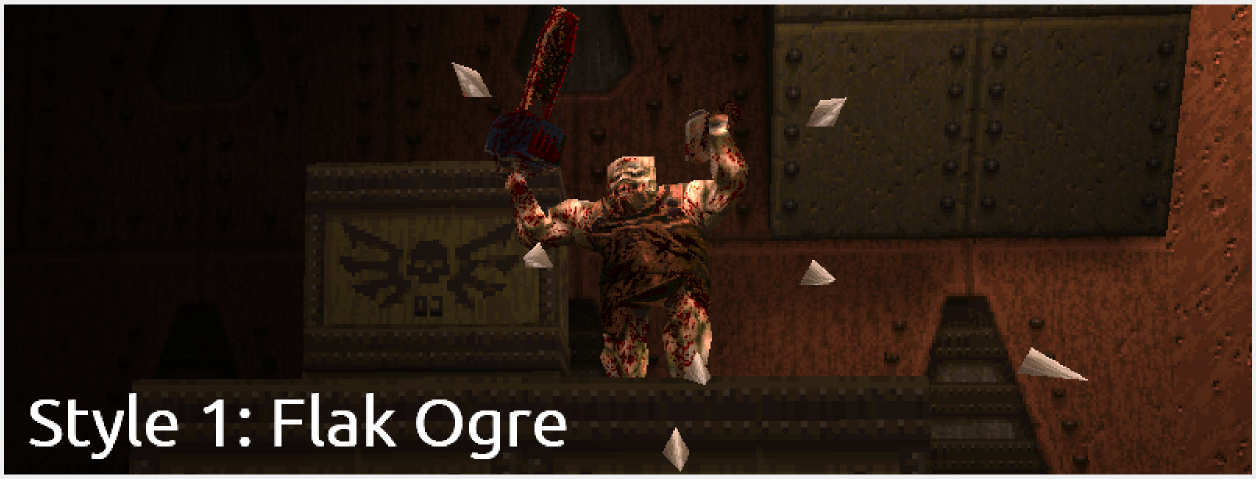


Style	Description	Projectile Homing
0	Default	yes
1	Lightning	

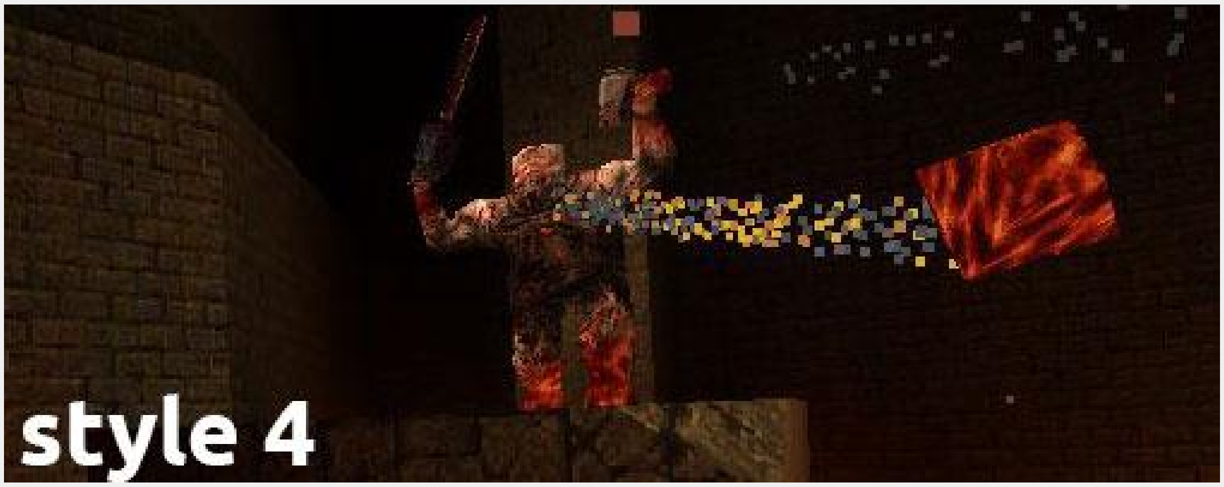
Ogre styles



Style	Description	Projectile Homing
0	Grenades (default)	
1	Flak Ogre (also seen in Quoth and The Marcher Fortress)	yes
2	Sniper. Shoots a single, deadly lava round.	yes
3	Multi-Grenade (Mission Pack 2)	
4	Lava Ogre (shoots Chthon’s exploding lavaballs)	yes



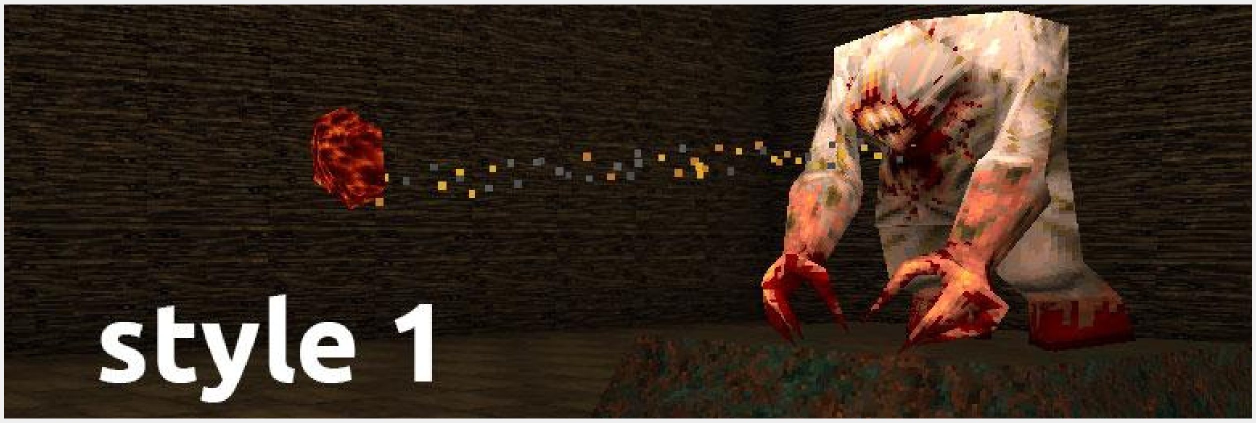
You cannot replace *mdl_proj* on an Ogre set to style 3 but all other projectiles and models can be replaced. If you prefer “lava spikes” for the Flak Ogre, set the Ogre’s *skin_proj* key to 1.



Style 4 Ogres shoot Chthon’s spinning, explosive lavaballs by default. As with most other styles, you can replace the projectile model with *mdl_proj* and the projectile’s skin with *skin_proj.* You can set the *cust_avelocity* key to a custom angle velocity (yaw, pitch and roll) to make the projectile rotate faster or slower on other axes. The default is *200 100 300*. Or you can set this to *0 0 1* to disable spinning, so you can use other projectile models like rockets or spikes that shouldn’t rotate but still explode on contact. You can set the lava ball to home in on targets with the *homing* key (described above). By default, two sounds are played when lava balls are launched: *snd_misc2* is the sound made when Shalraths launch voreballs. And *snd_attack* is the whoosh sound the lava ball makes while flying. Both of these can be customized. Check out the sample map *prefab_styles* to see this in-game.

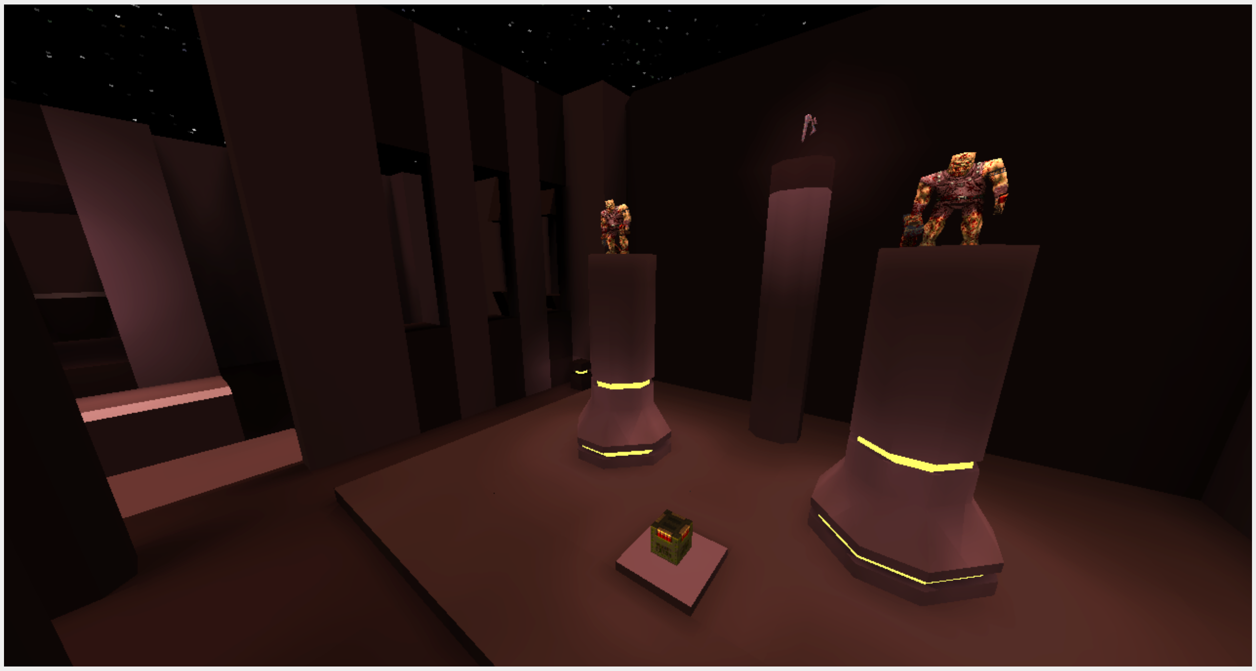
Shambler Styles

A Shambler set to Style 1 will throw Chthon’s spinning, explosive lavaballs. As with most other styles, you can replace the projectile model with *mdl_proj* and the projectile’s skin with *skin_proj.* You can set the *cust_avelocity* key to a custom angle velocity (yaw, pitch and roll) to make the projectile rotate faster or slower on other axes. The default is *200 100 300.* Or you can set this to *0 0 1* to disable spinning, so you can use other projectile models like rockets or spikes that shouldn’t rotate but still explode on contact. You can set the lava ball to home in on targets with the *homing* key (described above). By default, *snd_attack* is the whoosh sound the lava ball makes while flying. This can be customized. Check out the sample map *prefab_styles* to see this in-game.

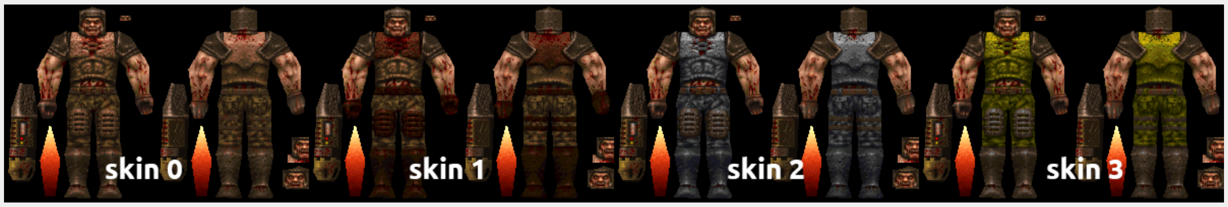


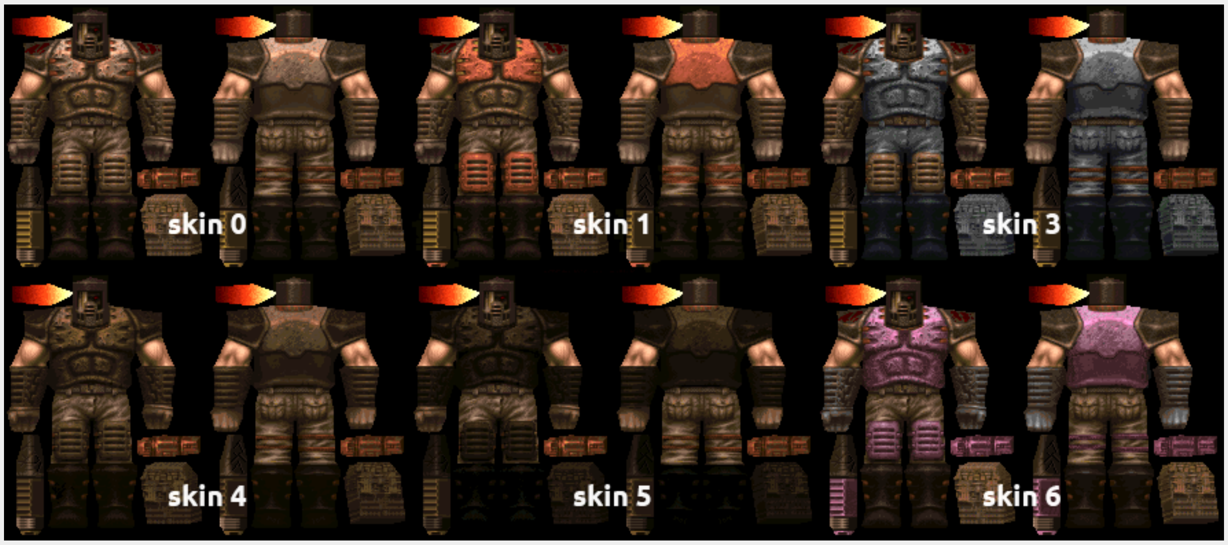
Turret Mode

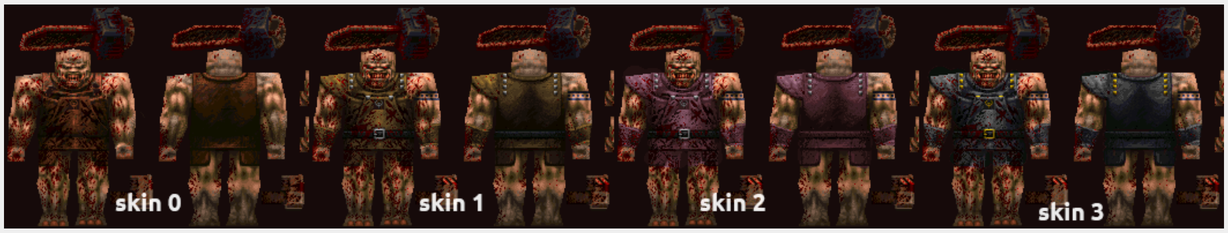
Grunts, Enforcers, Ogres, Ogre Marksmen, Deathknights, Shalrath, Shamblers and Zombies can be set to turret mode via a spawnflag. This includes *styled* versions like Rocket Grunts and so on. They will never pursue the player “on-foot”, so keep that in mind when placing them. Also, when the player is out of sight they will cease fire. Zombies and Ogre Marksman variants are partially z-aware. The Shambler’s lightning attack range is increased by 300 to account for vertical positions. Same with the Enforcer and Deathknight lightning styles. You can see turret mode in action in the *pd_turrets* sample map.

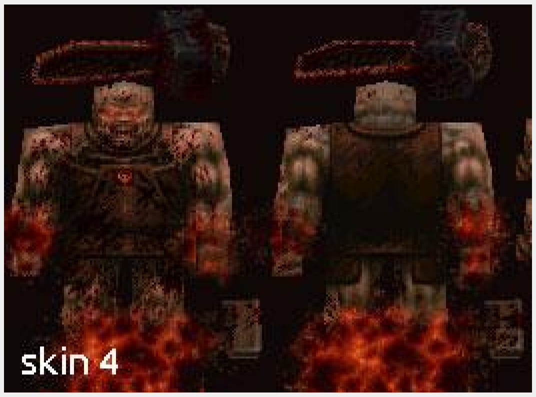


We've included models with a few simple custom skins for styled monsters that can be set in the *skin* key. But you can use your own choice of models and skins too.









func_monster_spawner

When activated, it spawns standard id1 monsters to its targeted *info_monster_spawnpoint*. The monster total is updated upon each spawn unless disabled by a spawnflag (see below).

Choose the *Style* of monster via dropdown. Refer to the FGD dropdown for choices. ***Style2* set to a value of 1 overrides *Style* and chooses a random monster**. *Count* is how many monsters to spawn in (default is 5). *Wait* is the default time between spawns (default is 5 seconds). *Berserk* can be set to 1 to skip pain animations. Can only use default health, models and sounds. New in *progs_dump* 3 are the following spawnflags:

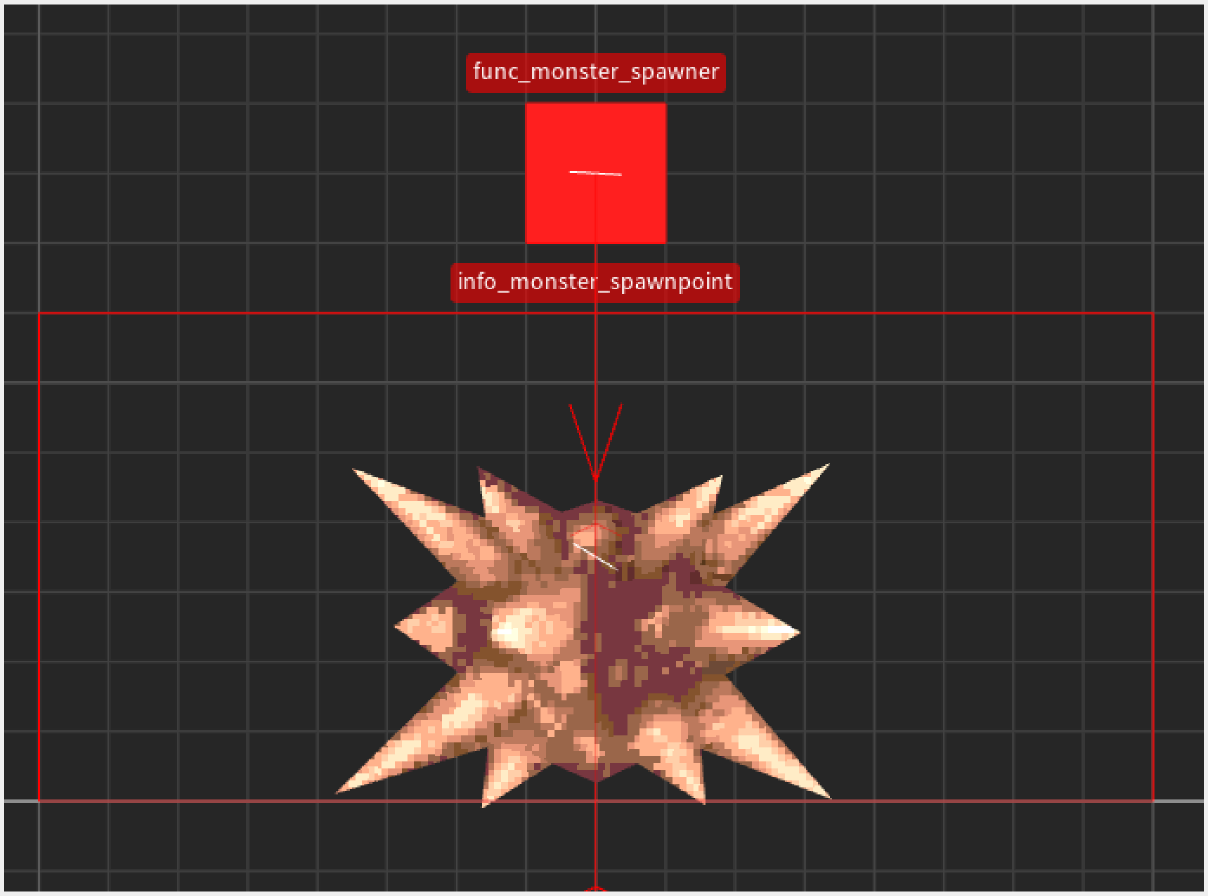
Spawnflag	**Details**
1: Reset after completion	Resets *count* and can be re-triggered again.
2: Don’t spawn angry	Will wait until angered to attack and move
3: Don’t add to count	Does not contribute to final monster kill count
32: Spawn Silently	No visual or sound effects when spawned

info_monster_spawnpoint

Destination for *func_monster_spawner*. Alternatively, you can use a *misc_teleporttrain* for a moving spawn point. See the next section for details.

A *func_monster_spawner* will wait to spawn until there is nothing that can take damage around a 128 unit radius. The *info_monster _spawnpoint* only sets the position for the spawn, the *func_monster_spawner* determines if a monster is in range. **So ensure both entities are close together as pictured below.** You cannot use multiple *func_monster_spawners* with one spawn point.

Because monster bounding boxes are so varied, it's best to pay attention to the larger bounding box of the *info_monster_spawnpoint*. Keep it clear of doors, buttons and other geometry so your monsters have some room to enter this dimension!



misc_teleporttrain

This was used for the final boss level in the original game. In *progs_dump* you can use it as a moving decoration with a custom model or even target it as a spawn point for a *func_monster_spawner*.

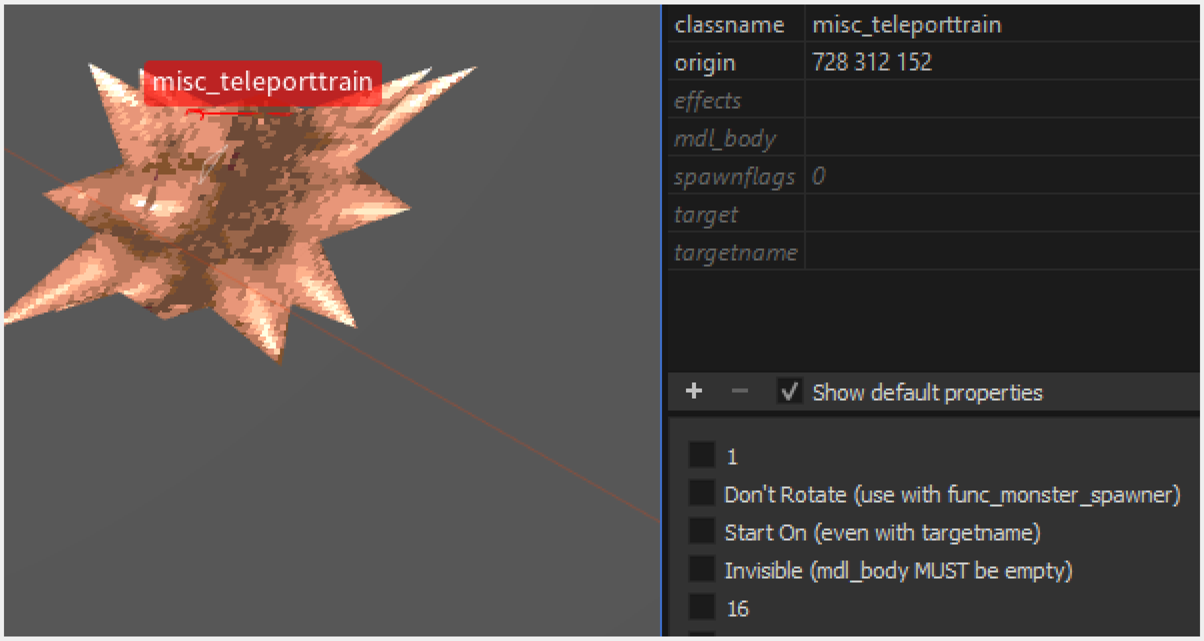
Make sure and select the *Don't Rotate* spawnflag when a *func_monster_spawner* is targeting a *misc_teleporttrain*, or you'll experience a pretty hilarious game breaking effect! You have been warned!

NOTE: This is a new model in *progs_dump* that replaces the original. You can use this new model as a decoration or custom projectile for other monsters. Take a look at the different skins here.

You set this up like a *func_train* using *path_corners*. By default, it will move automatically between path corners upon map load. However, you can have it wait to move by giving it a *targetname*. If you need to target it as a spawner and want it to move on map load, use the *Start On* spawnflag. Here's a video tutorial on how to use trains and *path_corners* in Quake.

You can add *effects* using the dropdown, use a custom model using the *mdl_body* keyvalue and even make it *Invisible* with spawnflag 8. For example, you can animate a moving light around a level using the dimlight effect with the invisible flag set.

When using this entity to spawn in monsters, it's best to not use it as a destination for the player to use. This can cause a crash under certain conditions.

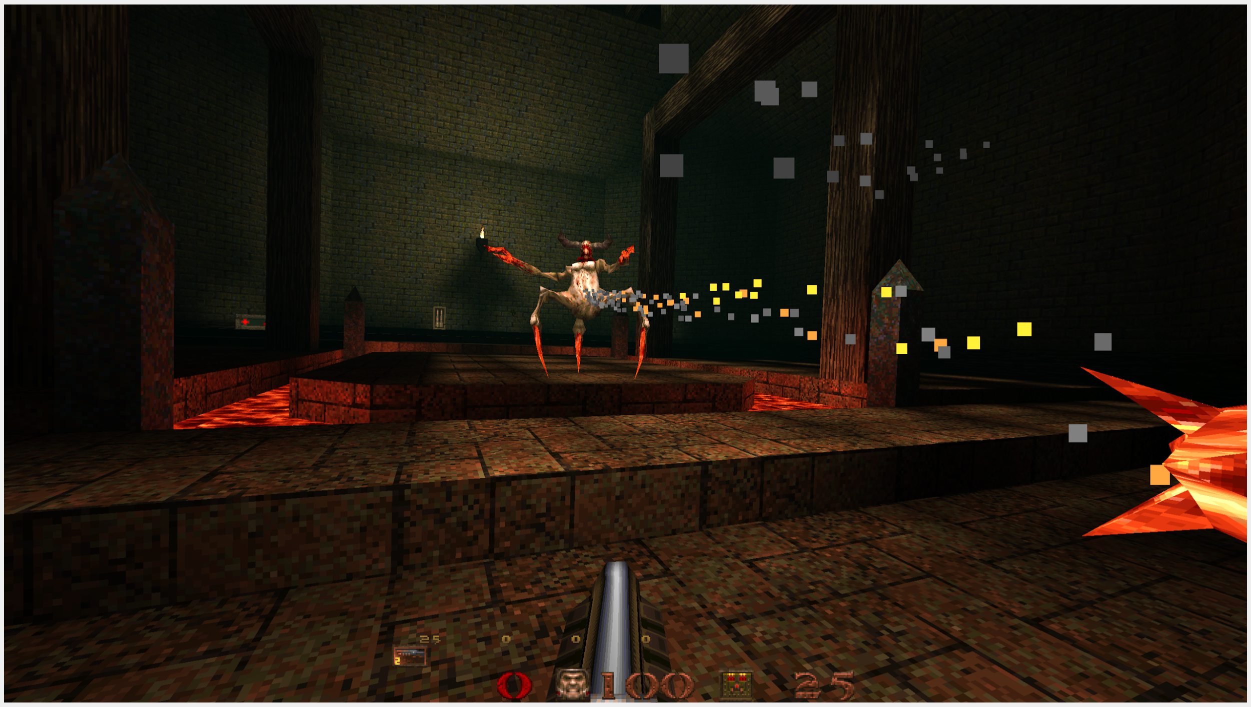


classname	misc_teleporttrain
origin	728 312 152
effects	
mdl_body	
spawnflags	0
target	
targetname	

- ☐ 1
- ☐ Don't Rotate (use with func_monster_spawner)
- ☐ Start On (even with targetname)
- ☐ Invisible (mdl_body MUST be empty)
- ☐ 16

Custom Monster Example

We've created a "plug-in" monster template for *progs_dump* that will make it easy to share or reuse custom monsters. The first example of this is the *Hellrath* which you can download here. This monster uses the *monster_shalrath* as a base and adds a replacement body model, projectile model, a new skin, new sounds and some modifications to health and damage to make a "new" mini-boss.

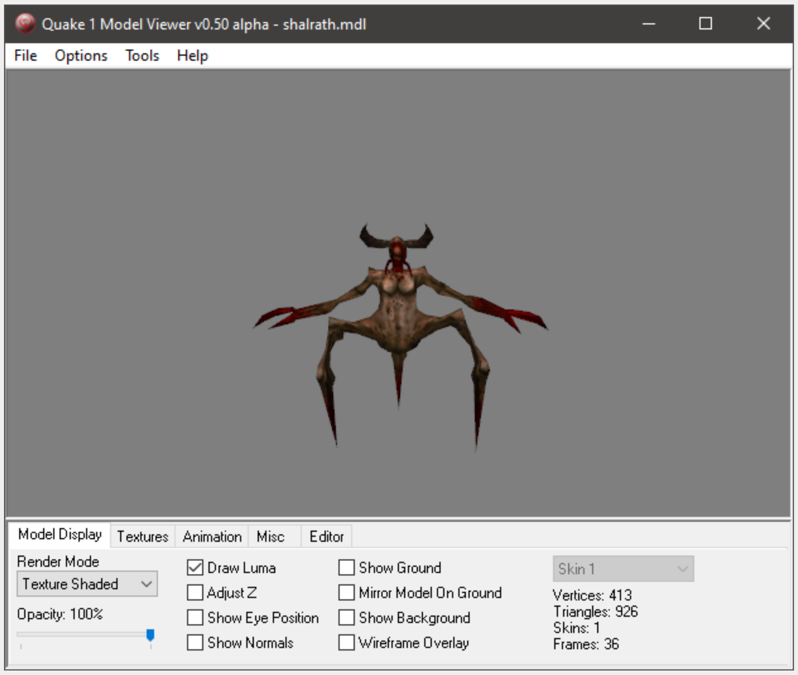


Make sure you credit the original creators of the assets you are using in the readme file for your mod. And of course, make sure you are free to modify and distribute their work.

In this section, we'll cover the basic steps we followed to get this working in *progs_dump* but we cannot go into detail on how to use each application. Some of the apps you can use are:

Quake 1 Model Viewer view models and animations, import and export skins
AdQuedit 1.3 Powerful app that allows editing of most Quake files formats
AdQuedit Manual
PakScape Browse PAK files and export their contents.
Quake Model Editor a.k.a QME
TexMex Texture manager
Wally texture editor
Wally Tutorial
Ocenaudio excellent cross-platform, donationware audio editor
Grafx2 cross-platform, free, 8 bit paint program (a bit clunky but great for quick edits)
Grafx2 Tutorial on YouTube
Links to 8 bit friendly graphic apps

The first step was choosing a custom Shalrath model created by “Chillo" as part of their replacement monster pack. You can see animation frames and export a .bmp skin file with the Quake 1 Model Viewer. Unfortunately, you can only overwrite the existing skin file. However, AdQuedit can add skin files without replacing the original. Note that AdQuedit uses the .pcx image format not .bmp.

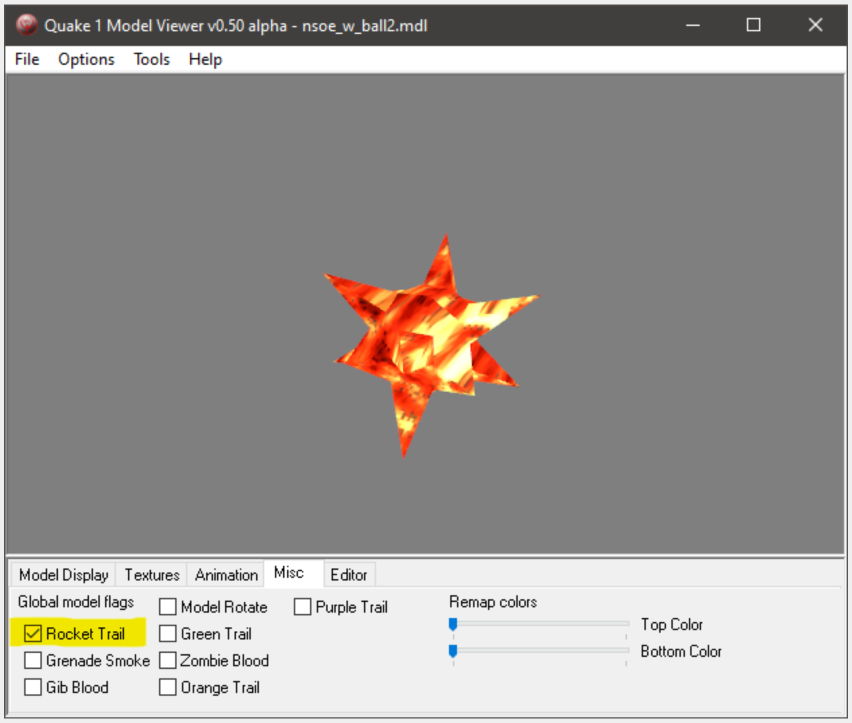


Render Mode: Texture Shaded
Opacity: 100%

[✓] Draw Luma
[ ] Adjust Z
[ ] Show Eye Position
[ ] Show Normals
[ ] Show Ground
[ ] Mirror Model On Ground
[ ] Show Background
[ ] Wireframe Overlay

Skin 1
Vertices: 413
Triangles: 926
Skins: 1
Frames: 36

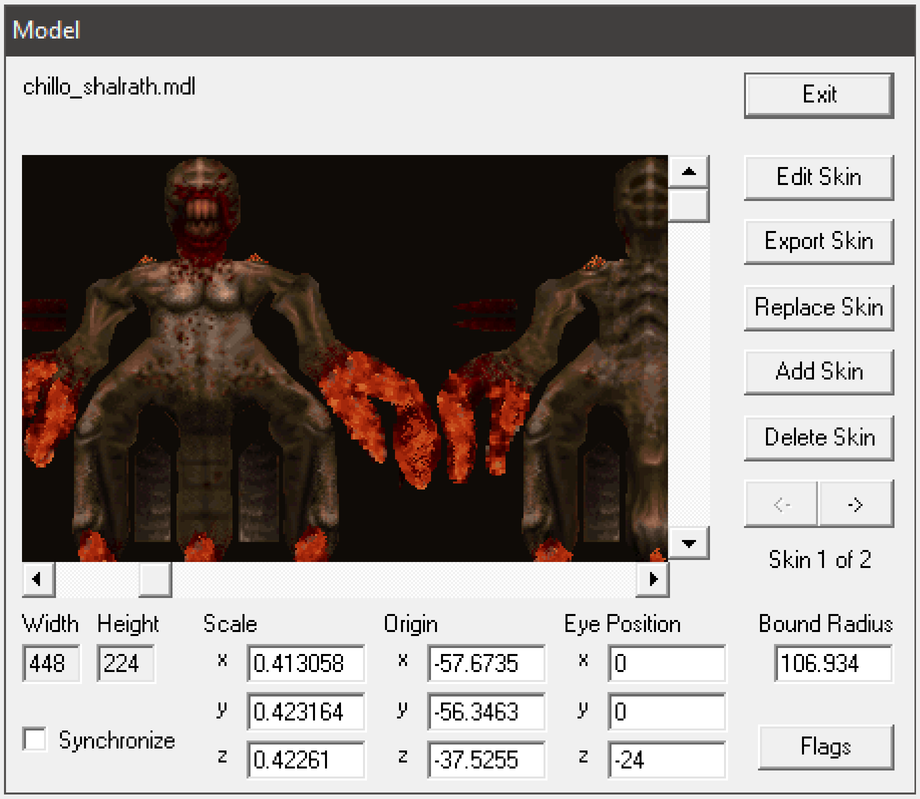
We decided this monster should fire lava projectiles instead of the standard “voreballs”. Using Quake 1 Model Viewer we changed a flag on the model to have a rocket trail. Here’s more info on using model flags.



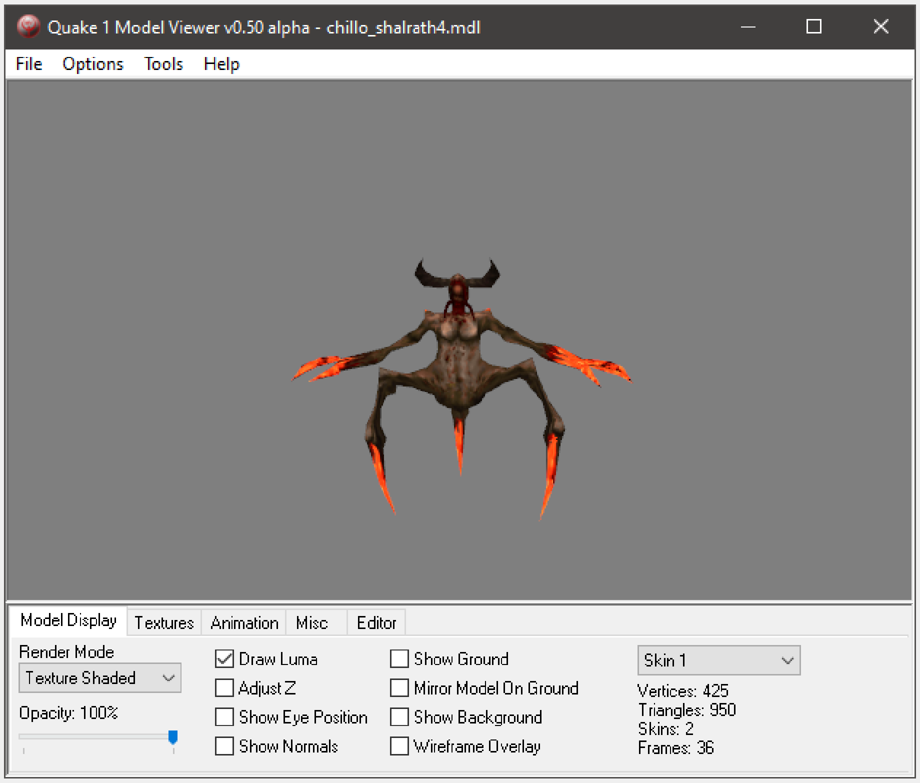
Global model flags
[✓] Rocket Trail
[ ] Grenade Smoke
[ ] Gib Blood
[ ] Model Rotate
[ ] Green Trail
[ ] Zombie Blood
[ ] Orange Trail
[ ] Purple Trail

Remap colors
Top Color
Bottom Color

Next we added a skin to the existing model. Using a paint program to add a lava texture to its limbs. We used AdQuedit to insert the .pcx format skin into the model.



Width	Height		Scale		Origin		Eye Position	Bound Radius
448	224	x	0.413058	x	-57.6735	x	0	106.934
		y	0.423164	y	-56.3463	y	0	
		z	0.42261	z	-37.5255	z	-24	



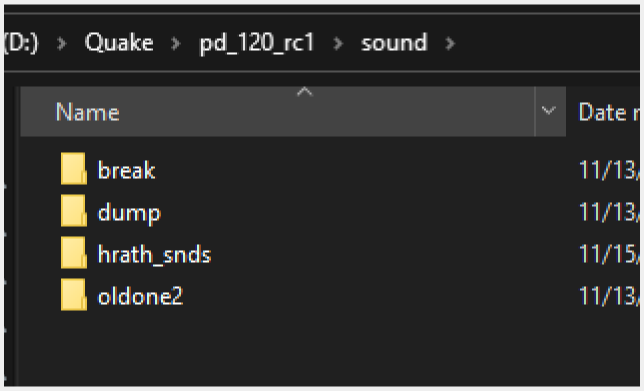
Render Mode: Texture Shaded
Opacity: 100%
☑ Draw Luma
☐ Adjust Z
☐ Show Eye Position
☐ Show Normals
☐ Show Ground
☐ Mirror Model On Ground
☐ Show Background
☐ Wireframe Overlay
Skin 1
Vertices: 425
Triangles: 950
Skins: 2
Frames: 36

Keep in mind when you are making skins they must be in the Quake palette. You can learn more about Quake's textures on Quakewiki.org and in the dumptruck_ds texture series on YouTube.

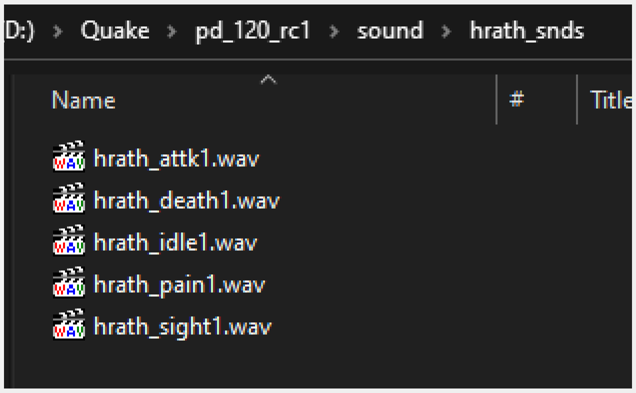
After that, we created some audio files to replace the standard Shalrath sounds. These must go into the sound directory of your mod. In this case, we named a subfolder *hrath_snds.*

The sounds should roughly be the same duration of the original monster sounds you are replacing or they might cut off in-game.

You can audition and extract sounds, models and other files using PakScape. Make sure and use a compatible sound format. 11k, 16bit mono was used for the Hellrath. More on that in the Custom Sounds section below.

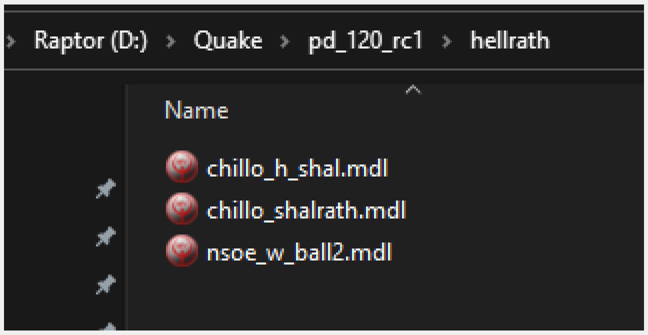

(D:) › Quake › pd_120_rc1 › sound ›

Name	Date r
break	11/13,
dump	11/13,
hrath_snds	11/15,
oldone2	11/13,


D:) › Quake › pd_120_rc1 › sound › hrath_snds

Name	#	Title
hrath_attk1.wav		
hrath_death1.wav		
hrath_idle1.wav		
hrath_pain1.wav		
hrath_sight1.wav		

The model files for *Hellrath* go in their own folder inside your mod folder. Usually Quake models are found under the progs folder. You can place your models there if desired, it will work either way.

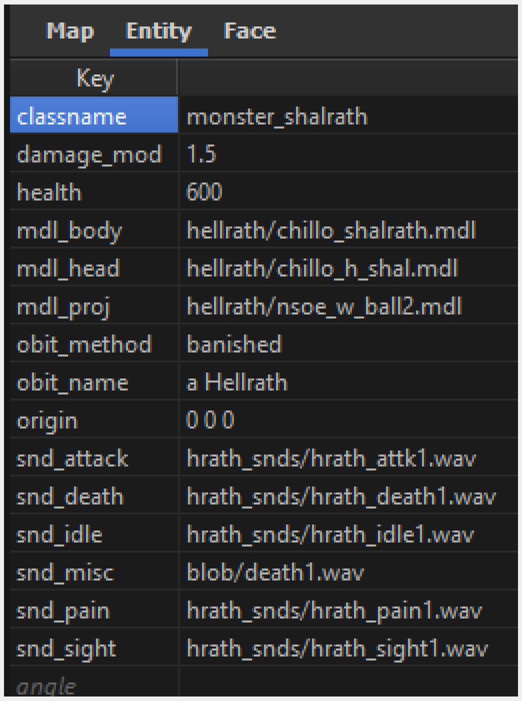

› Raptor (D:) › Quake › pd_120_rc1 › hellrath

Name
chillo_h_shal.mdl
chillo_shalrath.mdl
nsoe_w_ball2.mdl

Remember, model files go in their own folder (or inside the progs folder) and sounds **must** be in a folder inside the sound folder of your mod.

The next step is to add these paths to your monster in your map editor. The paths to the new sounds do not need to include *sound* in the value so you start instead with the *hrath_snds* folder as seen below. However, if the models are in the progs directory (not pictured) you need to add *progs* to that path.

The monster below does 1.5 times the damage of a Shalrath and has 600 health. When a player is killed by this monster the obituary will read: “Player was banished by a Hellrath”.



Key	
classname	monster_shalrath
damage_mod	1.5
health	600
mdl_body	hellrath/chillo_shalrath.mdl
mdl_head	hellrath/chillo_h_shal.mdl
mdl_proj	hellrath/nsoe_w_ball2.mdl
obit_method	banished
obit_name	a Hellrath
origin	0 0 0
snd_attack	hrath_snds/hrath_attk1.wav
snd_death	hrath_snds/hrath_death1.wav
snd_idle	hrath_snds/hrath_idle1.wav
snd_misc	blob/death1.wav
snd_pain	hrath_snds/hrath_pain1.wav
snd_sight	hrath_snds/hrath_sight1.wav
angle	

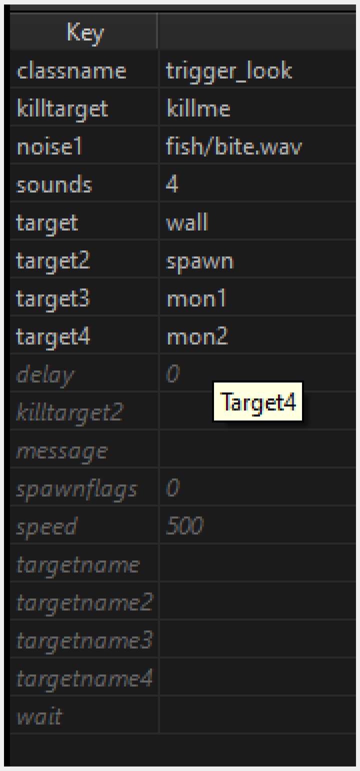
It’s really easy to make a typo as all these entries are done by hand. So if for some reason your monster isn’t working, **check your paths for typos or other mistakes.** Most of the time, this is the culprit.

The process for adding custom models for ammo, health and other items in the game is the same as above.

Multiple targets, targetnames and killtargets

Most entities can now trigger up to four separate targets at once (target, target2, target3 and target4). They can also have multiple targetnames (targetname, targetname2, targetname3 and targetname4). Mappers can also create setups with killtarget and killtarget2. In addition, mappers can use target and killtarget in the same entity. This is not possible in vanilla Quake.

Multiple triggers can be used in nearly any combination or order. For example: target3 can trigger targetname2 in a different entity.



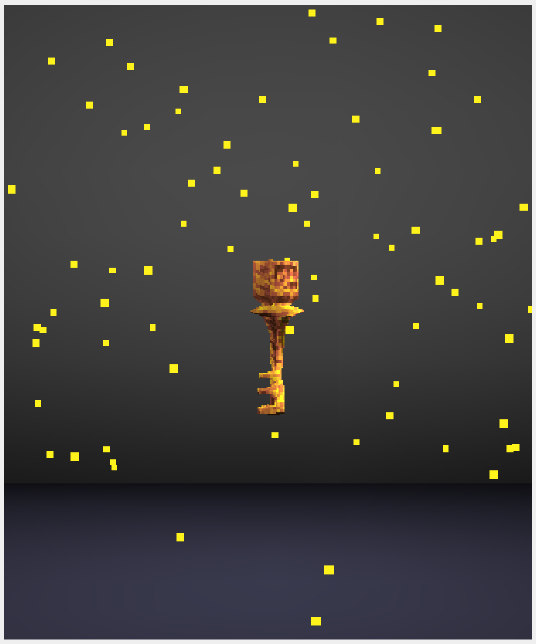
Key	
classname	trigger_look
killtarget	killme
noise1	fish/bite.wav
sounds	4
target	wall
target2	spawn
target3	mon1
target4	mon2
delay	0
killtarget2	
message	
spawnflags	0
speed	500
targetname	
targetname2	
targetname3	
targetname4	
wait	

IMPORTANT: When using path corners or other similar entities, use the primary *target* and *targetname* fields for navigation only. The additional numbered fields may not function as expected in these cases. The Quoth mod has the same feature and the rule of thumb applies here. As Preach states on the Quoth tutorial site: *A recommended structure is to use the original targetname field to give entities unique identifiers, and use the remaining fields for group triggers.*

Items

Most items have enhanced capabilities in *progs_dump*. This includes ammo, weapons, keys and power-ups. Items can be suspended in mid-air via a spawnflag or trigger spawned just like monsters. Set a *targetname* for the item and select the *Trigger Spawned* spawnflag. Like monsters, they can spawn silently or with the "t-fog" teleport visual and sound effects.

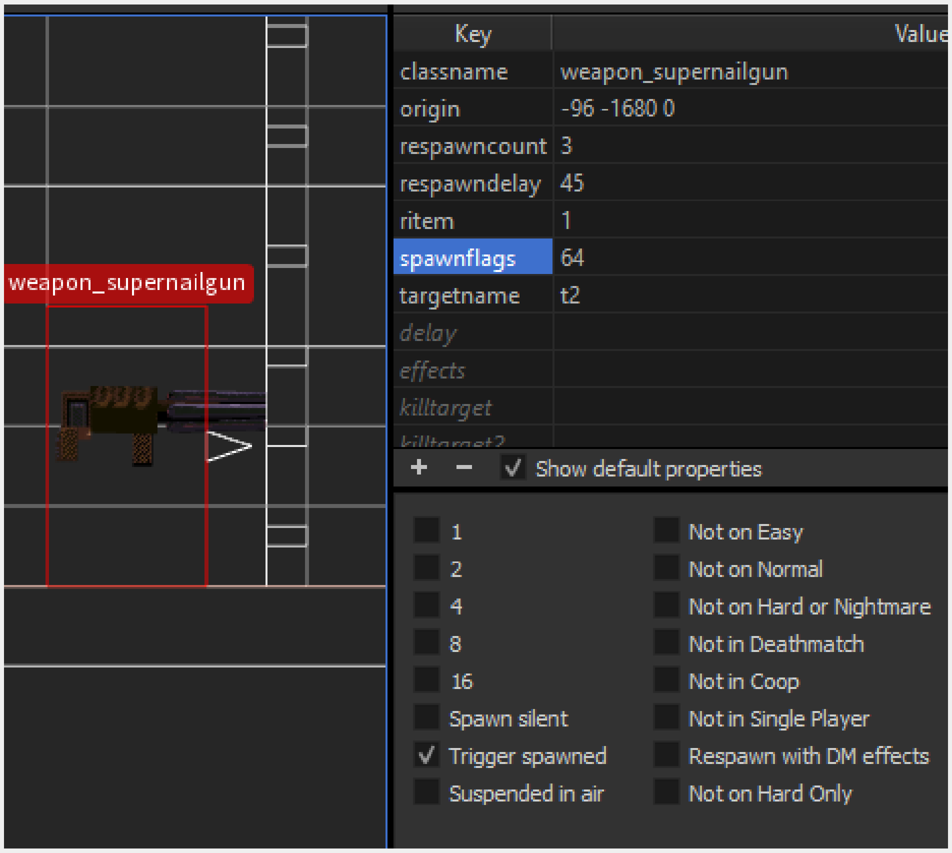
The *effects* key allows you to add some built-in visual effects to items. A drop down is available with brightfield (yellow particles), bright light and dim light effects. The *alpha* key controls the model transparency so you can have "ghostly" items or monsters.



Respawning Items

Most items can also be set to respawn. Setting the *ritem* key value to 1 will cause the items to respawn. Items will respawn based on the default time settings for a deathmatch game. You can set a custom respawn time using the *respawndelay* key and control how many times an item respawns with the *respawncount* key. By default, items will display the “t-fog” effects when respawning. You can mimic deathmatch respawns with the *Respawn with DM Effect* spawnflag. This skips the “t-fog” effect and plays a more subtle sound effect.

In the example below, the super nailgun is trigger spawned when the player presses a button targeting “t2”. After the player picks up the weapon it will respawn in 45 seconds but only 3 times.

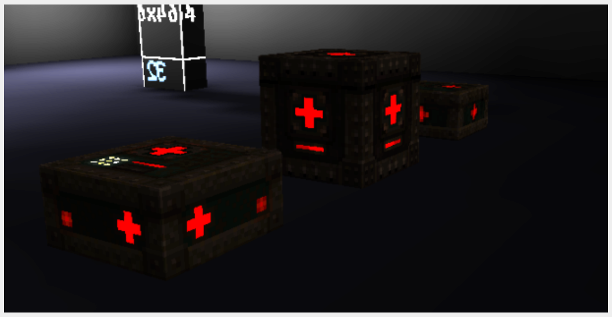


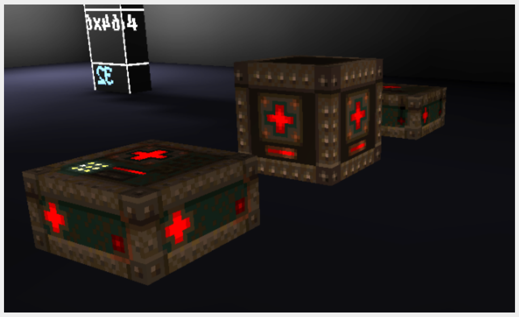
Key	Value
classname	weapon_supernailgun
origin	-96 -1680 0
respawncount	3
respawndelay	45
ritem	1
spawnflags	64
targetname	t2
delay	
effects	
killtarget	

☐ 1	☐ Not on Easy
☐ 2	☐ Not on Normal
☐ 4	☐ Not on Hard or Nightmare
☐ 8	☐ Not in Deathmatch
☐ 16	☐ Not in Coop
☐ Spawn silent	☐ Not in Single Player
✓ Trigger spawned	☐ Respawn with DM effects
☐ Suspended in air	☐ Not on Hard Only

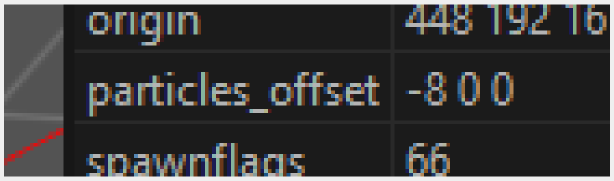
Custom Item Models

Similar to monsters; health, ammo, armor, artifacts and other items in the game can use custom models. Also, for health and ammo boxes, mappers can choose the original .bsp models or .mdl versions by setting the *worldspawn* entity *style* key to 1. Mdls will "accept" the light on the brush directly beneath them. Bsp models are "pre-lit" at a set value. Using .mdls can make for more realistic lighting in your maps. Many thanks to Lunaran for sharing these mdls from his excellent Copper mod! Of course, you can use any model for these by setting the *mdl_body* key as with other entities.

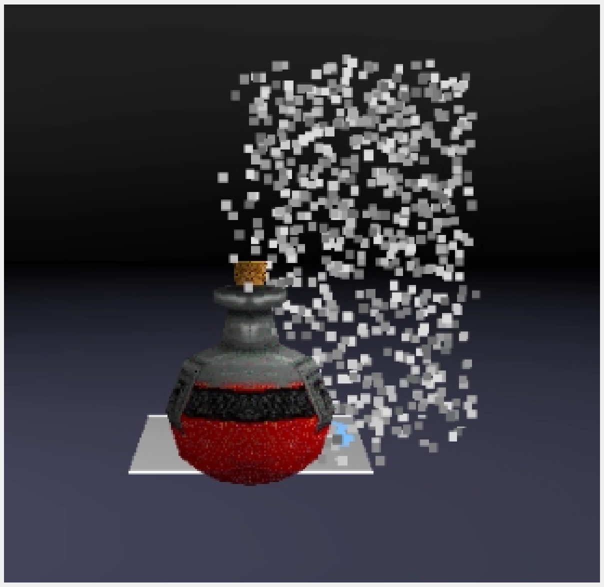




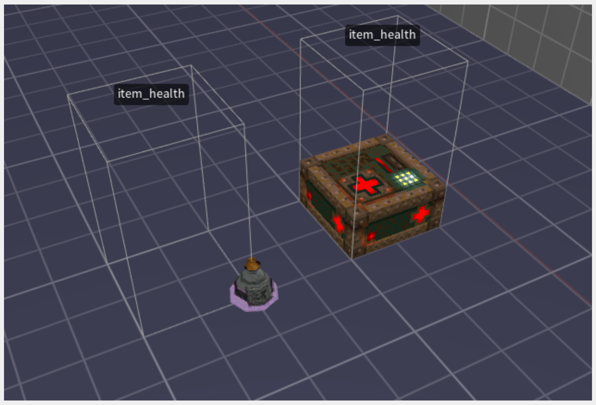
Some custom models may have a different origin than the item you are replacing. If you trigger spawn these items, the teleport "t-fog" particles will be off center. Use the *particles_offset* key to adjust the coordinates. It may take a little trial and error to get the right adjustment.



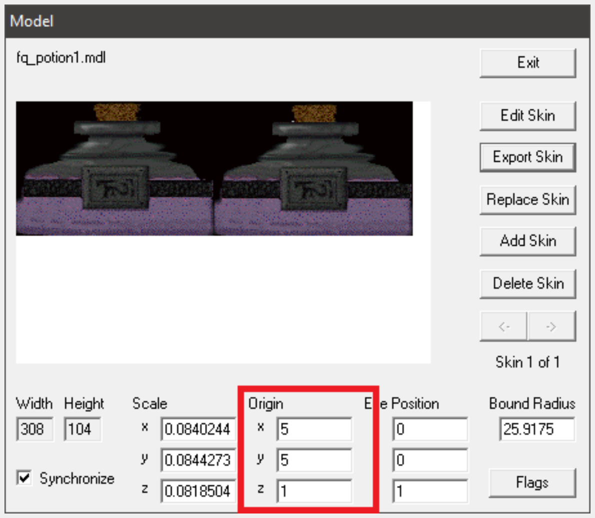
```
origin            448 192 16
particles_offset  -8 0 0
spawnflags        66
```



You may want to edit the model's origin using AdQuedit or a modeling program. Again, it might take a little trial and error depending on the model. Models that rotate should **not** be edited like this unless you remove the rotate flag.



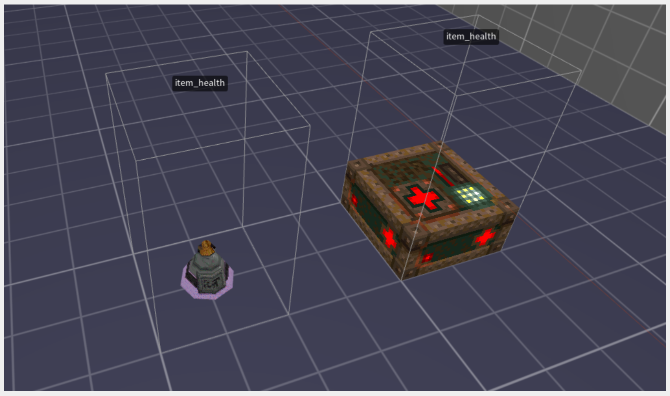
Before


Model

fq_potion1.mdl

Width	Height	Scale		Origin		Eye Position	Bound Radius
308	104	x	0.0840244	x	5	0	25.9175
		y	0.0844273	y	5	0	
☑ Synchronize		z	0.0818504	z	1	1	

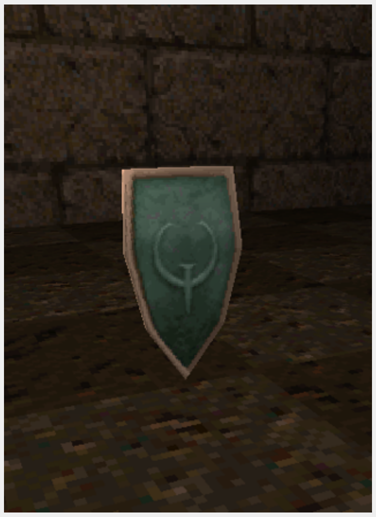
AdQuedit



After

item_armor_shard

Originally *progs_dump* was less ambitious and the end goal was just an "enhanced" vanilla Quake for mappers. But as ladders, breakables and rotating brushes were requested, that strict goal was relaxed. Armor shards have been controversial in the single player community, but can be useful for mappers who want to guide the player in certain directions or reward them with a less potent power-up. These shards originally were part of the canceled *RemakeQuake* project. The code has been modified a bit from that version and *pd_300* has a new original model created by starshipwaters.

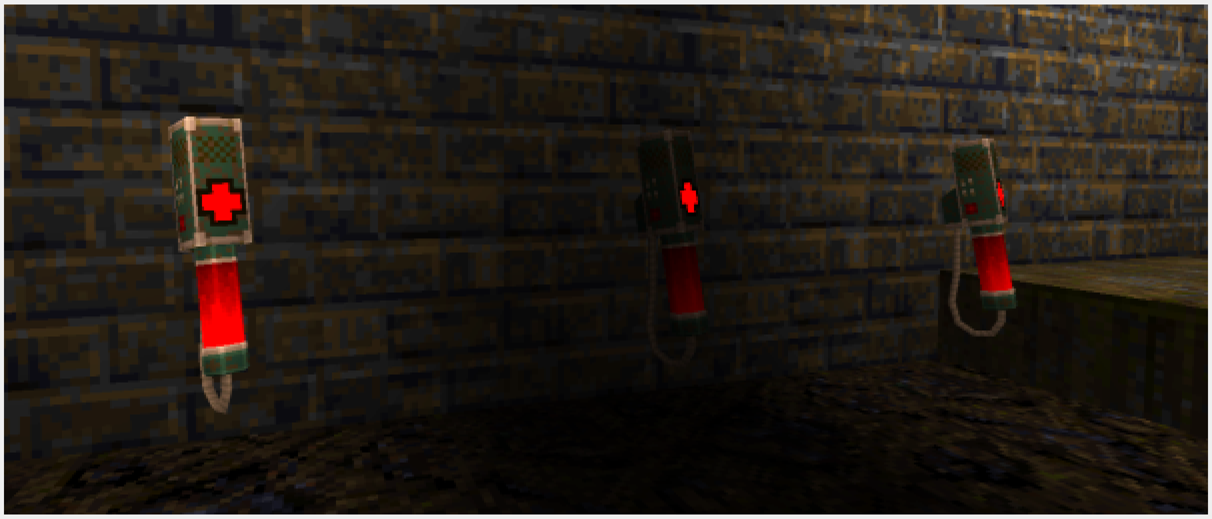


Quake's armor uses a proportional system to protect the player's health points. Each shard is worth 5 points of armor. On their own, shards protect the same proportion of damage as green armor. If you pick up yellow or red armor, the proportion changes to those levels, but the extra points do not carry over. You can buff each armor level up to 25 extra points.

You can set a custom model with the *mdl_body* key, select a *skin*, change its name with *obit_name* and use a custom pickup sound with *snd_misc.*

item_health_vial

Featuring a redesigned base theme for *pd_300*, these are worth 5 health. Vials will now "over heal" the player to a max of 250 health and then "rot" as with the megahealth. As with other items, you can set these to trigger spawn and respawn. You can set a custom model with the *mdl_body* key, select a *skin* and use a custom pickup sound with *snd_misc.* See the FGD for more info. If the new vial design is not to your liking, find out how you can download a pack of alternate models here.



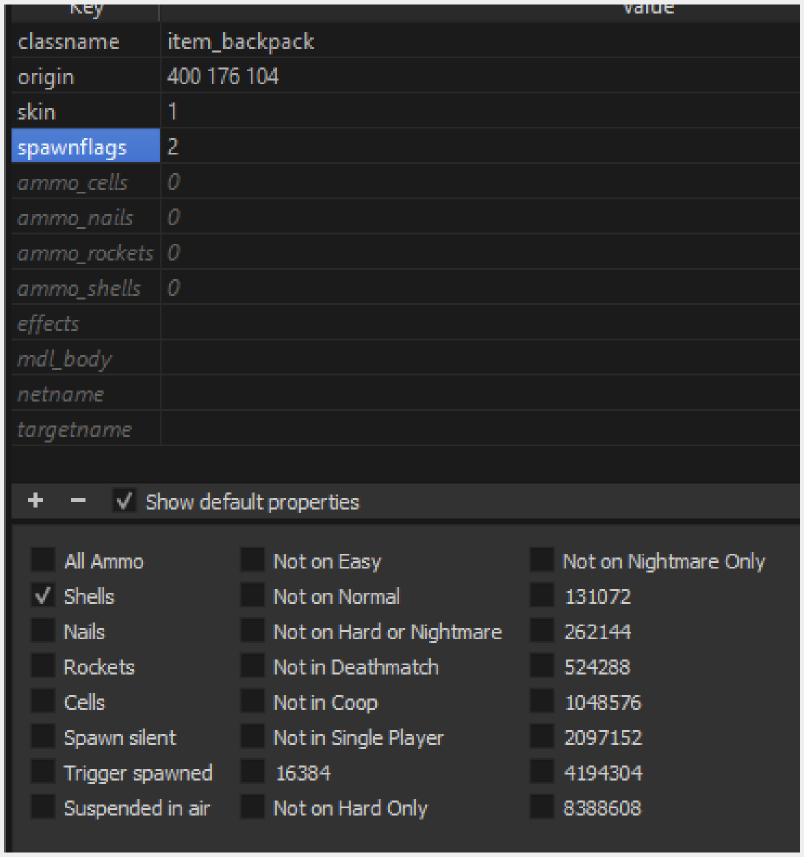
item_backpack

Use this as an alternate ammo pickup for secret areas or for other special gameplay setups. This entity requires you to set an ammo type spawnflag. If the *All Ammo* flag is selected, *item_backpack* gives roughly half the ammo from each of the 4 standard pickups:

10 Shells
12 Nails
2 Rockets
3 Cells

Or you can check off spawnflags to mix and match types. e.g. *Shells* and *Cells*. Override the spawnflag defaults by adding custom amounts to the keys for *ammo_shells, ammo_nails, ammo_rockets* or *ammo_cells*. Make sure and select both the spawnflag and matching *ammo_* type. Remember, player ammo counts will always max out at 100 or 200 depending on the type!

item_backpack



Key	Value
classname	item_backpack
origin	400 176 104
skin	1
spawnflags	2
ammo_cells	*0*
ammo_nails	*0*
ammo_rockets	*0*
ammo_shells	*0*
effects	
mdl_body	
netname	
targetname	

+ − ✓ Show default properties

All Ammo	Not on Easy	Not on Nightmare Only
✓ Shells	Not on Normal	131072
Nails	Not on Hard or Nightmare	262144
Rockets	Not in Deathmatch	524288
Cells	Not in Coop	1048576
Spawn silent	Not in Single Player	2097152
Trigger spawned	16384	4194304
Suspended in air	Not on Hard Only	8388608

***item_backpack* can be trigger spawned, spawn silently and be suspended but cannot be set to respawn.**

Make sure and select one or more ammo_ type spawnflags for this to appear in your map. *All Ammo* will always override any other spawnflags.

item_backpack uses a new model created for *progs_dump* by starshipwaters. However, you can use any other model similar to how monsters work with the *mdl_body* key. You can change the pickup sound with *snd_misc.*

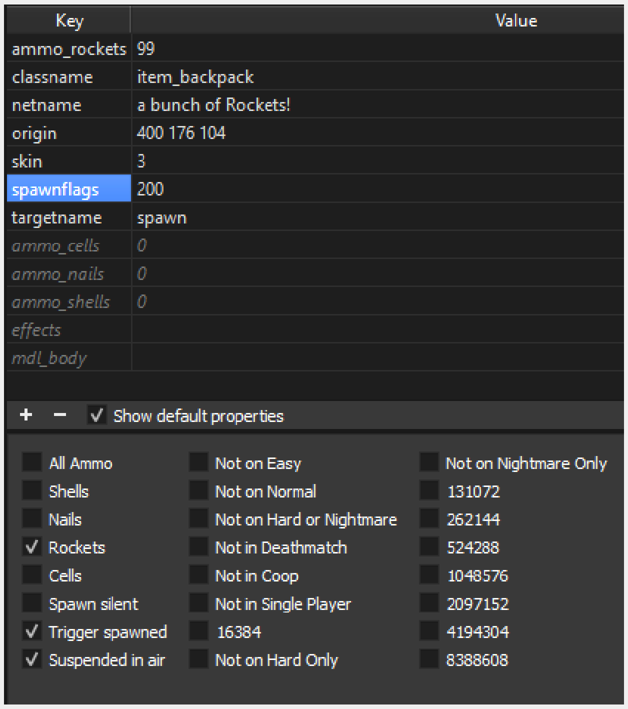
There are 12 different skins you can choose from, including distinct versions for the four standard ammo types. The FGDs feature a dropdown for easy skin selection.

If you prefer the original vanilla backpack, use skin 12 or set the *mdl_body* key to *progs/backpack.mdl.* You can set a *skin* index if you are using a different custom model with built-in skins. To make an invisible backpack set *mdl_body* to *progs/s_null.spr.*

Grunts, Enforcers and Ogres will still drop the standard vanilla backpack when they are killed unless you set *keep_ammo* to 1.

The default pickup message is `You got a backpack.` But you can set a custom message with the *netname* key. 'You got' will be the prefix and the mapper chooses the rest of the message.

e.g. For 'You got a bunch of rockets!' the *netname* value would be 'a bunch of rockets!' In the example below, the backpack will have the rocket skin (3), give 99 rockets only, trigger spawn and be suspended in mid-air. Use the *effects* key to add lighting and particle effects.



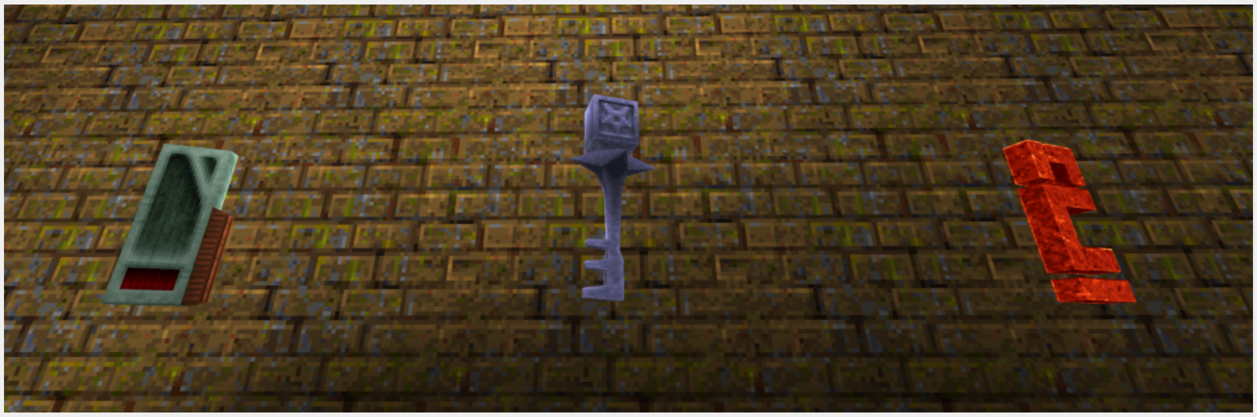
Key	Value
ammo_rockets	99
classname	item_backpack
netname	a bunch of Rockets!
origin	400 176 104
skin	3
spawnflags	200
targetname	spawn
ammo_cells	*0*
ammo_nails	*0*
ammo_shells	*0*
effects	
mdl_body	

☐ All Ammo	☐ Not on Easy	☐ Not on Nightmare Only
☐ Shells	☐ Not on Normal	☐ 131072
☐ Nails	☐ Not on Hard or Nightmare	☐ 262144
✓ Rockets	☐ Not in Deathmatch	☐ 524288
☐ Cells	☐ Not in Coop	☐ 1048576
☐ Spawn silent	☐ Not in Single Player	☐ 2097152
✓ Trigger spawned	☐ 16384	☐ 4194304
✓ Suspended in air	☐ Not on Hard Only	☐ 8388608

item_key_custom

This allows mappers to use any Quake compatible model, sprite or BSP as a key. We've also included new key models with many different color variations.

The new key models are not hard-coded, but you can find them in the *progs* folder and set their paths in the *mdl* key. See Appendix A for more details and screenshots.



Key	Details
keyname	name of the key, e.g. 'bronze key' (required)
mdl	model file path (required)
noise	sound file for the pickup sound (default is per worldtype)
skin	skin index (default 0)
frame	display this single frame of the model, if animated (default 0) NOTE: The key will not display any animation.

Custom keys are not represented on the player's HUD, so keep this in mind if you are going to use them extensively. You will want to communicate to the player what key they require for a specific task using text or other methods.

The *keyname* value is used both for the pickup message and to associate the key with the entity that it unlocks. To make a *func_door* or *trigger_usekey* require this key, set the *keyname* value of that entity so that it matches the *keyname* value of the *item_key_custom* entity.

If different *item_key_custom* entities have the same *keyname* value, they will be treated as different copies of the same key and may be used interchangeably. A map may have a maximum of 23 unique keyname values across all entities.

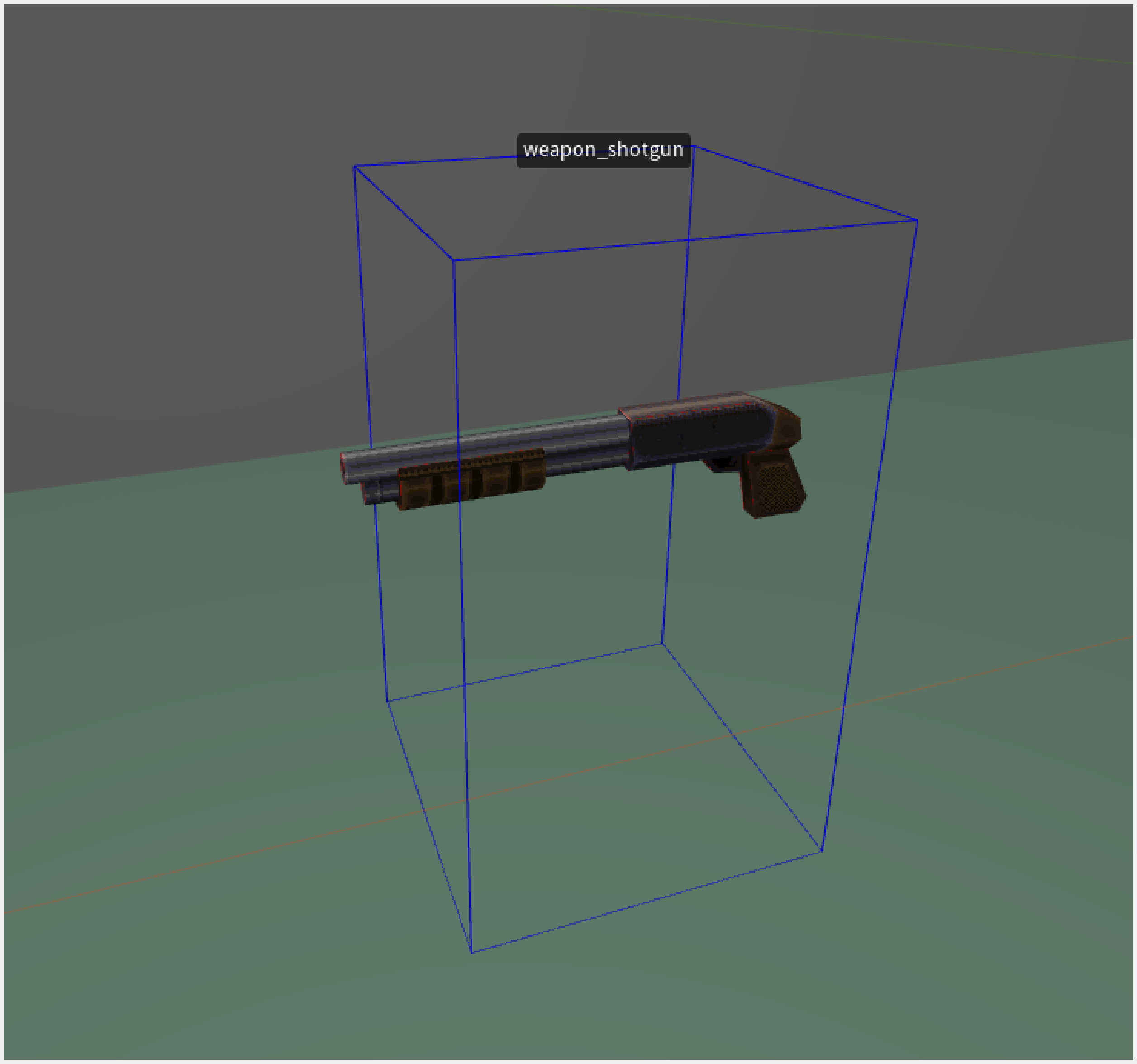
The behavior of an *item_key_custom* should be as the player expects (based on the behavior of the silver and gold keys), except for the fact that it will not appear as an icon in the player's status bar when picked up. This is a limitation of the engine. Finally, there is a sample map called *pd_keys* you can review.

item_key1 & item_key2

In *pd_300*, we've added the ability to use custom models and skins with the standard key entities. Like other entities in *progs_dump*, these are set in the *mdl_body* key and the *skin* key. Unlike *item_key_custom*, you cannot rename, change the pickup message or sounds that are hard coded for these entities. By default, these use the id original models.

weapon_shotgun

This is a pickup model that should be used when you want a player to spawn with only an ax and then later, get the shotgun. e.g. When using *trigger_take_weapon* or *reset_items 2* in worldspawn.



Item Customization Table

Items have different levels of customization options depending on the type:

Category	**Model / Skins**	**Sound**	**Name**	**Details**
ammo	yes	no	no	Set style to 1 in Worldspawn to replace default BSP models with Copper ammo models
armor	yes	pickup sound	yes	Use the *obit_name* key for custom name
armor shard	yes	pickup sound	yes	Use the *obit_name* key for custom name
health	yes	pickup sound	no	Set style to 1 in Worldspawn to replace default BSP models with Copper ammo models
health vial	yes	pickup sound	no	
artifacts	yes	no	no	
weapons	no	no	no	
runes	no	no	no	

If you want to fully customize ammo pickups, use item_backpack instead.

NOTE: If the wrong number is entered in the skin key the model will not display in TrenchBroom.

Custom Sounds

Custom sound files used with these entities must be in the SOUND folder of your mod (or a sub folder under that SOUND folder.) There is no need to add "sound" in the "noise" path. (e.g. *boss2/sight.wav*) Most Quake source ports require a mono sound file for custom sounds. Do not use stereo files in your mod except for music.

Attenuation

A note on the "speed" key (a.k.a attenuation factor) in sound entities. Attenuation in Quake means the reduction of a sound over a distance. Here's a table of what the different speed keys mean in *progs_dump*.

Speed	QuakeC name	Attenuation effect
-1	ATTN_NONE	heard everywhere
1	ATTN_NORM	fades to zero at 1000 units
2	ATTN_IDLE	fades to zero at 512 units
3	ATTN_STATIC	fades to zero at 341 units

play_sound_tiggered

Play a sound when triggered. Most of these key / value pairs can be left to their defaults. Can be looping or a "one off" sound.

Key	Details
volume	how loud (1 default full volume)
noise	path of the sound to play (e.g. *blob/sight1.wav*)
impulse	sound channel 0-7 (0 automatic is default)
speed	attenuation factor (default recommended)

Spawnflag	Details
toggle	sound can be stopped and started when triggered
looping	check this *and* the toggle spawnflag to allow triggered, looping sounds to automatically restart after a save / load

You may encounter problems triggering sounds that are far away from the player. If you do, move the sound and trigger closer to each other.

play_sound

Plays a “one off,” non-looped sound at a random interval. Like thunder or a monster sound. **IMPORTANT**: **Do not use looped sounds with this entity.** For looped sounds see *ambient_general* below. Check out this video tutorial on creating looping sounds for Quake.

Key	Details
volume	how loud (1 is default full volume)
noise	path of the sound to play (e.g. *boss2/sight.wav*)
wait	random time between sounds (default 20)
delay	minimum delay between sounds (default 2)
impulse	sound channel 0-7 (0 automatic is default)
speed	attenuation factor

ambient_general

Plays a custom looped sound. Cannot be toggled off or triggered. *noise* = path of the sound to play (e.g. *ambience/suck1.wav*)

ambient_thunder

Originally unused in the game. Plays the sound of thunder at a random interval. You only need one of these in your map. It will play everywhere. If you want it to play locally instead, use a play_sound with a different speed setting. The path for the sound is: ambience/thunder1.wav

ambient_water1

Swirling water sound effect. Usually this is added automatically to maps with water when you run VIS. If you want to place these in your map by hand, you can run VIS with the -noambient command line switch.

ambient_wind2

Howling wind sound effect. Usually this is added automatically to outdoor sections of maps with sky textures. If you want to place these in your map by hand, you can run VIS with the -noambient command line switch.

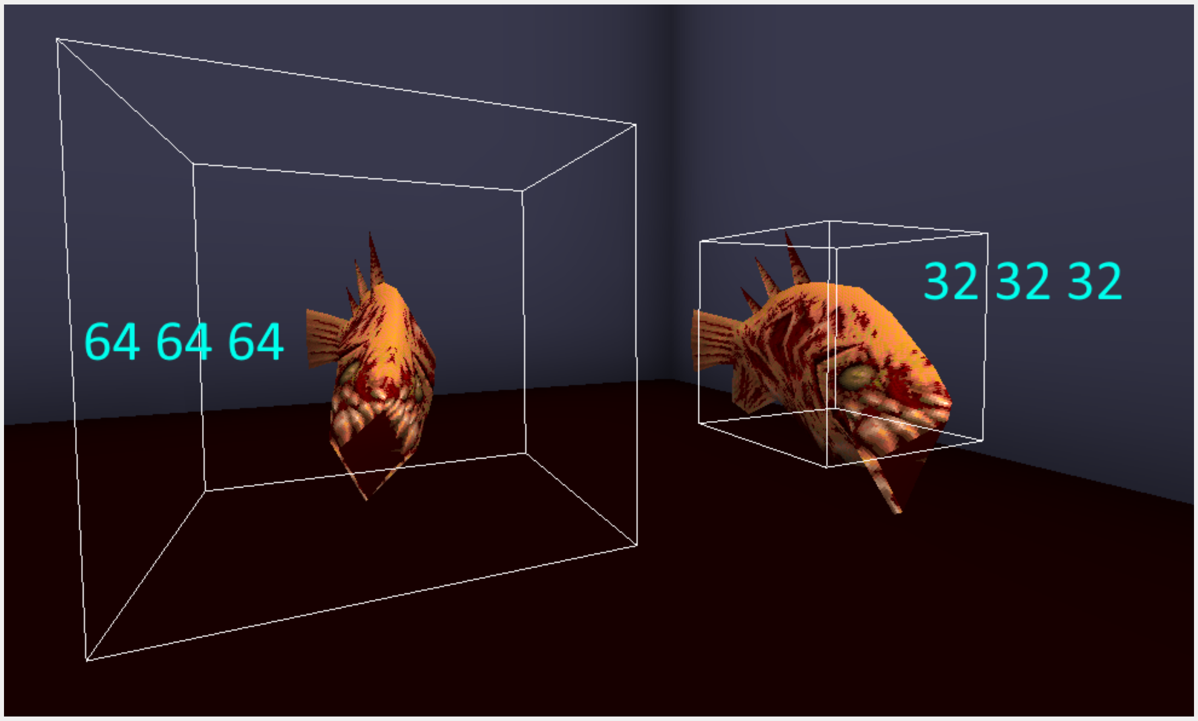
ambient_fire

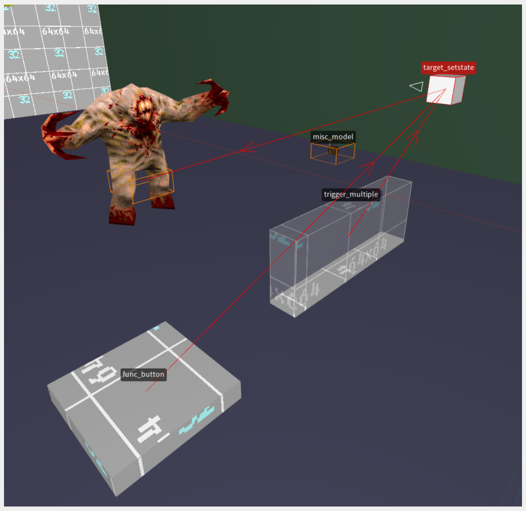
This is a simple looping sound from the torches. Use this if you are using custom fire sprites or models. This is the same effect as FireAmbient in earlier versions of *progs_dump*.

Custom Models and Sprites

misc_model

A point entity for displaying models and sprites. A frame range can be given to animate the model. Starting with version 3.0.0, *misc_model* has many more features including the ability to trigger animations on and off via a *target_setstate* entity. There are now spawnflags that add gravity, solidity, and new animation options. Additionally, models can be set to *Start Hidden* and can be toggled off and on. You can also set a custom bounding box size. See below for details.





Key	**Details**
mdl	The model to display. Can be of type mdl, bsp, or spr.
frame	Single frame to display. Can also be used to offset the animation starting frame for variations (e.g. when using fire sprites)
first_frame	The starting frame of the animation.
last_frame	The last frame of the animation.
speed	The frames per second of the model's animation. Divide 1 by the fps for this value. (Default 0.1)
angles	pitch roll yaw (up/down, angle, tilt left/right) **IMPORTANT**: In TrenchBroom set the *angle* (no S) value to 0 if using *angles* (with an S) key to rotate mdls (see gib _section for more info or the video linked below.)
mdlsz	Custom bounding box in x y z (default is 32 32 32)
centeroffset	Model center offset (similar to paticles_offset in Custom Item Models)
spawnflags	**Gravity** - model can ride lifts or trains **Solid** - model has collision box, use mdlsz to set the area **Back and Forth** - will animate back and forth between first_frame and last_frame **Play count times** - will animate a set number of times, use count key **Start Hidden -** The model will not be visible until triggered. You can toggle this on and off with additional triggers.

Here's a link to an older progs_dump video that has some more info and tips for using *misc_models* in your mod.

Enhanced Triggers

is_waiting

If this value is set to 1, triggers will do nothing until another trigger activates it. The FGD provides a dropdown selection or you can enter the value by hand. See the Entity State System section for more details.

***In order to use *is_waiting* on a trigger_teleport, make sure and use *targetname2* to "wake it up" instead of *targetname*.**

trigger_changetarget

Use this to change the target field on an entity to a different value.

Key	Details
target	The targetname of entity you want to change
message	The new targetname
cnt	Which target field should be changed? target (default) target2 target3 target4

trigger_changelevel

We’ve made some changes to this entity in version 3.0.0 that allows mappers to display the “Congratulations” graphics and a custom message. These are displayed in the original game at the end of an episode when the player gets a rune or defeats a boss. This feature can also display multiple “pages” of text before the player warps to the next level using added *info_intermissiontext* entities. If no *message* is set in the *trigger_changlevel*, the entity will work as usual.

The setup for the custom ending screen is pretty simple. Set a string of text in the *message* key of a *trigger_changelevel* entity. There is a limit of 40 characters on each line, so use “\n” to continue the text on the next line if needed. If you want another page of text, use the *message* key in an *info_intermissiontext* entity. Set the *cnt* to 2, which tells the game to display this as the second “page” of text. If you want more pages, add additional *info_intermissiontext* entities and set the *cnt* to 3 for the third page and so on.

On *trigger_changelevel* entities that point to a hub or start map, the *Use info_player_start2* spawnflag will spawn the player on the *info_player_start2* entity when the map changes. Use this to skip skill selection when completing an episode as in the original game. Or you can return the player to a different part of a hub map.

Ensure there is an *info_player_start2* on the map you are changing to when the *info_player_start2* spawnflag is used.

Key	**Details**
map	Next map
message	Message to show on intermission screen

Spawnflags	**Details**
1	No Intermission
2	Ignore info_intermissiontext Use this when you have multiple *trigger_changelevels* in your map to warp the player from the level with no additional messages
8	Start off
16	Use info_player_start2

trigger_heal

When a player enters this trigger they are healed at a rate of 5 HP per second (by default.)

Key	**Details**
heal_amount	Healing per second (default is 5 HP)
health_max	The upper limit for healing (default 100, max 250)
spawnflags	**Start on -** Start on if using targetname. Only needed if triggered by something other than touching. **Player only, Monsters only**
noise	path to custom healing sound

trigger_look

John Romero wanted this in the original game and thanks to NullPointPaladin, it's finally here! This will trigger when a player is within the brush trigger and looks *directly* at a target entity. Use the first *target* key for the "looked at entity" and use the subsequent targets (2-4) to trigger other events. See sample map *pd_cutscenes* for one setup using a skip textured *func_wall* (more on this below.).

The entity targeted by *trigger_look* needs to be an entity with a bounding box (bbox), collision (solid), be targetable and visible (not moved out of bounds). A *func_wall* is a great example although entities like gibs, lasers, dead monsters, or others can be targeted as long as they are "solid" by using the *solid* spawnflag where applicable.

A *func_illusionary* doesn't work because it doesn't have a bbox that interacts with the player. A *func_detail* is ignored, as it cannot be targeted.

If you need an invisible brush, use a skip textured func_wall. If you need to avoid triggering a second time use a *trigger_relay* to killtarget the *trigger_look's* target or the *trigger_look* itself. Also, the default distance from the player to the look target is 500 Quake units. If the target needs to be farther away or closer, set the *speed* key to the proper distance. Use a brush as a "ruler" to measure the distance in TrenchBroom if needed. You can set a custom sound by setting sounds to 4 and adding a path to the sound in the *noise1* value.

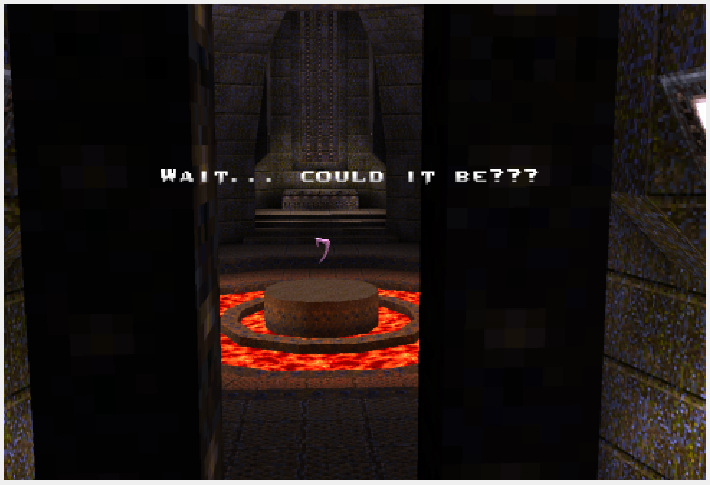
This will **not** work or is not recommended with the following brushes: *func_illusionary, any func detail variant, func_group, func_particle_field, non-solid func_laser, func_bossgate* or *func_togglewall*.

Brushes textured in *clip* will not work.

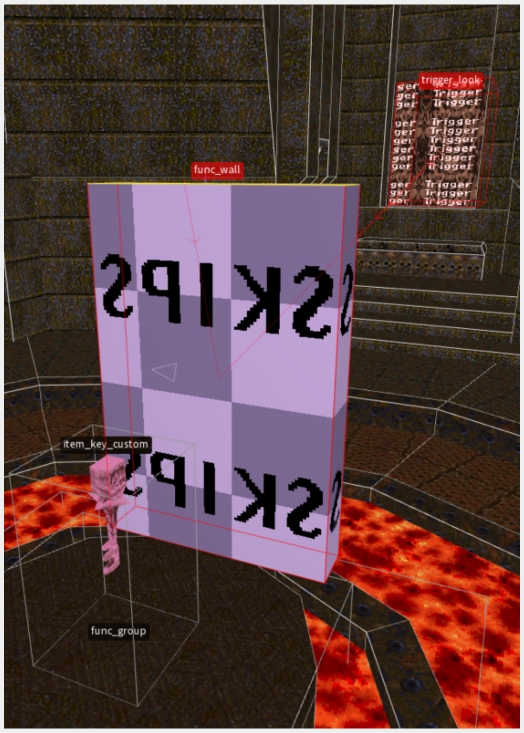
Don't use items or monsters as the trigger as they can be removed and /or lose their bbox upon death. .

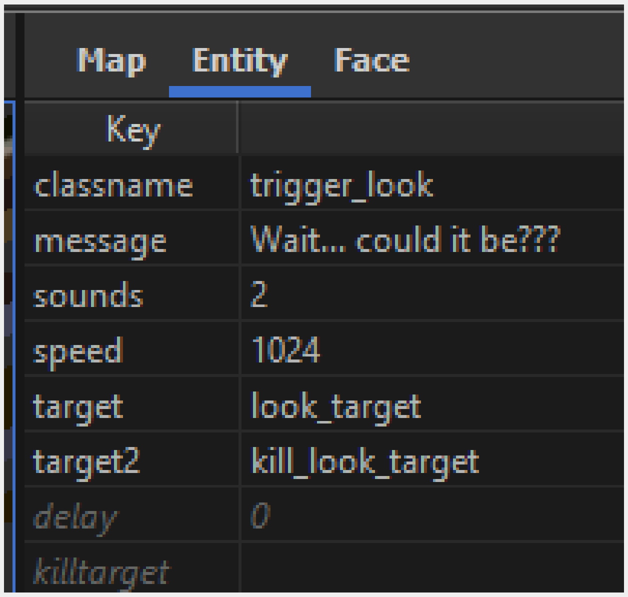
Key	Details
speed	Distance from player to search for trigger, adjust if the target is too far from the trigger (default 500 units)
wait	Time between re-triggering (default 0)
sounds	0-3 are standard Quake trigger choices, 4 allows a custom sound, requires a path set in noise1 key
noise1	Path to custom sound. Use with sounds key set to 4 (e.g. fish/bite.wav)

You can see *trigger_look* in action near the beginning of the sample map *pd_cutscenes*. When the player looks at the rune a message plays. Yes, you could do this with a *trigger_once* using an *angle* key, but using this method it will *only* trigger if the player looks directly at the platform.



In this example, we used a *func_wall* textured in skip. The reason this is used instead of the rune is that the rune's bounding box is smaller *and* farther away, making it harder to trigger correctly during testing.





Key	
classname	trigger_look
message	Wait... could it be???
sounds	2
speed	1024
target	look_target
target2	kill_look_target
delay	0
killtarget	

We used a larger brush in this case and killtargeted it before the player could collide with it. If the setup is in a smaller area you will likely be able to use the entity itself. Test early! Test often! One tip is to have a backup method of targeting entities in case the player doesn't look at what you need them to trigger! In this map, we used the button nearby to killtarget the func_wall just in case the player missed this moment in the "story."

trigger_multiple

We've added a *Wait for retrigger* spawnflag to the standard *trigger_mutiple*. This allows you to toggle these in more simple setups without having to use *target_setstate* entities.

trigger_push_custom

This can be used to create traps, jump pads, water currents and more.

If the *Start Off* spawnflag is set the entity will not trigger until targeted. This can be targeted and toggled off and on. If the *Silent* spawnflag is set it won't make the standard "windfly" sound. Use *Custom Noise* spawnflag and the noise key/value together to use a custom push sound. Custom sounds should be "one off" sounds NOT looping sounds. A good way to simulate a water current is to have the trigger_push_custom under the surface of your water brush by about 32 units. You can see an example in the pd_gallery.map

trigger_monster_jump

If the *Start Off* spawnflag is set the entity will not trigger until targeted. This can be targeted and toggled off and on. So monsters can be attacking from a distance and then be triggered to jump.

The way *trigger_monsterjump* works requires a monster to be "awake" and "angry" at the player before the jump is activated. You can always target a monster with a *trigger_once* to wake them up.

trigger_take_weapon

This will remove the shotgun from the player's inventory and all shells. Place this over an *info_player_start* to have the player start with only the ax… or use this trigger to surprise the player in some devious way. Make sure and place a weapon_shotgun in your map for the player to get eventually!

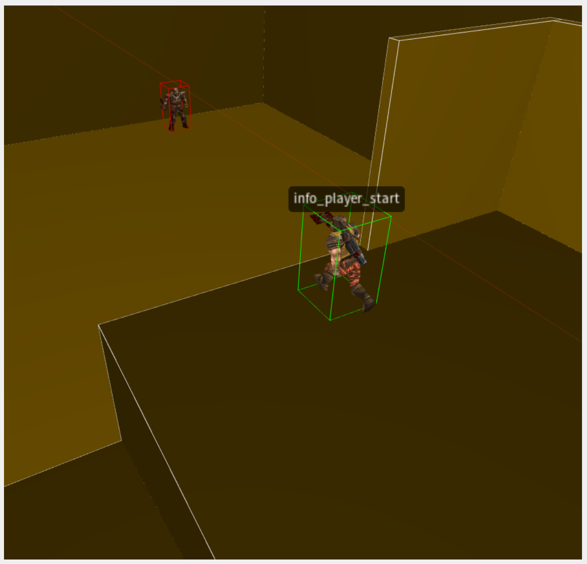
An alternative would be setting *reset_items* to 2 in your worldspawn if you want an "ax only" start.

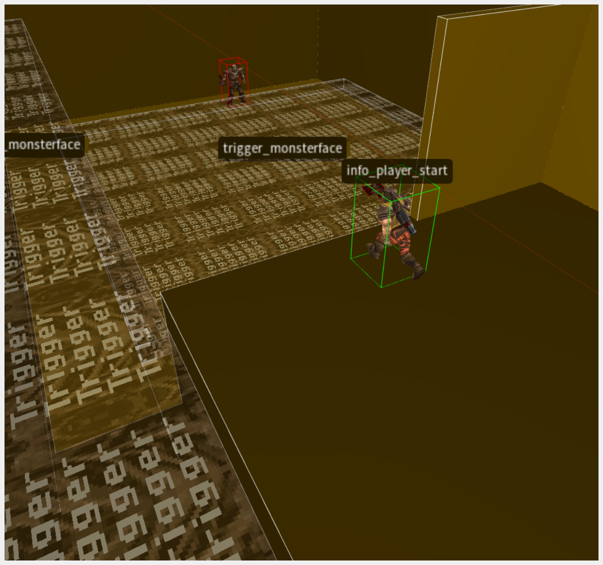
trigger_monsterface

Running monsters that cannot see their target and touch this, will face in the direction of the trigger's angle instead of the unseen target. Use this trigger to make monsters leave their sniping spot and execute complex maneuvers.

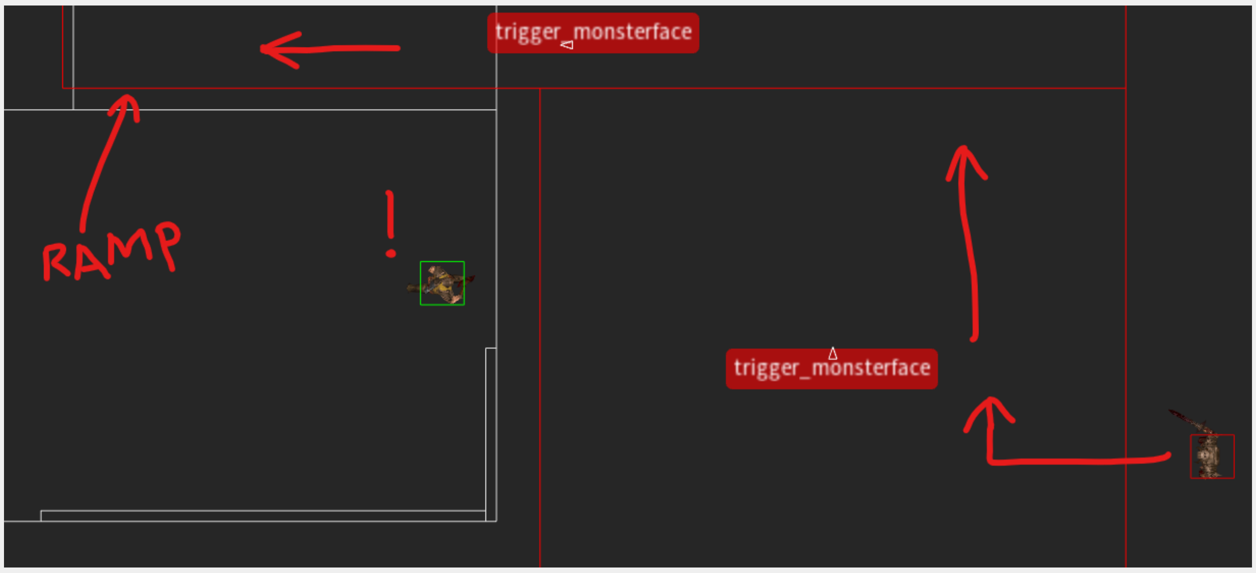
Key	**Details**
angle	Angle the monster will face (and run) while in the trigger volume.
wait	Wait time between re-triggers (default is 0)

This is a new trigger from TheSolipsist that can help you guide monsters who might otherwise get stuck on geometry or wander around trying to find the player. This takes a bit of testing to get right and it was designed for specific situations. For example: The player is above a Hellknight who can run up a ramp to reach them. However, if the player is hidden, the Hellknight will run around almost randomly and it's not guaranteed that he will ever find the ramp.





But you can guide the monster once they lose sight of the player with a series of two or more *trigger_monsterface* brushes. As you can see below, the triggers point the monster in the correct direction using the *angle* key. This won't upgrade the monster AI, but it's a powerful new tool for monster setups if used wisely.



Remember: *trigger_monsterface* only works when an already "angry" monster <u>is inside the trigger volume and cannot see the player.</u> Once they see their target, their behavior will go back to the default (attacking or running) until the player is hidden again.

trigger_setgravity

If the *Start Off* spawnflag is set the entity will not trigger until targeted. This trigger changes the gravity on a player or monster that touches it. The trigger itself can be toggled on and off.

The amount of gravity can only be changed by touching *another* trigger_setgravity with a different setting.

The *gravity* key defaults to 0 which is normal gravity. Lower numbers (e.g. 25) equal lower gravity. Setting 100 is normal gravity. Numbers above 100 will make the player "heavier", i.e. harder to jump.

If you want multiple *trigger setgravity* triggers in one room or area, make sure the brushes are not touching each other. This can cause the triggers not to work properly.

trigger_shake

Earthquake trigger - shakes players in its radius when active. Strength of the tremor is greatest at the center.

dmg is strength at center (default is 120.) *wait* duration of shake (default is 1.) *count* effect radius (default is 200.) *noise* path of sound to play when starting to shake. *noise1* path of sound to play when stopping. We've included one earthquake sound effect, see Appendix A for details. *targetname* must be triggered. The *VIEWONLY* spawnflag shakes the view, but player movement is not affected. Check out the *pd_bosses* and *pd_lava* sample levels to see this in action.

trigger_usekey

Variable sized single use trigger that requires a key to trigger targets. Must be targeted at one or more entities. Use the *message* key to create a custom message for this. e.g. "Bring the Gold Key here mortal!" This trigger can be set to dormant with *is_waiting* 1. Setting *cnt* to 1 will not remove the key from the player's inventory, which mimic's the key behavior of Doom. Make sure and add this key | value to all doors and / or let the player know the default key behavior has changed. e.g. Perhaps a pickup message on the keys that reads: "This key works on many doors."

trigger_void

Use this for a 'void' area. Removes monsters, ammo, etc... also kills players. Spawnflags can be used to protect players or monsters.

Music Triggers

These allow you to change or start a music track. One is brushed based and the other is a point entity that needs to be activated by a trigger. They basically work the same with some differences noted below. The reason there are two methods is to ensure backward compatibility with an older *progs_dump* based mod.

Unfortunately, you cannot fade music up or down and the music track will begin immediately.

We recommend using track numbers 12 and up for your custom music since track02 through track11 are the original track names. Some players like to play maps with the original soundtrack.

trigger_cdtrack

A point entity. The number of the track to play goes in the *count* key. e.g. 32 for track32.ogg See *trigger_changemusic* below for more information on formats etc. **NOTE**: the track number uses the *count* key here but *trigger_changemusic* uses the *sound* key for the same info. Also, this entity does not use the entity state system while *trigger_changemusic* can.

trigger_changemusic

A trigger brush that changes the currently playing music track. The number of the track to play goes in the *sounds* key (just like *worldspawn*). Most Quake engines require the XX for trackXX.xxx format. e.g. 02 for track02.ogg or 98 for track98.mp3

Different Quake engines play different formats. This is why Quakespasm, Ironwail and vkQuake are recommended, as they can play mp3, ogg, wav and FLAC formats. You can read more about formats here and much more detailed information about how to format music for Quake here.

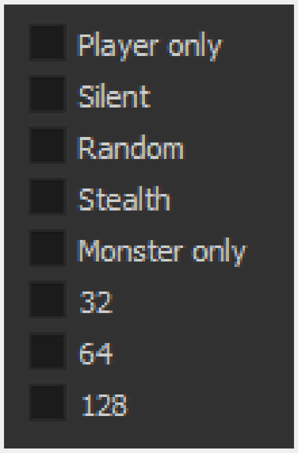
music/track99.ogg

We've added this silent file to *progs_dump* so mappers can "stop" a music track altogether if desired. Simply use one of the two methods above and change the track to 99 in the appropriate key for track99.ogg You can rename this to whatever you'd like as long as it follows the naming scheme described above.

trigger_teleport

info_destination_random

progs_dump adds a number of new features to *trigger_teleport*. These are selected with new spawnflags. In the case of the *random* spawnflag, there is a new entity *info_teleport_random* which is a random destination marker for the trigger. *info_teleport_changedest* in another entity that can be used to tell a *trigger_teleport* to change its target value to a different destination. More info below.



Spawnflag	**Details**
Player Only	as in original Quake
Silent	no ambient sounds as in original Quake, use for monster closets and secrets etc.
Random	can teleport to random destination entities
Stealth	No particles or sound effects
Monster Only	can be used with other spawnflags: stealth, silent and random in any combination

Using the *random* spawnflag on *trigger_teleport* requires use of the *count* key and targeting a *info_teleport_random* (instead of the regular *info_teleport_destination*). This causes the teleporter to send the player to a random destination among the *info_teleport_random* markers in the level. In the *count* key, add a number equal to the number of *info_teleport_random* entities you placed.

info_teleport_changedest

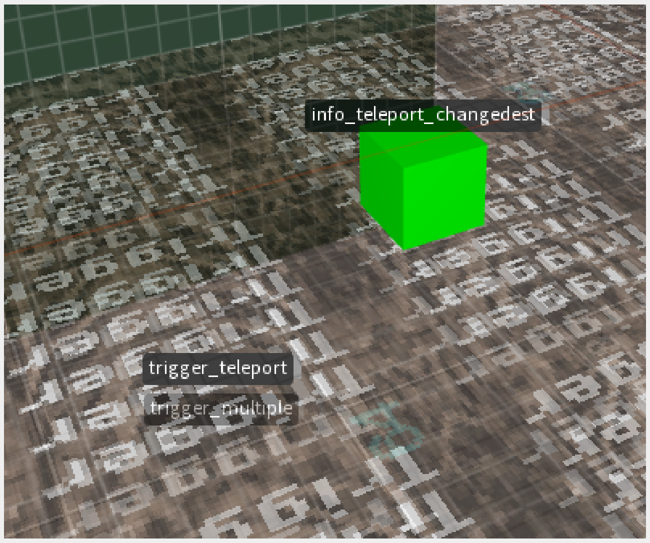
Allows a mapper to change the target of a teleport_trigger. Useful in maps where the player may fall into a void and the mapper wants to update where they "respawn" as they progress through the level. Could also be used for teleport puzzles and more. This requires the addition of a trigger_multiple in some setups.

Key	**Details**
target	the targetname of the trigger_teleport to change
message	new info_teleport_destination's targetname to switch to
targetname	name of this entity so we can trigger it with a trigger_mulitple or other

NOTE: The best way to understand how this works is to look at the sample map *pd_change_dest*.

The basic workflow is to set up a *trigger_teleport* with a *target* as usual, then add a *targetname*. However, instead of using an *info_teleport destination*, the *targetname* will be referenced by an *info_teleport_changedest* point entity. The *info_teleport_changedest's target* key should match the targetname of the *trigger_teleport*. The *message* key is the "replacement" targetname for the trigger_teleport. Finally, the *targetname* is used to trigger this entity. This pattern can be duplicated multiple times to change the destination of a single *trigger_teleport*.

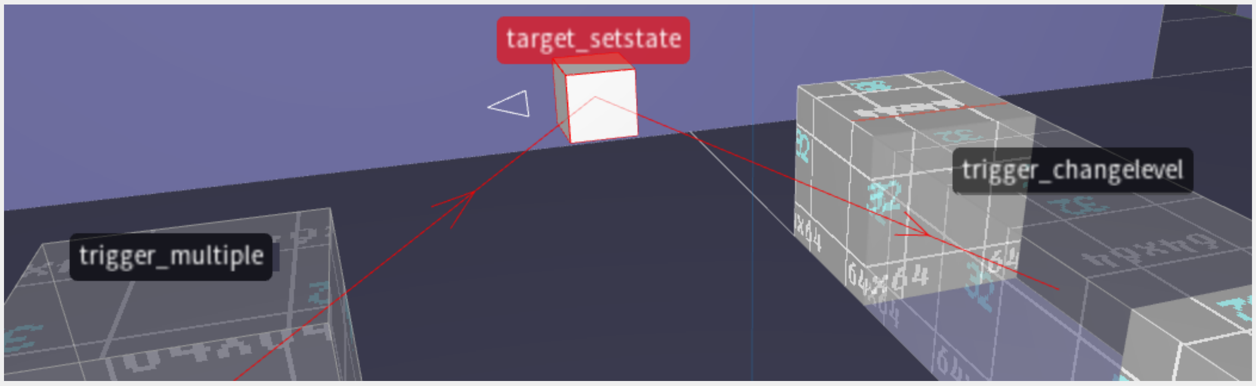
NOTE: when a *trigger_teleport* has a targetname it must be triggered to operate, so adding an overlapping *trigger_multiple* targeting the *trigger_teleport* will be necessary. Place the *trigger_multiple* 8 units inside the *trigger_teleport* and this will re-trigger it. If you need to "kill" the *trigger_teleport*, killtarget the *trigger_mutiple* and the teleporter will no longer work.



Entity State System

The text below was adapted from the Alkaline Devkit Mapping Guide by bmFbr.

Starting in version 3.0.0 *progs_dump* has a state system somewhat similar to AD's "estate" system and Copper's target_lock entity. With it you can disable/enable entities at will, giving you greater control over complex map scripting and events. Entity behavior when disabled is class-dependent. Take a look at the *prefab_estate* sample map for examples.



target_setstate

You set an entity's state using the *target_setstate* point entity. You can directly disable, enable, or toggle the state. Target entities can start the map already disabled through the spawnflag *Targets start disabled*.

If you need to target an entity that has its behavior changed when given a targetname (like *func_door* or *trigger_teleport*), you can make the *target_setstate* target the entity's *targetname2* field.

This system is integrated with dormant triggers (*is_waiting* key), meaning that you can "wake" a trigger not only by triggering it, but also by using a *target_setstate* on it.

The entity's state is kept under the *estate* field, with 0 meaning enabled/normal operation, and 1 disabled. So for triggers, you're able to make them start off by directly setting this value to 1. You <u>cannot</u> do that to doors nor buttons, in those cases you need to use a *target_setstate* with "start disabled" set. In some cases, the FGD will not display the *estate* field, but you can add this by hand if needed. A lot of entities support enabling/disabling:

Most trigger brushes

When disabled, most *trigger_** brush entities get their touch action turned off. This includes self-sufficient entities that don't need to target anything, like fog triggers, ladders, monsterjumps, changelevels, etc.

Also, any entity that fires their targets won't do it while disabled (like *trigger_relay*), and a *trigger_counter* won't get its use count increased.

Doors

Like in Copper, disabling a door makes it not respond to touches or triggers, and disables its trigger field. Doors will also close immediately when disabled. Toggling doors or with *wait -1* set won't close by default, but you can change that behavior using the spawnflag *Force all doors to close*. In this case, if re-enabled, the door will return to the position it was before disabling.

Buttons

Also similar to Copper, buttons depress themselves when disabled and cannot be activated. If the button has a *wait -1* value (that is, it's a press-once button), when re-enabled it'll reset back to its un-pressed state regardless if it had been activated before. You can change that using the spawnflag *Don't reset button state*, in which case it'll revert to its previous state when re-enabled.

Key	**Details**
style	0 : Toggle current state (default) 1 : Force active 2 : Force inactive
Spawnflag	**Details**
Targets start disabled	force the target entities to spawn disabled/locked
Force all doors to close	Forces doors with *wait -1* to close
Don't reset button state	Revert button to previous state when re-enabled if *wait -1* is set

Advanced Triggering

The text below was adapted from the Alkaline Devkit Mapping Guide by bmFbr.

progs_dump version 3 has advanced scripting through two specialized entities:

trigger_filter

This point entity relays only if a given condition evaluates to true, unleashing ultimate scripting capabilities. You can test various fields on the entity targeted by the *include* field (you can target another entity's *targetname2* as well). If you leave the *include* field empty, it'll test against the activating entity instead, be it a player, a monster or anything else - see trigger_everything below for more details.

You can, for example, relay a trigger only if a door or plat is in a certain position, or if a targeted enemy is alive or dead or below a certain health amount, or if the player is carrying some specific weapon or item, or if the activator is a certain type of weapon projectile - the possibilities are almost endless.

In case you're filtering to activate only in case if it's a specific weapon projectile (like a grenade for example), you can make the targets fire with the activating entity's owner as the activator with the spawnflag *Relay activator as owner* - say, if the trigger is activated by a player's grenade, it'll fire with the player as activator, instead of the grenade itself.

There are various fields available to test, selectable through the style key. They all fall into three field types (float, flag and string), and for each there's a set of operations available to evaluate, selectable through the weapon key.

Float types can do all operations, and are compared to this entity's count field. Flag types support Equal and bitwise AND, and are compared to the aflag field. String types have only Equal evaluation, and are compared to this entity's type field. You can set the trigger to fire if the condition evaluates to false by activating the spawnflag Trigger if false. It'll never fire if the entity targeted by include isn't found, regardless of this spawnflag.

If you need to evaluate more than one condition, you can chain multiple filters in a row. Also, the state of the last evaluation is stored in this entity's state field (1 for fired, 0 for not fired).

trigger_everything

A no-holds-barred touchable solid trigger. Everything triggers it. Players, monsters, projectiles, gibs, even other solid entities like doors and plats.

It can only fire targets matching its *target* field. Due to its "fire-all" nature, it doesn't have support for the '*target2-4*' fields, neither *message* nor *killtarget*. Useful when used in conjunction with the *trigger_filter* point entity to filter out its firing.

By default it'll fire for each and every registered touch, even if simultaneous. If given a *wait* value, it'll act like a *trigger_multiple*, where a single activation puts it on hold for *wait* seconds. If the *target* is a *trigger_filter*, you can set it to wait only if the filter evaluates to true through the spawnflag *Wait only if trigger_filter evaluates to true*.

Enhanced Platforms

func_new_plat

This entity adds new capabilities to plats. It uses spawnflags to dramatically change its behavior. As with the standard plat, build your plat in the raised position so the entity will be lit correctly when you compile your map.

You must use one of the following spawnflags with *func_new_plat*. Even though they use the same entity name, each spawnflag creates a very different plat.

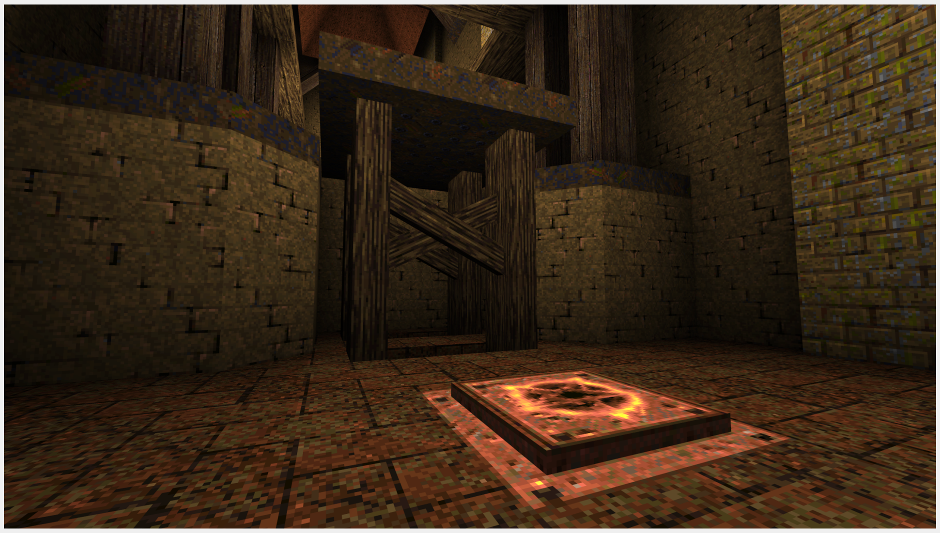
Spawnflag 1: Setting the *Plat Start at Top* spawnflag creates a plat that starts at the top and when triggered, goes down, waits, then comes back up. *health* = number of seconds to wait (default 5)

Spawnflag 2: Setting *Toggle Plat* creates a plat that will change between the top and bottom each time it is triggered.

You must use the *height* key when *Toggle Plat* is used. Use a negative height number to start the plat off in a lower position.

Spawnflag 16: *Plat2* creates a plat in the bottom position, just like the standard plat. If a plat2 is the target of a trigger, it will be disabled in the lowered position until it has been triggered. *Delay* is the time before the plat returns to its original position.

You can set the *height* to tweak the amount of lip needed. See *The Gallery* map for an example.



Elevators

func_elvtr_button

This entity turns a *func_new_plat* into a multi-floor elevator. Here are the steps to follow to create one. You can see this setup in the *pd_elevator* demo map:

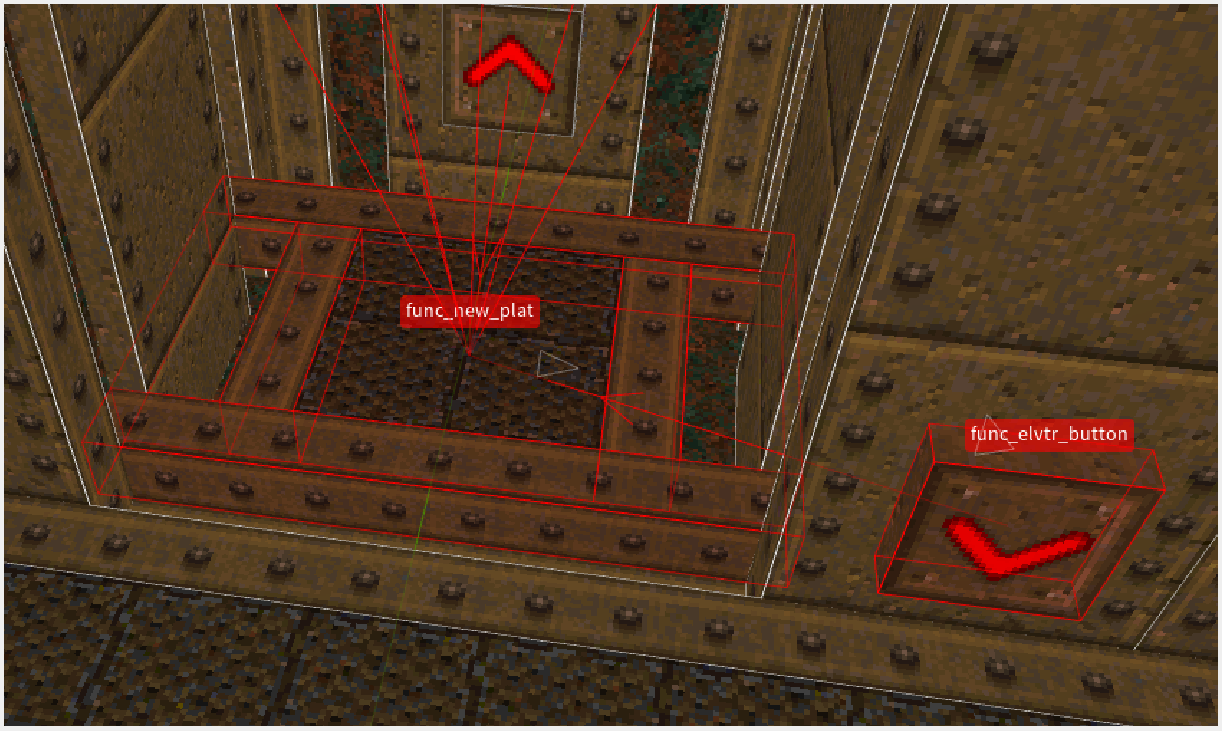
First, create a *func_new_plat*. Select the *Elevator* spawnflag (4). Set the *cnt* key to the number of floors (3 in the demo map). Next, set the *height* key to the vertical distance between floors (256 in the demo map). Then, give the func_new_plat a targetname.

By default, the elevator starts at the bottom floor, so that's where the func_new_plat needs to be positioned in the editor. Alternatively, if the mapper wants it to start at the top floor, they can manually position the bmodel at the top floor and set spawnflag (8) Elevator Start at Top.

With the func_new_plat done, create any number of func_elvtr_button entities. Make each func_elvtr_button target the func_new_plat. A func_elvtr_button is an "up" button by default. To make it a "down" button, use the spawnflag Down Button.

When the spawnflags are set to elevator the wait key on a func_new_plat is defaulted to zero. This means the player will be able to hit another button right away between floors as seen in the demo map. The wait key on a func_elvtr_button behaves just as a regular func_button would, controlling how long before you can hit a button each subsequent time.

NOTE: any *func_elvtr_button* will act as a "call" button if the elevator isn't already at that floor.



Fog System

Starting in version 3.0.0, *progs_dump* has a new fog system that allows mappers to change fog values over time or distance. One example of this would be an outdoor area with one fog color and density and when the player moves inside an indoor area these values change. These new entities are based on The Copper mod's implementation with changes that were incorporated into the Alkaline mod by bmFbr. They also work similarly to the fog triggers in Arcane Dimensions.

Most modern Quake engines feature a fog system and *most* of these are identical since they all stem from the same source code. Usually, a mapper can simply set a global fog density and color in the *worldspawn* entity. It's important to note that these triggers don't use these values. Lunaran explains all this in detail in the Copper manual:

An important thing to remember: the fog key / value that goes on Worldspawn doesn't interact with this system. It is a feature of various engines (such as those in the Fitzquake family) and isn't interpreted by game code. The reason for this is that it's interpreted as four values (one for density and three for color) and set by the engine at load time. The numeric fields that are passed on to the game code (ie Copper), however, can only be one value (a float) or three (a vector). If you set fog in your world with a fog parameter on worldspawn, and never bother with fog_color and fog_density on any entities, you'll get the usual static global fog the way it's always worked. Color and density on a player start will be evaluated on the first frame of gameplay, and will override any fog set by the engine.

If you are planning to use the fog system in *progs_dump*, use the key | values in the various controller entities instead of setting this in the *worldspawn.*

These fog entities can cause slowdowns and other issues if your triggers are too large. Also, a bug in most quake engine ports will reset the eye position smoothing that happens when climbing stairs or riding a plat on every frame that a 'stuffcmd' is sent, so fog transitions during upwards motion will cause noticeable stuttering.

If you want to learn more about how fog is used in Quake check out my Skyboxes and Fog tutorial on Youtube. There's also some good information on the VDU website.

trigger_fogblend

Acts as a smoothly blending portal between two zones of different fog. Sets the fog for any client passing through it, blending their global fog settings between *fog_color* and *fog_density* and *fog_color2* and *fog_density2* proportional to their position within the trigger.

Key	Details
fog_density	Start Fog Density (e.g. 0.3)
fog_color	Start Fog Color (e.g. 0.25 0.25 0.25)
fog_density2	End Fog Density
fog_color2	End Fog Color
angle	The axis of motion on which the blend happens is defined by *angle*, pointing to whatever zone has color2 and density2. Trigger therefore has two 'sides': the side that *angle* points to, and the opposite side.
distance	Override the length of the blend period in world units - defaults to bounds size on 'angle' otherwise. This is only useful for diagonal triggers.

target_fogblend

Activator's fog will be blended over time from start to end values.

Key	Details
fog_density	Start Fog Density (e.g. 0.3)
fog_color	Start Fog Color (e.g. 0.25 0.25 0.25)
fog_density2	End Fog Density
fog_color2	End Fog Color
angle	The axis of motion on which the blend happens is defined by *angle*, pointing to whatever zone has color2 and density2. Trigger therefore has two 'sides': the side that *angle* points to, and the opposite side.
delay	Pause before starting blend
speed	Time to blend to fog2 (-1 for instant)
speed2	Time to blend back, if different than speed (-1 for instant)

Spawnflag	Details
1	One-Way Only
2	Reverse Start/End
4	All clients

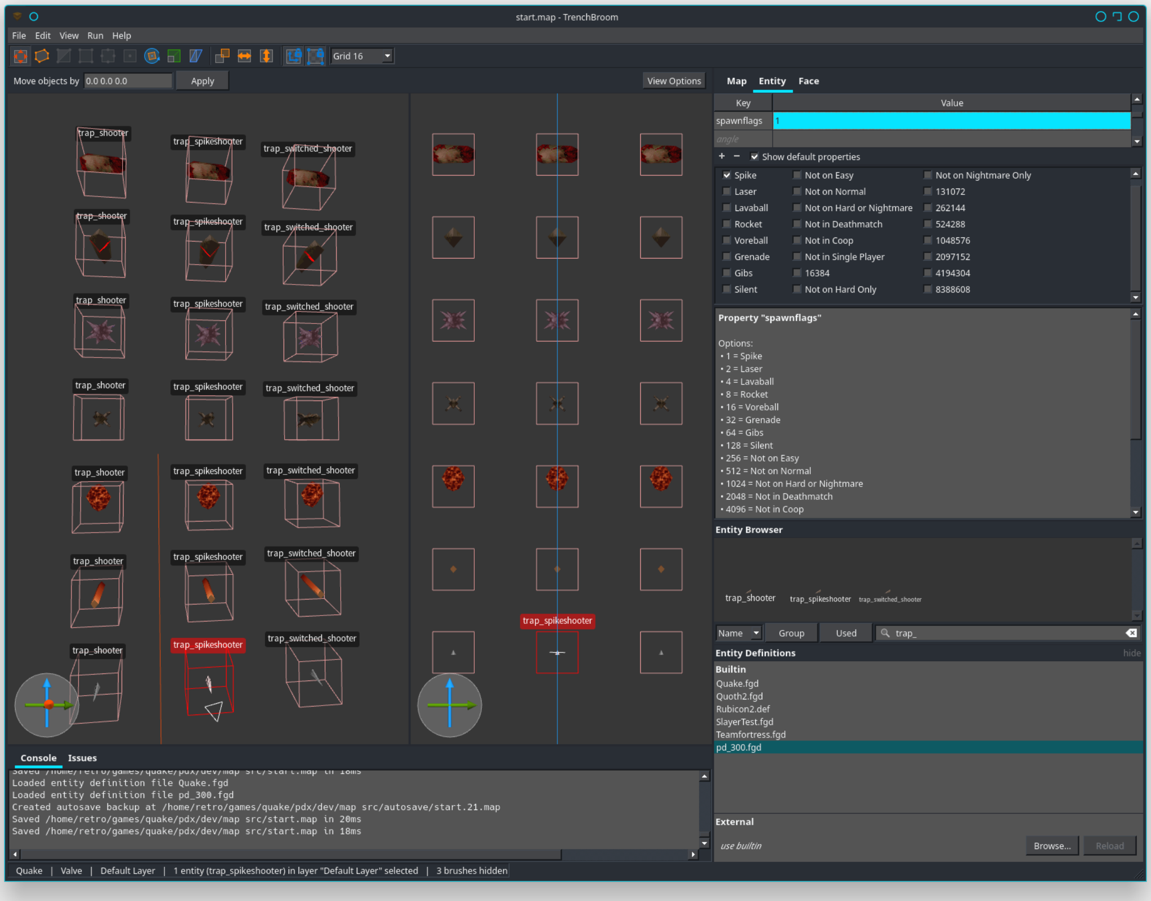
Other Enhanced Entities

Shooters

The original trap_spikeshooter shot only nails and lasers. All three of these entities can now shoot lavaballs, rockets, Voreballs, grenades or gibs. Set the spawnflag accordingly. Use the silent spawnflag if needed. Use the key *state* 0 for initially off, 1 initially on. (0 default) Refer to the table below for specifics on how to trigger these.

Entity	Details
trap_spikeshooter	use a trigger_mulitple to fire
trap_shooter	fires continuously (use killtarget to stop)
trap_switched_shooter	toggle on and off with triggers, buttons

Thanks to xaGe, starting in pd_300, the shooter type selected via spawnflag will display the correct projectile in TrenchBroom 2022.1 and above.



Key	Value
spawnflags	1
angle	

[✓] Show default properties

✓ Spike	Not on Easy	Not on Nightmare Only
Laser	Not on Normal	131072
Lavaball	Not on Hard or Nightmare	262144
Rocket	Not in Deathmatch	524288
Voreball	Not in Coop	1048576
Grenade	Not in Single Player	2097152
Gibs	16384	4194304
Silent	Not on Hard Only	8388608

Property "spawnflags"

Options:
- 1 = Spike
- 2 = Laser
- 4 = Lavaball
- 8 = Rocket
- 16 = Voreball
- 32 = Grenade
- 64 = Gibs
- 128 = Silent
- 256 = Not on Easy
- 512 = Not on Normal
- 1024 = Not on Hard or Nightmare
- 2048 = Not in Deathmatch
- 4096 = Not in Coop

Entity Definitions

Builtin
Quake.fgd
Quoth2.fgd
Rubicon2.def
SlayerTest.fgd
Teamfortress.fgd
pd_300.fgd

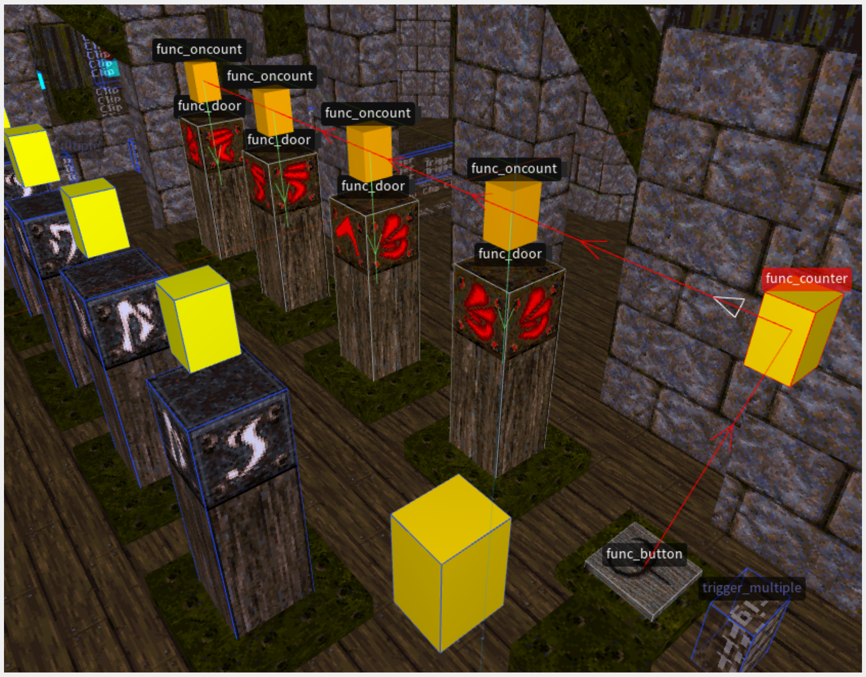
External
use builtin

```
Loaded entity definition file Quake.fgd
Loaded entity definition file pd_300.fgd
Created autosave backup at /home/retro/games/quake/pdx/dev/map src/autosave/start.21.map
Saved /home/retro/games/quake/pdx/dev/map src/start.map in 20ms
Saved /home/retro/games/quake/pdx/dev/map src/start.map in 18ms
```

Quake | Valve | Default Layer | 1 entity (trap_spikeshooter) in layer "Default Layer" selected | 3 brushes hidden

func_counter

This is used to trigger things in a series. You can do some amazing new game play setups with these. Make sure and take some time to play with this one and take a look at the *pd_counter* sample map.



TOGGLE causes the counter to switch between an on and off state each time the counter is triggered. *LOOP* causes the counter to repeat infinitely. The count resets to zero after reaching the value in *count*. *STEP* causes the counter to only increment when triggered. Effectively, this turns the counter into a relay with counting abilities. *RESET* causes the counter to reset to 0 when restarted. *RANDOM* causes the counter to generate random values in the range 1 to *count* at the specified interval. *FINISHCOUNT* causes the counter to continue counting until it reaches *count* before shutting down even after being set to an off state. *START_ON* causes the counter to be on when the level starts. *count* specifies how many times to repeat the event. If *LOOP* is set, it specifies how high to count before resetting to zero. Default is 10. *wait* the length of time between each trigger event. Default is 1 second. *delay* how much time to wait before firing after being switched on. You can see *func_counter* in action when the sarcophagi burst open in *pd_zombies*.map and when used to animate the particle fields in *pd_ladders*.map.

func_oncount

For use as the target of a *func_counter*. When the counter reaches the value set by *count*, *func_oncount* triggers its targets. *count* specifies the value to trigger on. Default is 1. *delay* how much time to wait before firing after being triggered. You can see *func_oncount* in action when the sarcophagi burst open in pd_zombies.map and when used to animate the particle fields in pd_ladders.map.

func_door

Setting *cnt* to 1 will not remove keys from the player's inventory, which mimic's the key behavior of *Doom*. Make sure and add this key / value to all doors and let the player know the default key behavior has changed. e.g. Perhaps a pickup message on the key that reads: "This key works on many doors."

Usually, key doors will remain open after use. However, *func_door* has a new spawnflag called *Doom-style unlock* that will close the door after unlocking it. Setting this spawnflag will set the door to *cnt 1* automatically, retaining the key in the player's inventory.

func_bossgate

From Quakewiki.org: In vanilla Quake *func_bossgate* is used in the start map to block entrance leading to the ending level with the final "Boss". When a player completes all four episodes and collects four item_sigils (Runes), the floor would be removed once the player enters the start map again. *progs_dump* now has a spawnflag that will reverse this functionality.

func_episodegate

From Quakewiki.org: The *func_episodegate* is used in vanilla Quake to create a special *func_wall* in the start level. It is tied to a specific item_sigil (Rune) in the player's possession upon entering a level. In the game it is used to block access to an episode that has already been completed by the player. As such there are four different func_episodegates each one related to a specific item_sigil. progs_dump now has a spawnflag that will reverse this functionality.

func_explobox

An explosive brush entity. Works just like *misc_explobox* but is made from a brush you create as opposed to the default model.

func_fall

A brush that drops and fades away when touched and/or triggered. Add some spice to your jumping puzzles or other scripted sequences! Monsters will not trigger *func_fall* but will be gibbed if one falls on them. *noise* = sound to play when triggered, the default is a switch sound. *wait* = wait this long before falling. Use the *DONT_FADE* spawnflag if desired.

When a *func_fall* brush touches another brush or entity it will stop, which can look odd in certain situations.

func_fall2

This is an enhanced version of *func_fall* that has different properties than the original and a lot more functionality. For example, *func_fall2* will not gib monsters but you can set them to trigger it, unlike the original. These take a bit of set up, so refer to *pd_prefab_func_fall2.map* for more information and examples you can modify for your maps. Whirledtsar has generously added the *Breakable* spawnflag in version 3.0.0.

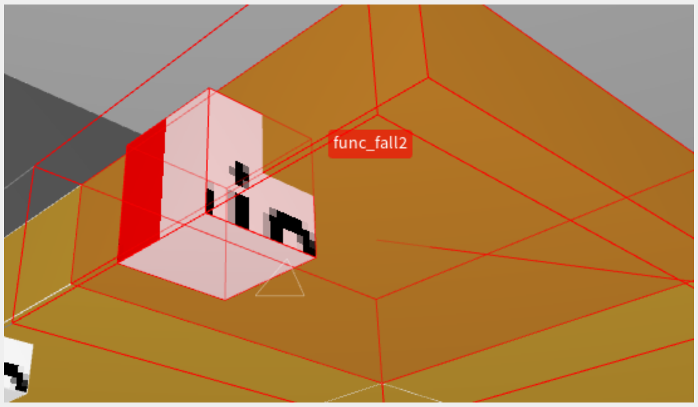
Key	**Details**
wait	how long until the brush begins falling
noise	the sound to make when touched / activated
noise2	the sound to make before it's removed, pain_finished of -1 disables noise2 as the object stays forever
cnt	0 is default behavior, 1 means collisions are disabled while falling, 2 turns the brush into a bouncing entit
count	Only needed with the *Breakable* spawnflag. Set the amount of debris you want (default 5)

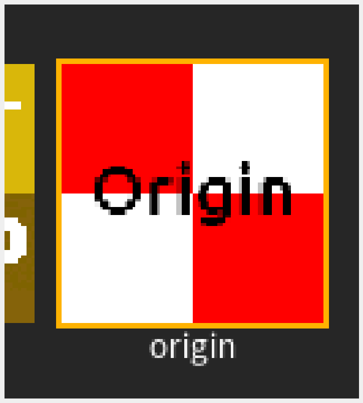
pain_finished	default of 0.01, higher value has the object/brush fade out faster thus in turn affecting how long it stays. -1 stays forever
speed	speed as to how fast something falls per game frame, default is 10, higher values mean faster falling. Only for cnt of 1. Recommended to use lip for max fall speed with cnt 0 and 2 as they follow Quake's default gravity
lip	maximum fall speed that can be achieved, caps 'speed' variable. Default is -800
avelocity	brush spins when activated using X, Y, Z vector coordinates. *cnt* of 2 ignores avelocity. Use an origin brush at the center of your brush(es) for proper spin!
spawnflags	Player or Monster only flags. Also, func_fall2 can be set to Breakable

If you need a falling brush to remain solid, use the original *func_fall* and not *func_fall2*.

When the spawnflag is set to *Breakable*, you can set one of 32 styles from the *Style* dropdown menu just as with func_breakable. One key difference is that *cnt* is already used in *func_fall2* for a different function. So set the *count* key to the amount of debris you want. (default is 5)

When using the *avelocity* key, add an origin textured brush at the point you want the brush to rotate around. This brush must be part of the *func_fall2* brush entity. In TrenchBroom you would control click both brushes and then right click to make them a *func_fall2*.





func_togglewall

Creates an invisible wall that can be toggled on and off. *START_OFF* spawnflag means the wall doesn't block until triggered. *noise* is the sound to play when the wall is turned off. *noise1* is the sound to play when the wall is blocking. *dmg* is the amount of damage to cause when touched. You can see an example of this in the *pd_ladders* example map above the barred teleport area.

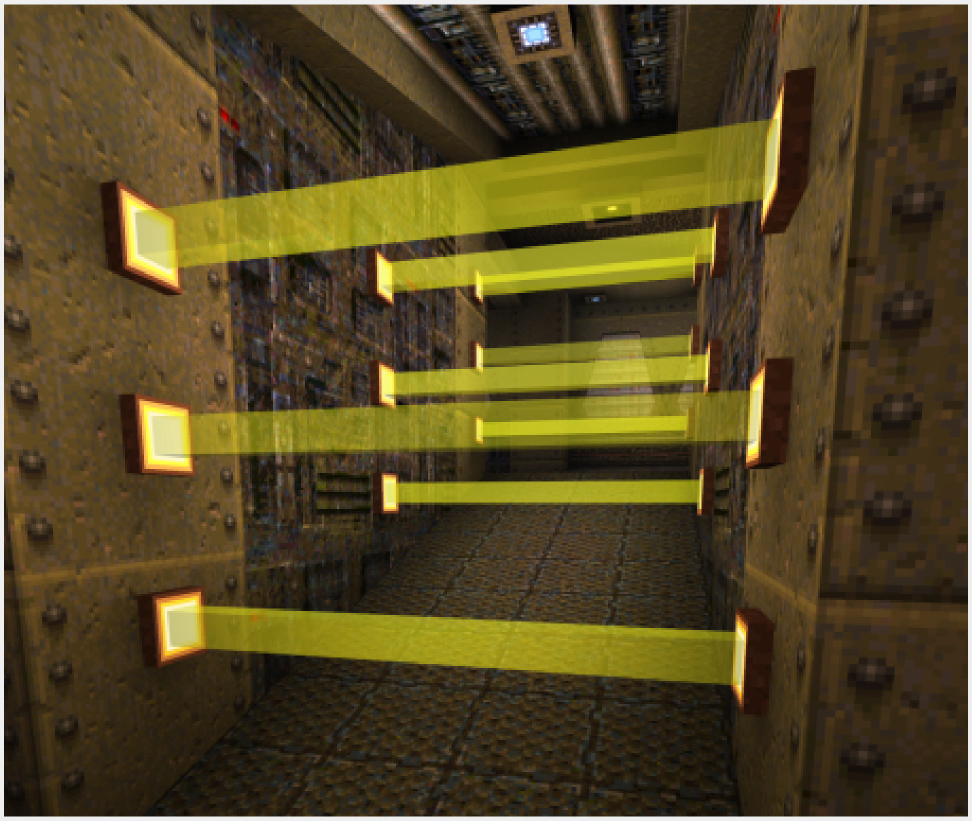
func_togglevisiblewall

A brush model that can be toggled on and off. Behaves much like a traditional func_wall in any other way, but you can target it to toggle visible/invisible. If the entity has a switchable shadow it also toggles. *Spawnflag 1* will start the brush off. *Spawnflag 2* sets it as non-solid.

func_laser

A togglable laser, hurts to touch, can be used to block players. START_OFF: Laser starts off. LASER_SOLID: Laser blocks movement while turned on. Keys: *dmg* damage on touch. default 1 *alpha* approximate alpha you want the laser drawn at. default 0.5. alpha will vary by 20% of this value. *message* message to display when activated *message2* message to display when deactivated.

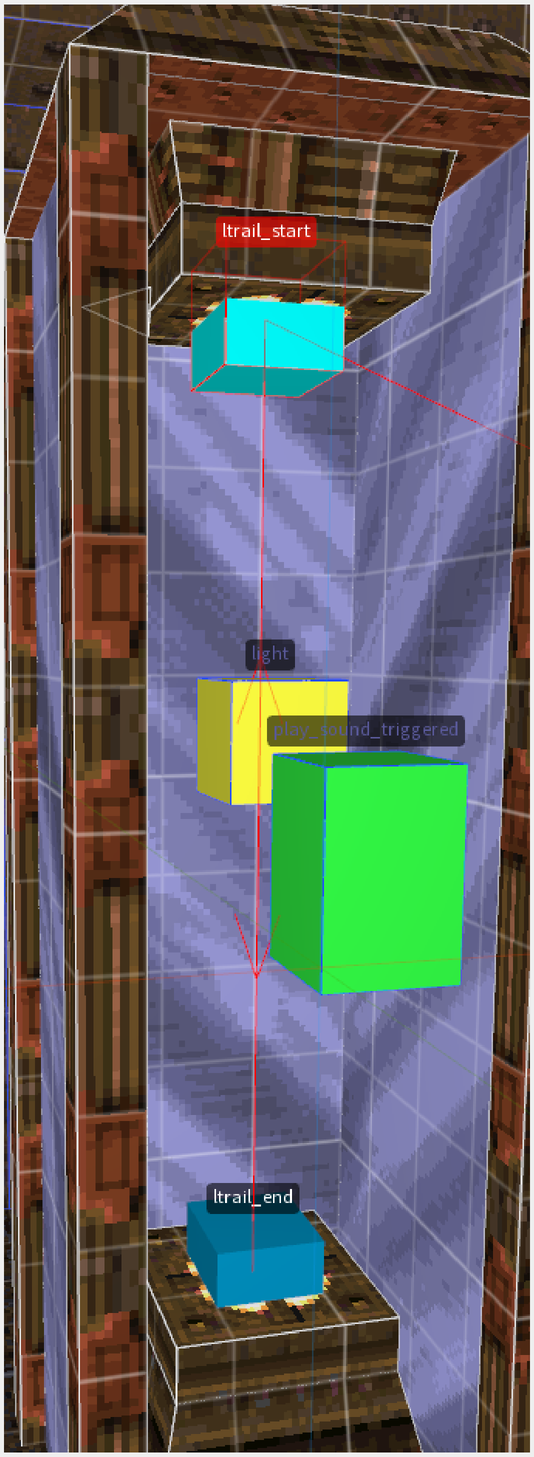
Use fullbright textures with func_lasers to ensure they stand out against darker backgrounds. Kreathor's Prototype wad has a good selection.

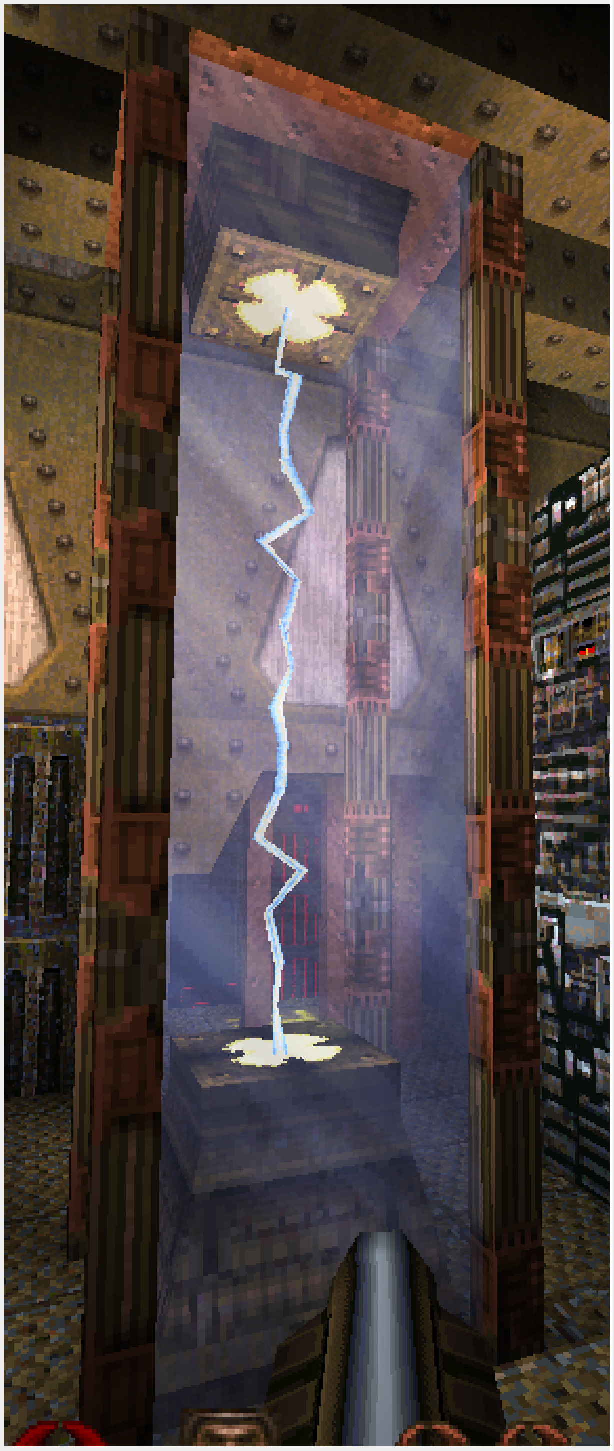


Lightning Entities

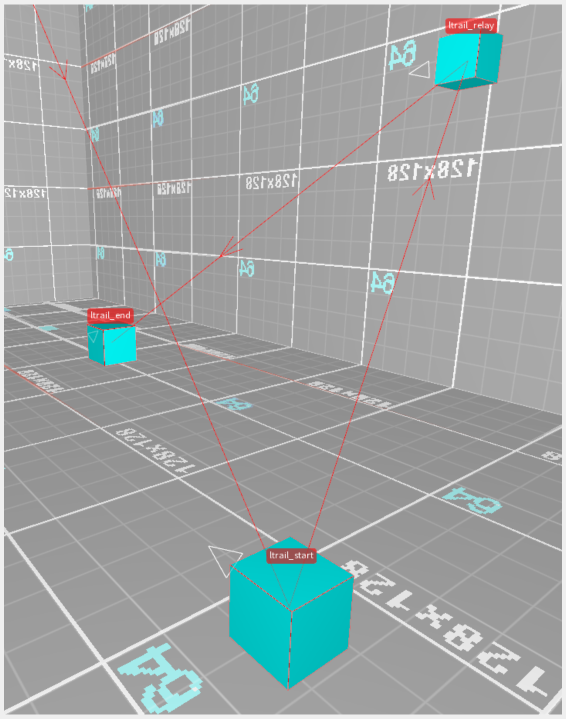
ltrail_start
ltrail_relay
ltrail_end

These lightning trail entities can be used for traps, decoration or for other scripted events. For the example below there are two entities. *ltrail_start* and *ltrail_end*, they are targeting each other.





If you want a chain of lightning events you would use a number of *ltrail_relays* between the start and end targeting one to the other, much like you would a *path_corner* with a *func_train*.



The key / values are weirdly named in these entities. This is a quirk of QuakeC, where coders try to limit the amount of fields used by "recycling" unused fields to save memory.

ltrail_start Starting point of a lightning trail. Set *currentammo* to the amount of damage you want the lightning to do. Default is 25. Set *frags* to the amount of time before the next item is triggered. Default is 0.3 seconds. Set *weapon* to the amount of time to be firing the lightning. Default is 0.3 seconds. Set *sounds* to 1 for no sound. (Yes, it is weird.) Set the *TOGGLE* spawnflag if you want the lightning shooter to continuously fire until triggered again. Set the *START ON* spawnflag to have the lightning shooter start on. Do NOT use both these spawnflags at once.

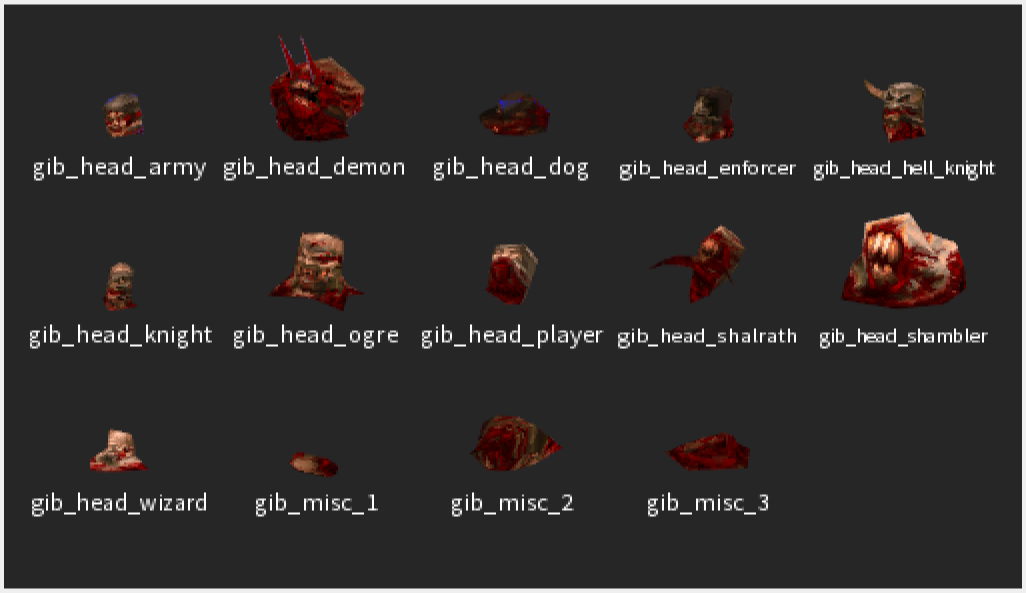
ltrail_relay Relay point of a lightning trail. Set *currentammo* to the amount of damage you want the lightning to do. Default is 25. Set *frags* to the amount of time before the next item is triggered. Default is 0.3 seconds. Set *weapon* to the amount of time to be firing the lightning. Default is 0.3 seconds Unfortunately, ltrail_relay entities cannot be set to silent.

ltrail_end Ending point of a lightning trail. Does not fire any lightning. Set *frags* to the amount of time before the next item is triggered. Default is 0.3 seconds.

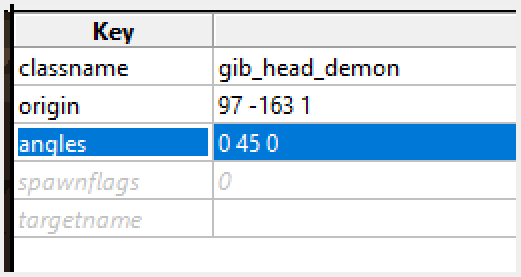
To have a continuously firing bolt between two points, have a *ltrail_start* and *ltrail_end* targeting each other in a loop and set *frags* to -1. The sound this makes is not ideal, so consider making these *silent* and use a *play_sound_triggered* with a custom looping sound. This is shown in the *pd_lasers* sample map. In the devkit, sounds/dump/elec22k.wav is included for this very reason.

gib_(classname)

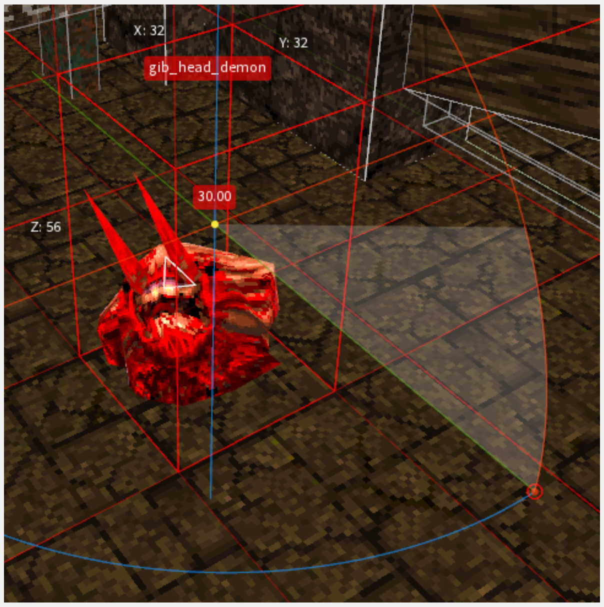
Easily add these bloody decorations to your map. (Also see *monster_dead_(classname)* below. You can use the SOLID spawnflag to enable collision on the model but clip brushes will work even better.



If you are using TrenchBroom take extra care when rotating these entities. The way TrenchBroom handles rotations for custom models requires a work around in some cases. If you want to simply rotate the gib model around the z axis there is no problem. However, if you wish to rotate the model in the X and Y or any combination, you will need to manually type in X Y and Z values *before* using the rotate tool. To do this, add the *angles* key (with an s) and type in something like 0 45 0 as the values. Then you can select the rotate tool and adjust the other values using the widget. Keep in mind the values 0 0 0 will not work. Also the *angle* key (no s) should be blank or set to 0 when using the *angle*s key.



Key	
classname	gib_head_demon
origin	97 -163 1
angles	0 45 0
spawnflags	*0*
targetname	



monster_dead_(classname)

e.g. *monster_dead_ogre* More decorations for your maps. You can use the SOLID spawnflag to enable collision on the model but clip brushes will work even better. Keep in mind the same issue with rotation mentioned above applies to these models as well.

monster_dead_army
monster_dead_demon
monster_dead_dog
monster_dead_enforcer
monster_dead_hell_knight
monster_dead_knight
monster_dead_ogre
monster_dead_shalrath
monster_dead_shambler
monster_dead_wizard

Worldspawn

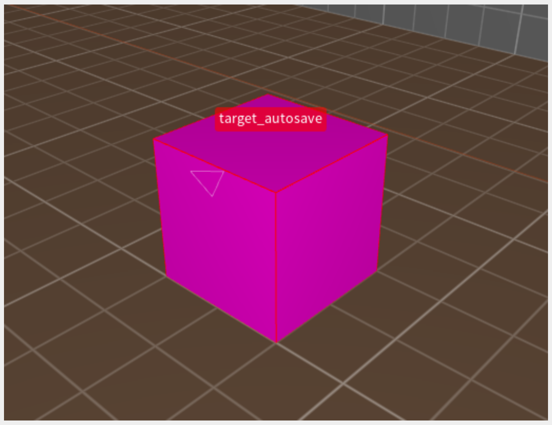
Features a *reset_items* key (default 0). Set to 1 to make the player start with the default shotgun and ax. Set to 2 for an ax only start. Starting with *progs_dump* 3, you can set default models. This allows you to on a map-by-map basis select "default" health / ammo / armor mdls. **IMPORTANT:** If you are replacing the armor model, make sure it uses multiple skins as the original does, or you will get an error on Yellow or Red armor in-game.

Worldspawn Key	**Details**
h_vial_mdl	health vial model
h_25_mdl	health box model
h_15_mdl	rotten health box model
h_mega_mdl	Megahealth model
s_sm_mdl	shell box model
s_lg_mdl	large shell box model
n_sm_mdl	spike box model
n_lg_mdl	large spike box model
r_sm_mdl	rocket box model
r_lg_mdl	large rocket box model
c_sm_mdl	cell box model
c_lg_mdl	large cell box model
a_shr_mdl	armor shard model
a_grn_mdl	Green Armor model
a_ylw_mdl	Yellow Armor model
a_red_mdl	Red Armor model

target_autosave

The following text is from the Copper documentation by Lunaran:

Saves the game when triggered by a player. Never appears in multiplayer. The messages tend to stomp any other prints on screen in most Quake engines, So use a delayed *trigger_relay* if you fire this from an important pickup or trigger that puts text on screen more important than the autosave blurb.



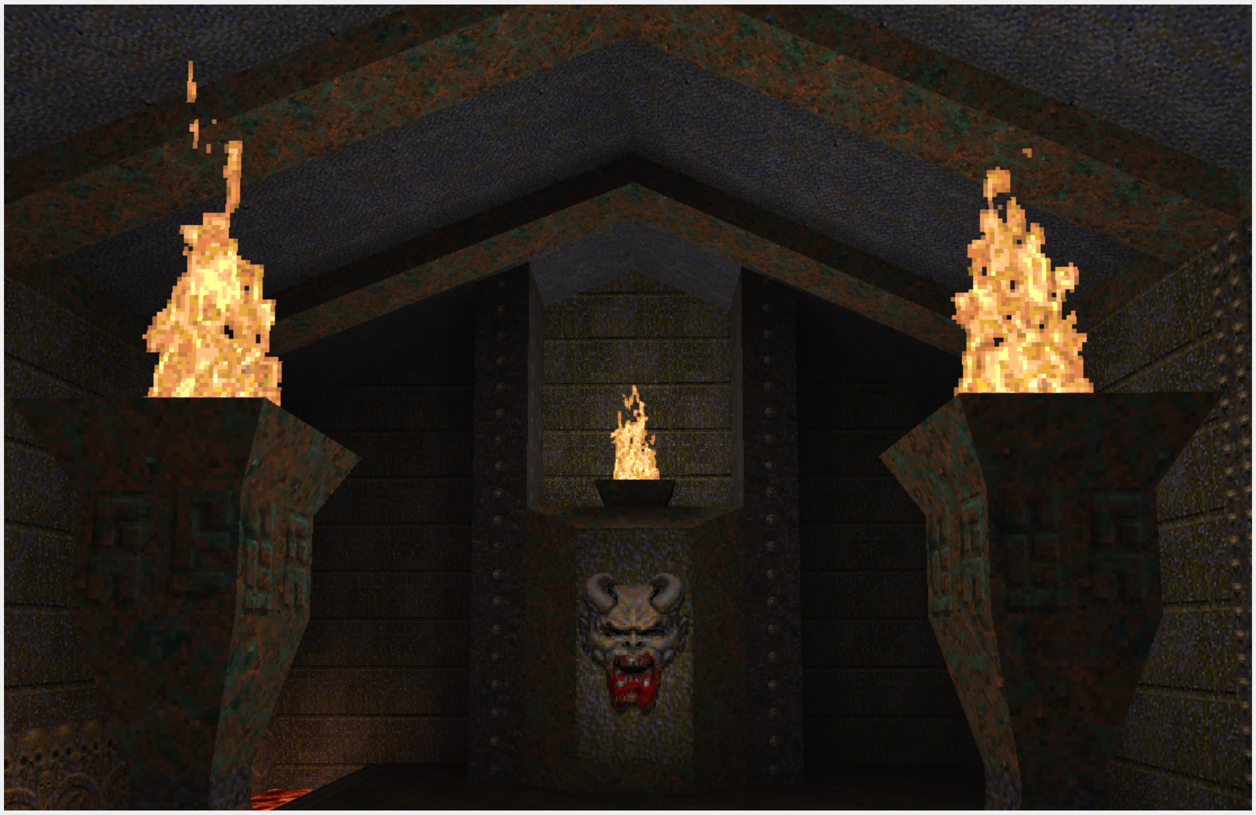
By default, the *quake.rc* file in *progs_dump* automatically binds the F8 key to load autosaves similarly to how you can *quicksave* and *quickload* in the vanilla game with F6 and F9 respectively.

light_candle

A simple light emitting candle from Mission Pack 2. You can place them into the ground for shorter varieties and rotate them for variety.

light_sprite_flame

New in *pd_300*, this is a drag and drop light entity with an animated flame sprite created by starshipwaters. Currently, the sprite cannot be displayed in the TrenchBroom entity browser. The start of the 13 frames of animation is randomized so each iteration will look slightly different.



Train System

Starting with version 3, progs_dump has reworked trains that offer many more options including the ability to animate models (including monsters) along a series of *path_corners*, trigger events at *path_corners*, easier alignment options and more. There are now two train options: *func_train* and *misc_modeltrain*. The new system was created by bmFbr based on the Alkaline devkit and the custom alignment feature was added by TheRektafire. See *prefab_trains* for an example of these new entities.

The 3.0.0 version of *func_train* is incompatible with prior versions of the devkit. The original *RETRIGGER* spawnflag was 1. It has been reworked and now is spawnflag 2 *Move on trigger.* In addition, the *noise* keys have changed and have new functionality.

func_train

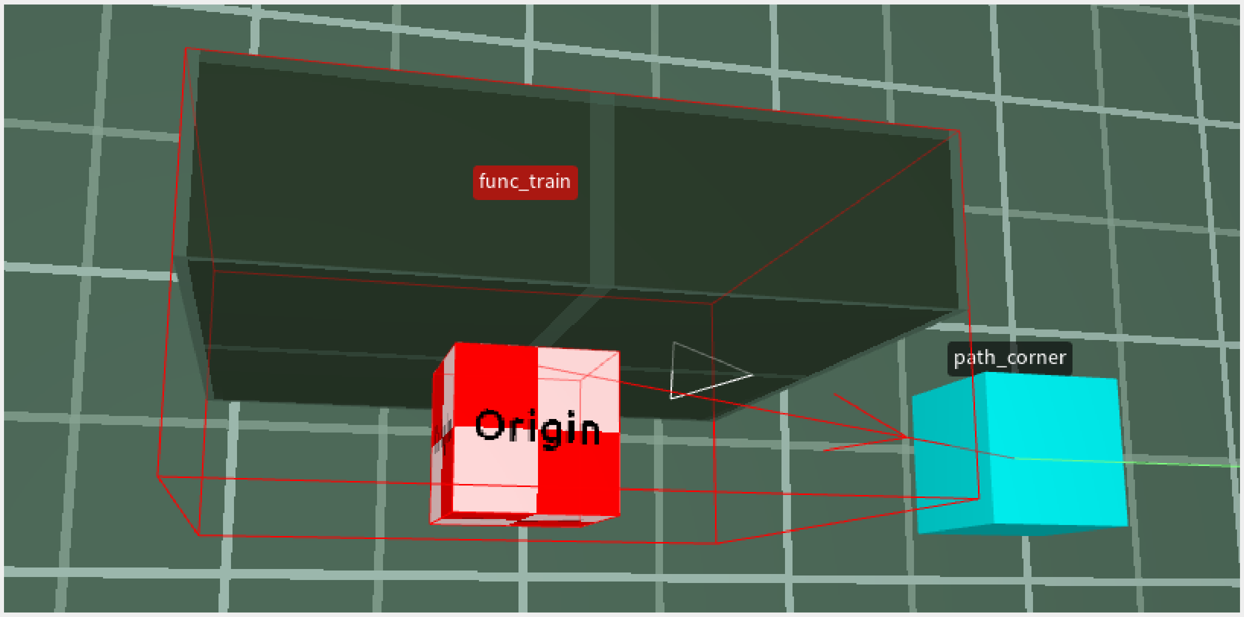
Trains are moving platforms that players, monsters and items can ride on. The target's origin specifies the min point of the train at each corner but you can now use an origin brush as part of the train to make alignment easier (see below). The train spawns at the first *target* it is pointing at. If given a *targetname*, it'll only start moving after triggered. *path_corners* with a *wait* -1 value will stop the train. Trigger the *func_train* again to resume its path. Also, a train can fire targets when it arrives at a specific *path_corner* by using the path_corner's target2-4 keys. So for example, you could spawn in a monster with *target2* when a train arrives at a specific *path_corner* with a matching *targetname2*.

Key	**Details**
pausetime	sets a default waiting time when a value is not defined in *path_corners*
speed	sets the initial moving speed. If the train encounters a path_corner with its 'speed' field set it will move at that speed afterwards
noise	Custom stopping sound (if set, overrides 'sounds'
noise1	Custom moving loop sound (if set, overrides 'sounds'
noise2	Custom corner sound, for when the train crosses a path_corner without stopping)
sounds	0 - silent, 1 - ratchet metal, 1 - base door
Spawnflags	**Details**
Move on trigger	will force the train to resume its path when triggered, even when temporarily waiting at a path_corner

Stop on trigger	will stop the train at the next path_corner when triggered. Trigger it again to resume
Custom alignment	allows you to directly specify how the train is offset relative to its *path_corners* using an 'origin' brush instead of using the vanilla corner alignment
Non-Solid	collision is disabled

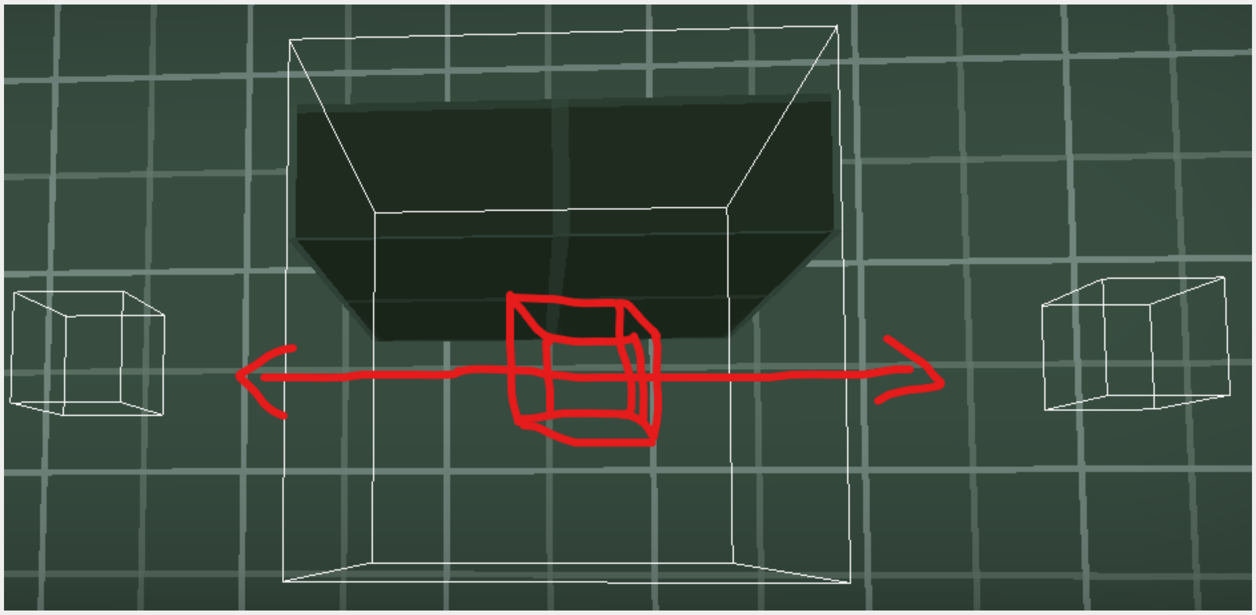
The *Move on trigger* and *Stop on trigger* flags above are mutually exclusive and will cause an objerror when selected simultaneously.

Similar to *func_fall2*, you can add an origin brush to a *func_train* to change its functionality. In this example, setting the *Custom alignment* spawnflag uses the origin brush as the “center” of the train as opposed to the *mins* (corner) of the model.

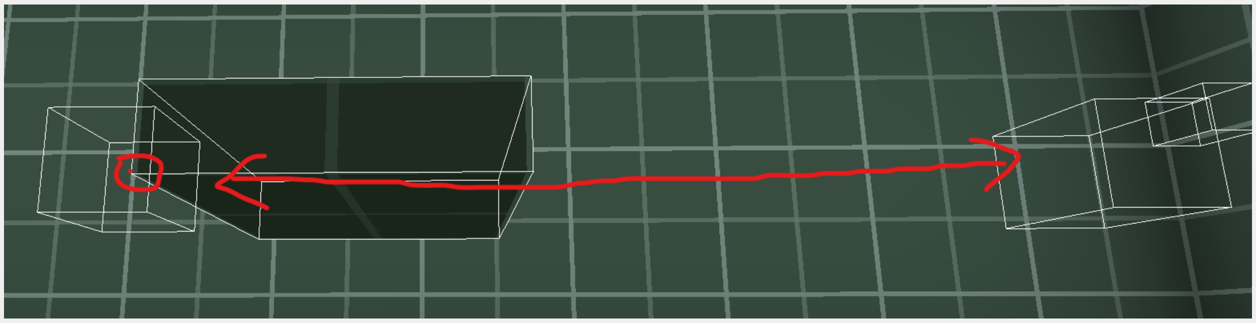


On the next page you can see the two different methods to align a train to path corners.

Using an origin brush and the *Custom alignment* method as seen in-game with *r_showbboxes* set to 1.



Using the standard method of aligning via the min corner of the train.



path_corner

These are waypoints for *func_train* and *misc_modeltrain* entities. In *progs_dump* they have additional keys that can alter some settings on those two entities. For example, setting a faster *speed* key on a *path_corner* will change the speed of a *func_train* that arrives there. Many of these new keys are specific to *misc_modeltrain*.

If you want to explicitly set frame 0 in any *_frame fields, you must use -1 instead, or else the code understands 0 as a blank field.

Key	Details
animtype	Animation type: how to play back frames while stopped at a waypoint 1 : "One way" 2 : "Back and forth"
animtype2	Animation type: how to play back frames while moving between waypoints 1 : "One way" 2 : "Back and forth"
first_frame	First frame of animation while stopped at a

	waypoint. Use -1 for frame 0
first_frame2	First frame of animation while moving between waypoints. Use -1 for frame 0
frtime	Speed of animation while stopped at a waypoint
frtime2	Speed of animation while moving between waypoints
last_frame	Last frame of animation while stopped at a waypoint. Use -1 for frame 0
last_frame2	Last frame of animation while moving between waypoints. Use -1 for frame 0
multiplier	Turning speed multiplier. Set to -1 to turn instantly. This setting persists on subsequent waypoints until changed.
noise	Path to custom stopping sound for this point (e.g soldier/idle.wav)
noise2	Path to custom corner sound for this point, for when the train crosses a path_corner without stopping (wait -2)
wait	Waiting time at this point. A positive value directly sets a waiting time in seconds. 0 (or not defined) uses the *pausetime* field value in the *func_train* (which defaults to 0s). -1 makes the train stop at this point until re-triggered -2 forces the train to not wait here even when *pausetime* is set on the func_train.

misc_modeltrain

This works the same way as a *func_train*, but using external models. **This allows you to animate monster models for in-game set pieces or cutscenes!**

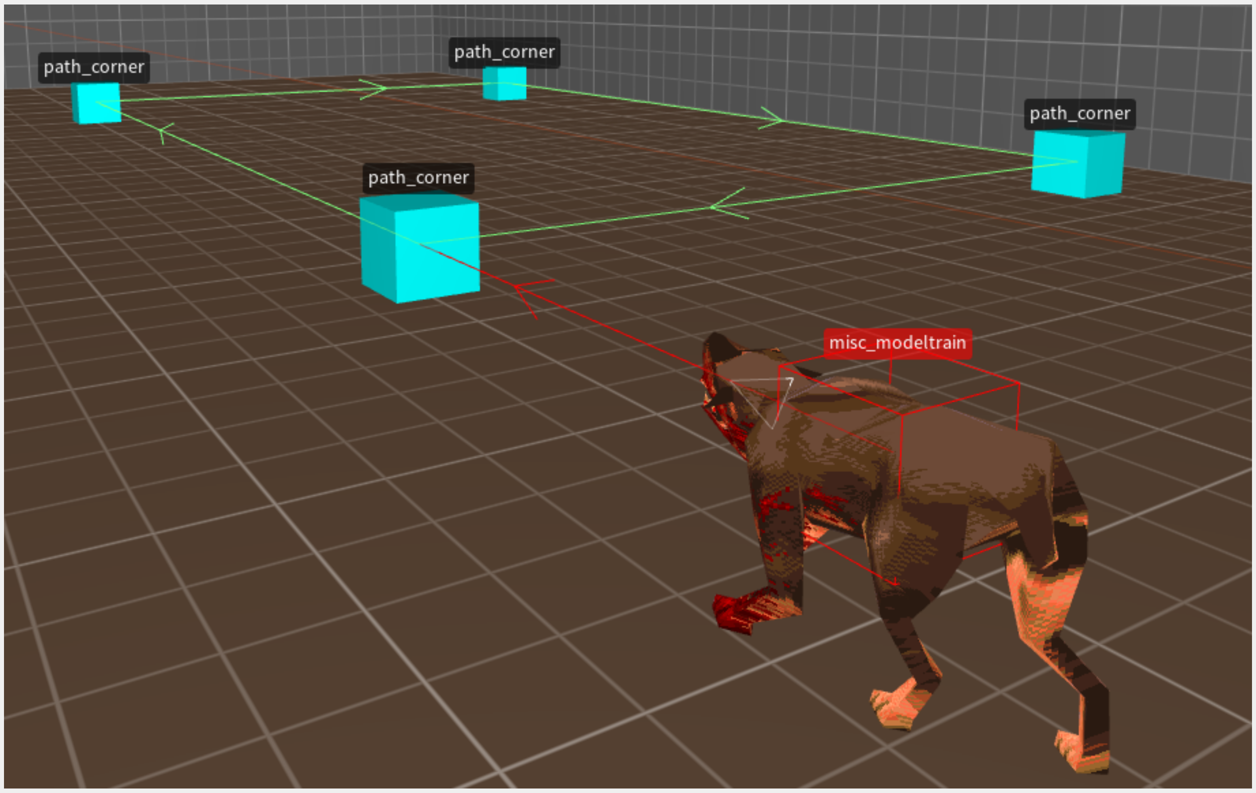
The model can smoothly orient itself towards the next path_corner when moving. The turning speed is automatic and based on the movement speed, but you can set a multiplier for the turning speed with the *multiplier* key. Set it to -1 to turn instantly. You can define two different frame ranges and animation speeds, for when the model is stopped and moving. If you don't need different stopped/moving animations, only setting *first_frame/last_frame/frtime* is enough. If you don't need any animation at all, just set *first_frame* to the desired frame and all other fields can go blank.

If you want to explicitly set frame 0 in any *_frame fields, you must use -1 instead, or else the code understands 0 as a blank field.

Key	**Detail**
cmaxs	Upper bounding box point (default is 8 8 8)
cmins	Lower bounding box point (default is -8 -8 -8)
first_frame	First frame of animation while stopped at a waypoint. Use -1 for frame 0
first_frame2	First frame of animation while moving between waypoints. Use -1 for frame 0
last_frame	Last frame of animation while stopped at a waypoint. Use -1 for frame 0
last_frame2	Last frame of animation while moving between waypoints. Use -1 for frame 0
mdl	Path to model file (e.g. progs/soldier.mdl)
multiplier	Turning speed multiplier. Set to -1 to turn instantly. This setting persists until changed by a key set in a *path_corner*.
noise	Path to custom stopping sound (e.g soldier/idle.wav) if set, overrides *sounds* key
noise1	Path to custom moving loop sound (if set, overrides *sounds* key)
noise2	Path to custom corner sound, for when the train crosses a path_corner without stopping
speed	Sets the initial moving speed in units per second. If the train encounters a *path_corner* with its *speed* key set, it will move at that speed afterwards.

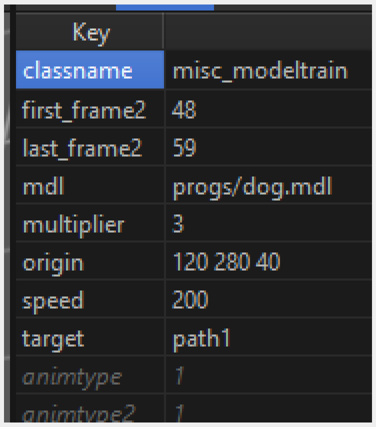
animtype	Animation type: how to play back frames while stopped at a waypoint 1 : "One way" 2 : "Back and forth"
animtype2	Animation type: how to play back frames while moving between waypoints 1 : "One way" 2 : "Back and forth"
dmg	Damage on crush (default is 2)
frtime	Speed of animation while stopped at a waypoint
frtime2	Speed of animation while moving between waypoints
pausetime	Default waiting time when a value is not defined in *path_corners*
sounds	Options: 0 (None) 1 (Ratchet Metal) 2 (Base door)
Spawnflags	**Details**
Move on trigger	Will force the train to resume its path when triggered, even when temporarily waiting at a path_corner
Stop on trigger	Will stop the train at the next path_corner when triggered. Trigger it again to resume
Non-Solid	No collision
No rotation	Model will not rotate when moving towards path_corners.
Rotate Y only	Model only rotates around the Y axis, so pitch/roll angles are kept as initially set (useful to fake walking NPCs)

Here's a simple example of how a *misc_modeltrain* can be set up to animate progs/dog.mdl running between four *path_corners*.



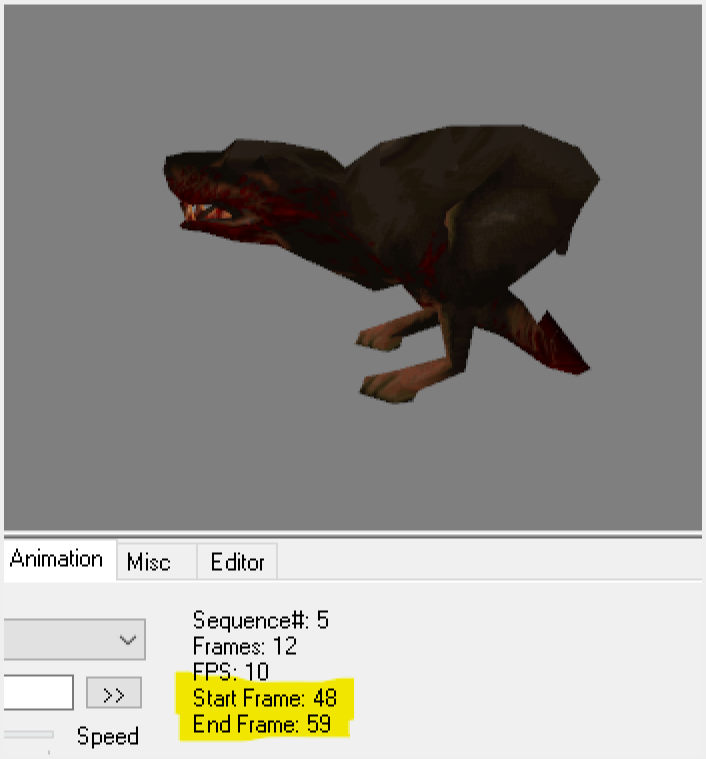
1. Set up path corners as you would with a regular *func_train*. You can see a video tutorial on func_trains here.

2. Next set up a *misc_modeltrain* with the following keys:



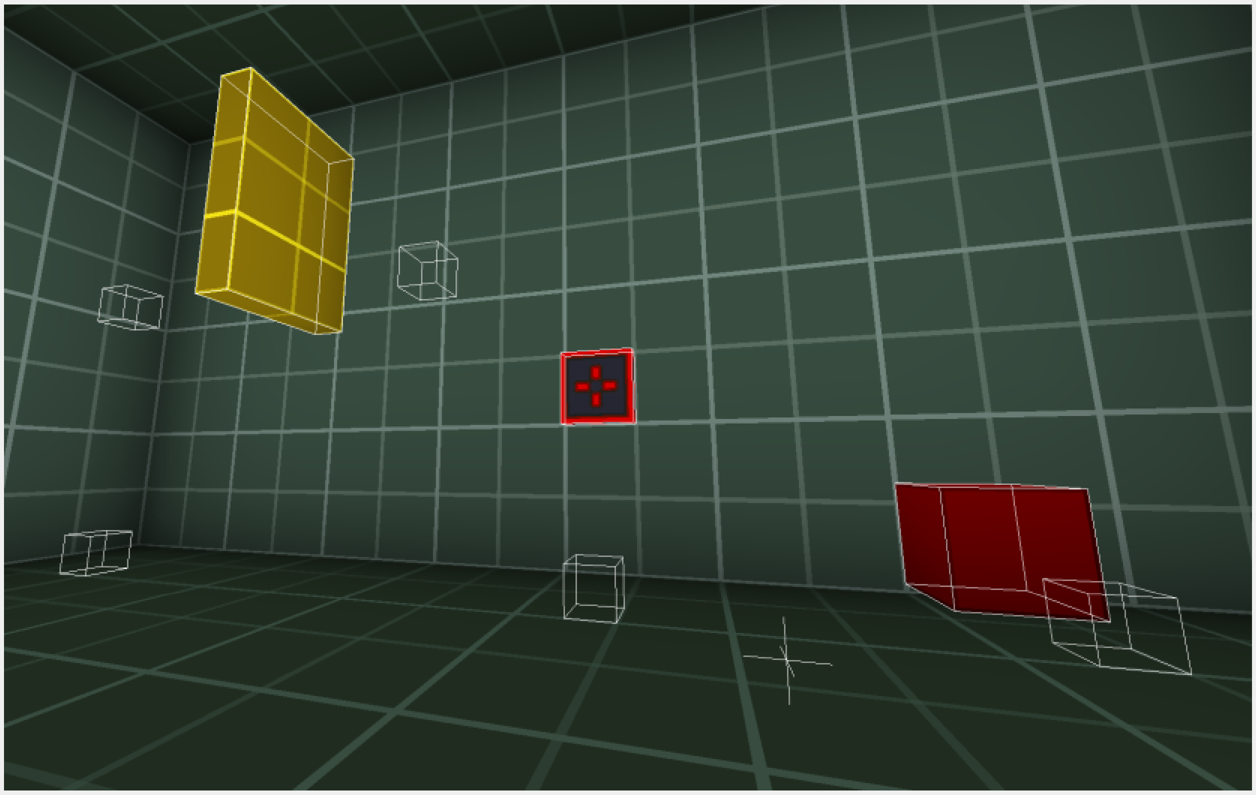
Key	
classname	misc_modeltrain
first_frame2	48
last_frame2	59
mdl	progs/dog.mdl
multiplier	3
origin	120 280 40
speed	200
target	path1
animtype	*1*
animtype2	*1*

As you can see, *first_frame2* is the starting frame of the dog model's run cycle and *last_frame2* is the last. You can use Quake 1 Model Viewer (or the FTEQW engine) to examine animation frames as seen below. I changed the *multiplier* to 3 which makes the dog turn more quickly to the next *path_corner*. A *speed* of 200 seemed pretty close to a *monster_dog's* running speed.



Sequence#: 5
Frames: 12
FPS: 10
Start Frame: 48
End Frame: 59

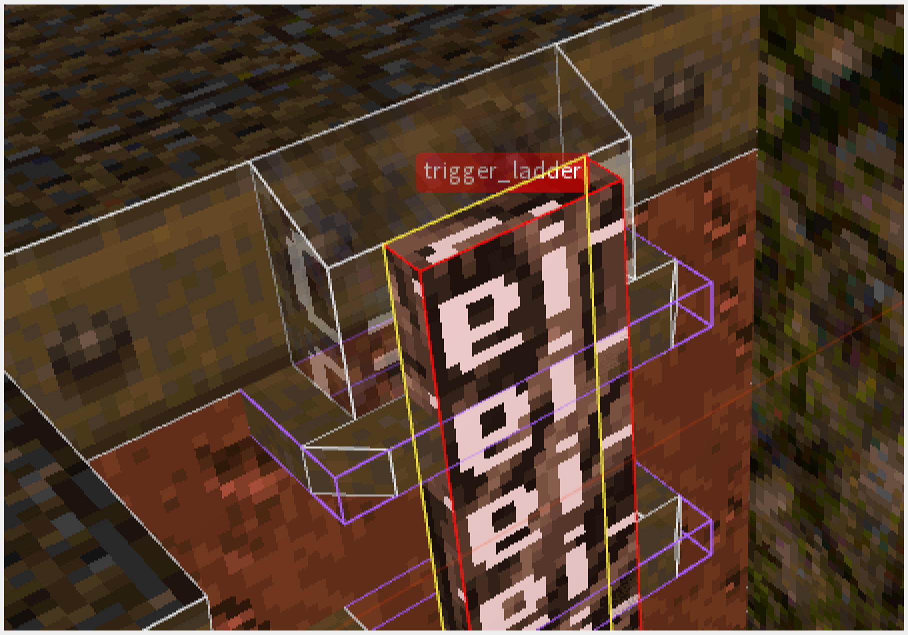
Much more advanced setups with models that start, stop and trigger events can be seen in the *prefab_trains* sample map.



Ladders

trigger_ladder

Create a small *trigger_ladder* brush covered with the trigger texture. Make sure the outside edge of the brush is flush with your ladder geometry. Set the *angle* key to the direction the player is facing when approaching the ladder. You can use a wedge shaped clip brush to smooth out any “sticky” movements at the top of the ladder as seen below. Please refer to *pd_ladders.map* for examples.

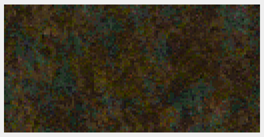
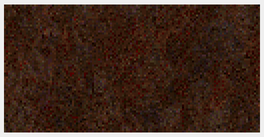
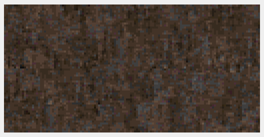
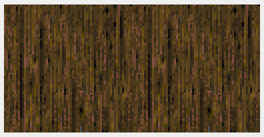
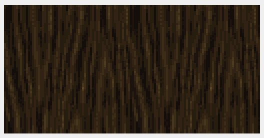
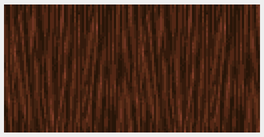
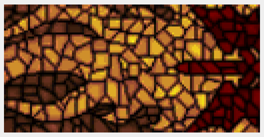
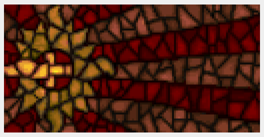
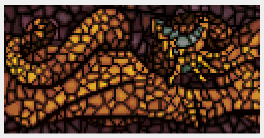
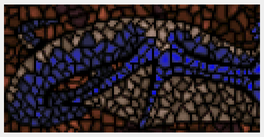
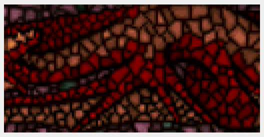


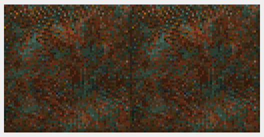
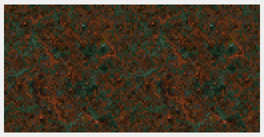
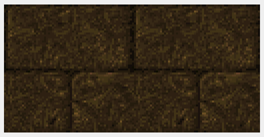
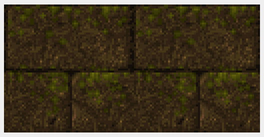
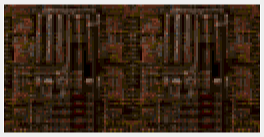
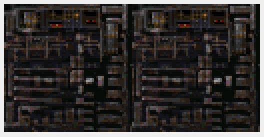
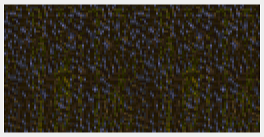
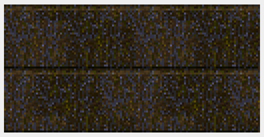
Breakables

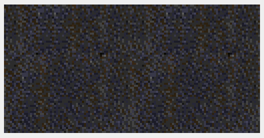
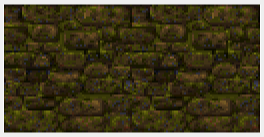
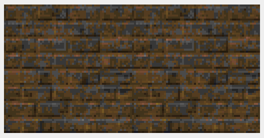
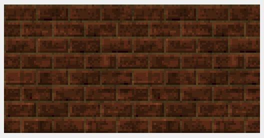
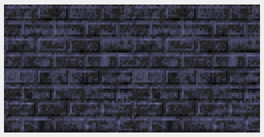
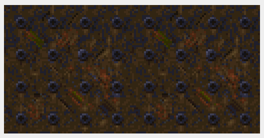
func_breakable

Breakables may seem overwhelming to new mappers, however it’s not as complicated as it looks. Also, there are two methods to choose from. One is the *Built-in* (easy) method and the other is the *Custom* method (more flexible.)

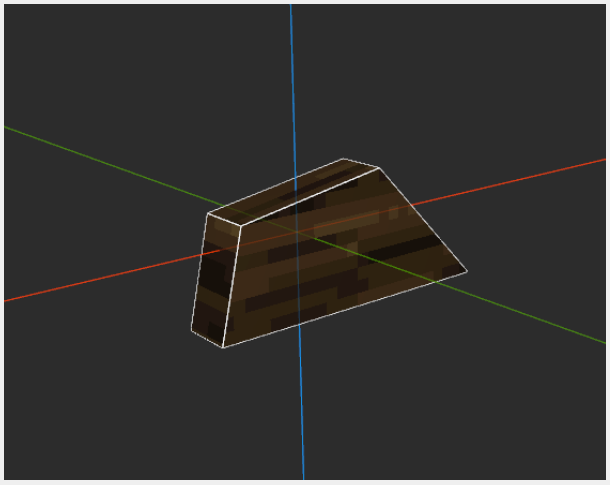
The Built-in method: Create your brush and make it a *func_breakable*. You can ignore any keys that begin with *brk* or *breakable*. Those are used with the custom method. With the built-in method you will set the *style* to one of thirty-two options from the original id1 texture wads listed below. By default, the breakable spawn 5 pieces of debris. You can change this amount with the *cnt* key/value. The default *health* of the brush is 20.There are placeholder sounds but you can use the *noise1* key to set a custom sound path. If you give the breakable a *targetname* it will only break when triggered. Use the *Explosion* spawnflag for an explosive brush. Use the *dmg* key to set a custom damage value. You can also use the *No Monster Damage* spawnflag to keep monsters from breaking the brush. As with monsters, you can use the *drop_item* key to spawn a Silver or Gold key, vial or armor shards upon breaking.

Style	Texture basis	Image	Description
0	custom		Green Metal (default)
1	custom		Red Metal
2	custom		Concrete
3	wood1_1		Pine wood
4	wizwood1_3		Brown wood
5	dung01_2		Red wood
6	window02_1		Stained Glass Yellow Flames
7	window01_4		Stained Glass Red Rays
8	window01_3		Stained Glass Yellow Dragon
9	window01_2		Stained Glass Blue Dragon
10	window01_1		Stained Glass Red Dragon

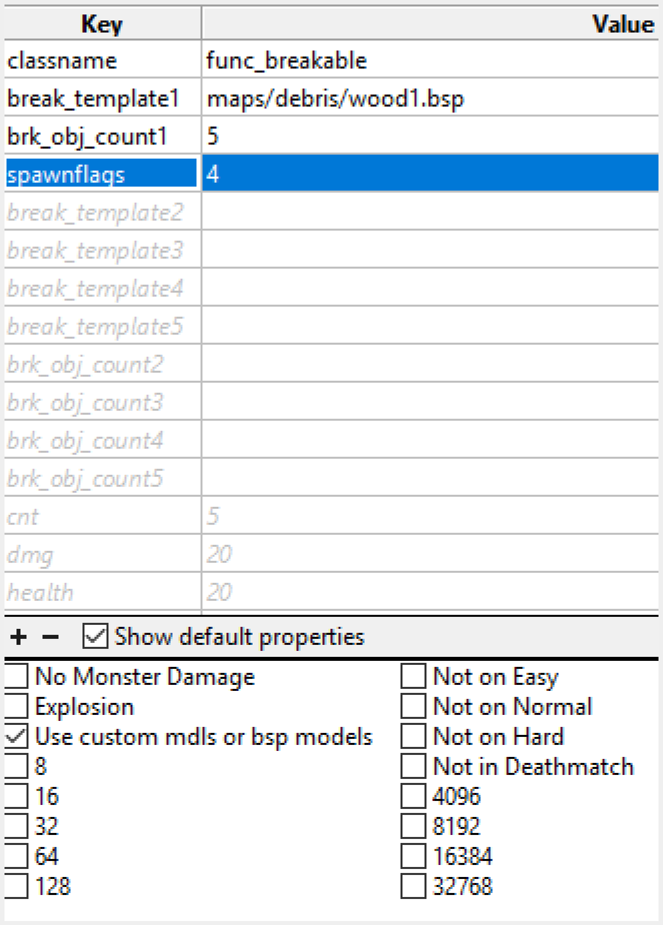
11	cop2_3		Light Copper
12	cop1_1		Dark Copper
13	wiz1_4		Tan Bricks Large
14	wbrick1_5		Brown Bricks Large
15	wswamp2_1		Green Bricks Large
16	tlight08		Generic Light Brown
17	comp1_5		Red Brown Computer
18	comp1_1		Gray Black Computer
19	metal4_5		Blue Green Metal
20	metal4_4		Blue Green Runic Wall
21	metal2_2		Brown Metal

22	metal1_3		Dark Brown Metal
23	metal1_2		Medium Brown Metal
24	m5_8		Blue Metal
25	city8_2		Green Stonework
26	city6_7		Blue Stonework
27	city2_8		Brown Bricks
28	city2_7		Tan Blue Bricks
29	city2_1		Red Bricks
30	city2_5		Blue Bricks
31	wizmet1_2		Metal Rivets

The Custom Method: This method uses external, custom models (.mdl format) or brush models (.bsp format) instead of the built-in system. You can make small pieces of debris by shaping them in a level editor and compiling them into a .bsp (See *Creating Debris* below.)



You can also use .bsps from other mods (check if you have permission to do so.) In the example below, we are only using one piece and duplicating it when the brush is "broken." Set the Use custom mdls or bsp models spawnflag to enable this mode. Then set the path to the .bsp or model in *break_template1*. The *brk_obj_count1* determines how many instances of that bsp will be used. You can have 5 different pieces of debris total (break_template1-5) and control how many instances each of those templates spawns with *brk_obj_count1-5. noise1* is the path to the sound when breaking. *Style* and *cnt* are not used in this method but *health* and *dmg* are.



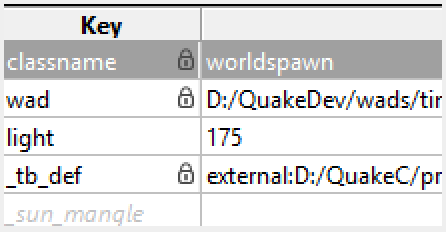
Key	Value
classname	func_breakable
break_template1	maps/debris/wood1.bsp
brk_obj_count1	5
spawnflags	4
break_template2	
break_template3	
break_template4	
break_template5	
brk_obj_count2	
brk_obj_count3	
brk_obj_count4	
brk_obj_count5	
cnt	5
dmg	20
health	20

☑ Show default properties

☐ No Monster Damage	☐ Not on Easy
☐ Explosion	☐ Not on Normal
☑ Use custom mdls or bsp models	☐ Not on Hard
☐ 8	☐ Not in Deathmatch
☐ 16	☐ 4096
☐ 32	☐ 8192
☐ 64	☐ 16384
☐ 128	☐ 32768

Creating debris

You can create *break_templates* as tiny maps and compile them into bsps. Create one piece at a time as their own map file. Create the debris at the center of the map (origin 0, 0, 0) Compile with qbsp.exe and light. No need to run vis.exe on these. You can add a *light* key/value to the Worldspawn to uniformly light the piece of debris.



Key	
classname	worldspawn
wad	D:/QuakeDev/wads/tir
light	175
_tb_def	external:D:/QuakeC/pr
_sun_mangle	

Place these pieces in your maps folder or a subfolder under maps called debris or breakables and remember to include these when you distribute your map.

If you want debris to fall during an earthquake or similar event, use a skip texture to create an invisible breakable. Make sure the player cannot touch the brush, as skip textured brushes have collision. Clip textures won't work for this. You can see an example of this in the *pd_bosses* sample map.

Effect Entities

You can trigger the following visual effects.

Effect	**Details**
play_explosion	grenade explosion, causes damage
play_spawnexpl	Spawn death explosion, causes damage
play_lavalsplash	large particle effect, can have custom sound
play_brlight	Toggles a bright lighting effect on or off.
play_dimlight	Toggles a lighting effect on or off.
play_mflash	When triggered, it plays a brief muzzle flash effect.
play_brfield	When triggered, toggles a spherical yellow particle effect.
play_gibs	When triggered, it plays gib effects and sound. Same as *meat_shower* from earlier versions. See an example of *meat_shower* used with a *func_counter* in pd_meat.map When triggered this entity will spawn a shower of gibs. *style* = 0 is regular gib effect, 1 is more violent *fly_sound* = 0 is silent, 1 plays randomized gib sounds *targetname* = Must be triggered
play_tele	When triggered, this shows the teleport particle effects and sound. Same as *tele_fog* from earlier versions. Use this when killtargeting an entity if the player can see it happen. You can see an example on the Shambler near the *trigger_use* key entity in *The Gallery* map.

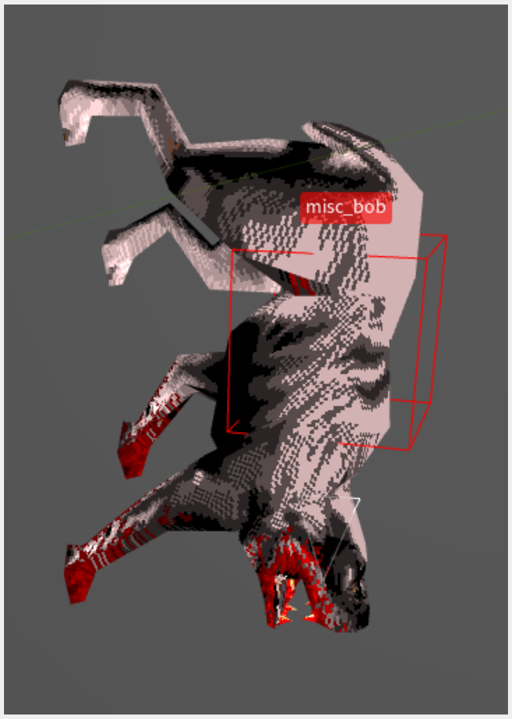
tele_fog, play_tbabyexplode **and** ***meat_shower*** **have been deprecated and renamed. The QuakeC code is still present for backward compatibility but the entities have been removed from the FGD.**

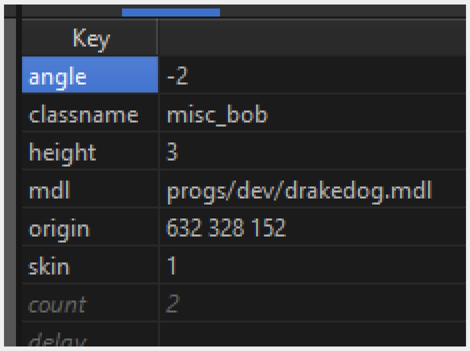
func_bob

This will create a brush that gently moves back and forth or up and down depending on the angle. Use *targetname* to trigger it on, *angle* is direction movement, use "360" for angle 0 *height* direction intensity (def=8) *count* = direction cycle timer (def=2s, minimum=1s) *waitmin* = Speed up scale (def=1) 1+=non linear, *waitmin2* = Slow down scale (def=0.75) *delay* = Starting time delay (def=0, -1=random) *style* If set to 1, starts off and waits for trigger *_dirt* -1 = will be excluded from dirtmapping, *_minlight* = Minimum light level for any surface of the brush model, *_mincolor* = Minimum light color for any surface (def='1 1 1' RGB) *_shadow* = Will cast shadows on other models and itself, *_shadowself* = Will cast shadows on itself. Use the BOB_COLLISION spawnflag for solid and conversely, BOB_NONSOLID.

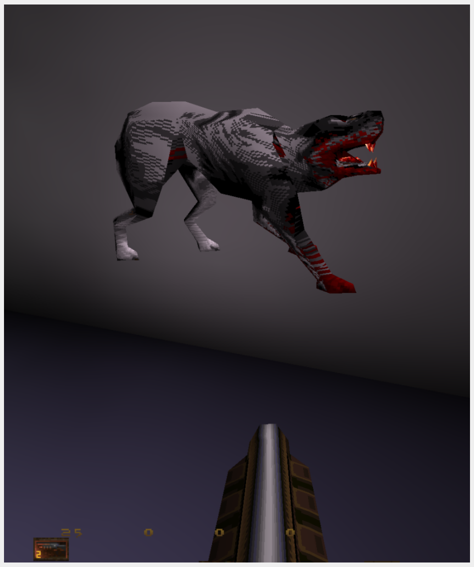
misc_bob

Same as above but uses a custom model instead of a brush. Use the *mdl* key to set the path of the model. There's a quirk in TrenchBroom that will not display the model in its proper orientation once you set the angle key as seen here. This is only in the editor, in-game the model will be correct.





Key	
angle	-2
classname	misc_bob
height	3
mdl	progs/dev/drakedog.mdl
origin	632 328 152
skin	1
count	2
delay	



It's best to NOT use *func_bob* and *misc_bob* for jumping puzzles, as they can get "get out of sync" and not allow the player to progress in certain instances.

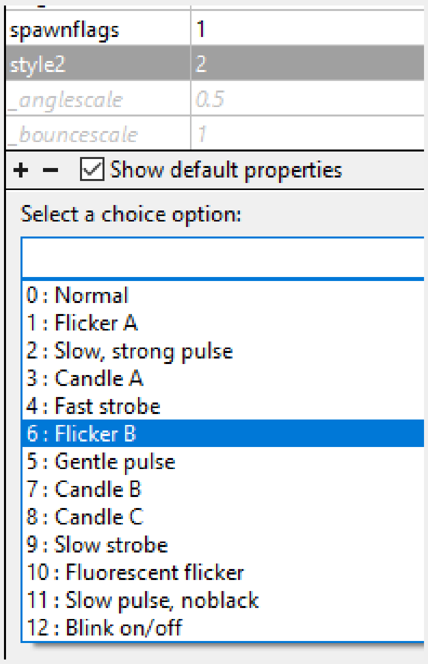
Light and Shadow Features

Switchable Light Styles

Normally, if you apply a *style* to a light (e.g. candle flicker, strobe) those cannot be triggered on and off. However, progs_dump has this ability, borrowed from c0burn's in-progress *Slipgate* mod. Just choose a *style2* selection from the dropdown and target the light as normal. Use the *START OFF* spawnflag if needed.

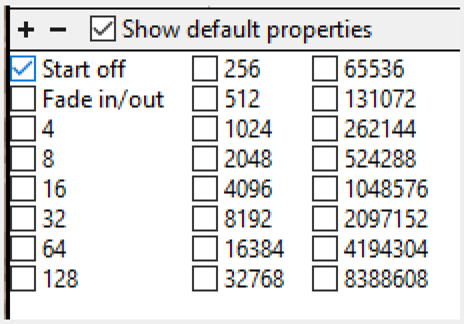
Select the *FADE IN / OUT* spawnflag for a smooth fade in / out effect on non-animated lights. The *speed* key controls the light transition time. Default 0.1

Fades will not work on animated lights (e.g. style or style2).



spawnflags	1
style2	2
_anglescale	0.5
_bouncescale	1

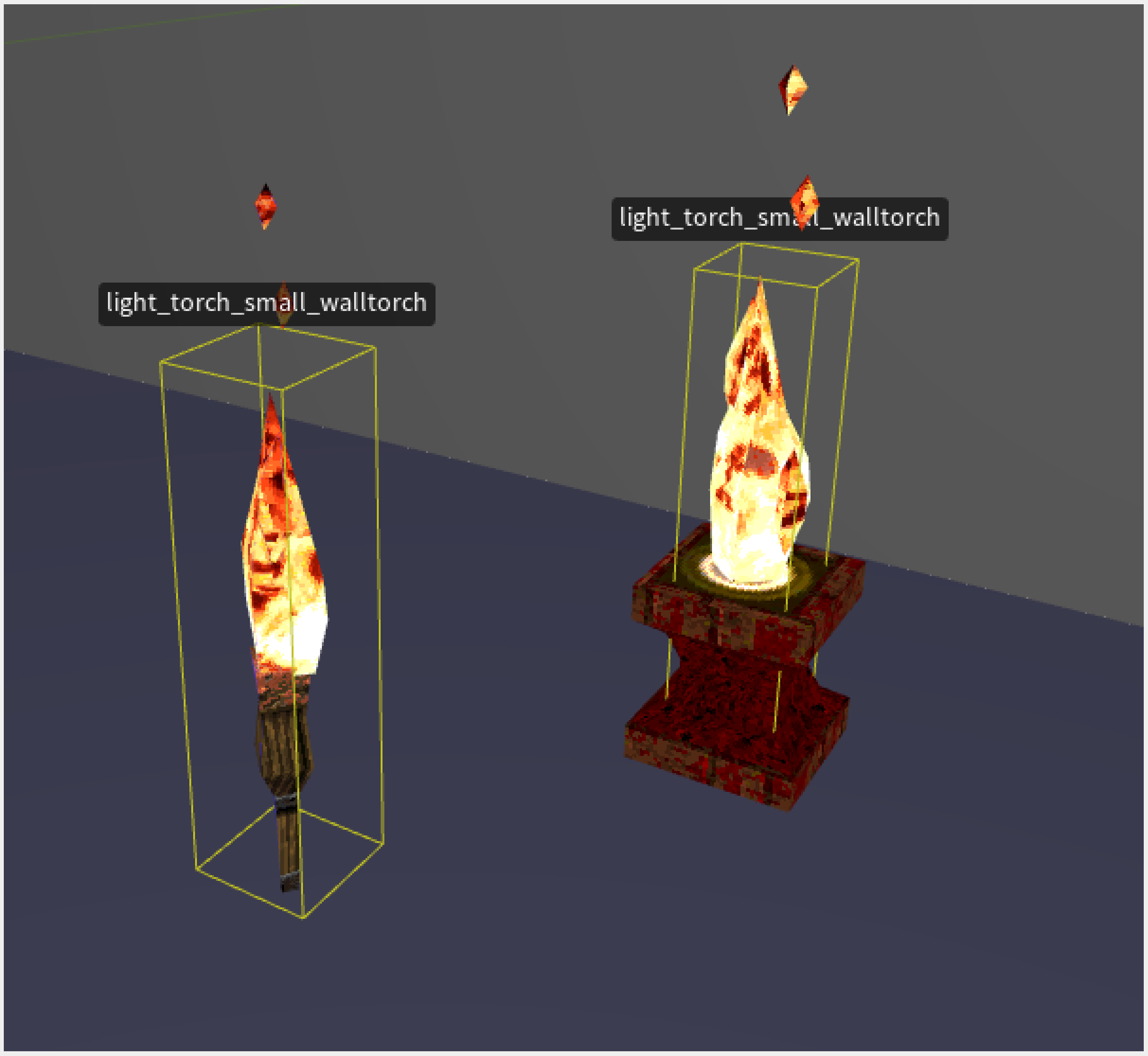
0 : Normal
1 : Flicker A
2 : Slow, strong pulse
3 : Candle A
4 : Fast strobe
6 : Flicker B
5 : Gentle pulse
7 : Candle B
8 : Candle C
9 : Slow strobe
10 : Fluorescent flicker
11 : Slow pulse, noblack
12 : Blink on/off



✓ Start off	☐ 256	☐ 65536
☐ Fade in/out	☐ 512	☐ 131072
☐ 4	☐ 1024	☐ 262144
☐ 8	☐ 2048	☐ 524288
☐ 16	☐ 4096	☐ 1048576
☐ 32	☐ 8192	☐ 2097152
☐ 64	☐ 16384	☐ 4194304
☐ 128	☐ 32768	☐ 8388608

light_torch_small_walltorch

Just like monsters and items, you can replace the model on this entity to make it easier to mix and match different light sources. Use the *mdl_body* and *skins* keys as you would with custom monsters and items. Use the *silent* spawnflag to disable the crackling fire sound if you want a custom sound. In this case, you'll want to use an ambient_general entity near the lightsource with a looping sound file.



Switchable Shadows

Thanks to bmFbr, *progs_dump* now has the switchable shadow system from *Alkaline*. Every brush entity, except triggers and *func_detail* variants, can have toggleable shadows by adding a *_switchableshadow 1* key to it. *func_door*, *func_breakable* and *toggle_visiblewall* already have built-in support for that, so all you need to do is select the key for it to work automatically.

For other entity classes, like *func_shadow*, you need to set up a *misc_shadowcontroller* point entity to control their switchable shadows. Here’s how:

- Create a _*switchableshadow* key in the desired entity and set it to 1;
- Give it a *targetname* - if the entity changes behavior when given a *targetname*, you can target a *targetname2* field instead.
- Create a *misc_shadowcontroller* entity targeting the bmodel, and give it a different *targetname*.
- Triggering the shadow controller toggles the shadow on/off with a fading animation.

Currently, ericw's LIGHT tool supports switchable shadows only when the map isn't using bounce lighting. That's been already corrected in development versions however, and should be out in a future release. (as of September 2022).

func_shadow

An invisible, non-solid brush entity that can be used to cast shadows In order for it to work correctly you need to either set a _*shadow 1* or a _*switchableshadow 1* key. See the *prefab_shadows* sample map for examples.

misc_shadowcontroller

Controls switchable shadows on any bmodel entity (except doors and breakables). Target this entity to toggle the shadows on/off with an optional fading animation.Targeted bmodel must have _*switchableshadow* set to 1. If the target entity changes its default behavior when 'targetname' is set (such as breakables), you can target the entity's 'targetname2' key. See the *prefab_shadows* sample map for examples.

Particle Effects

misc_sparks

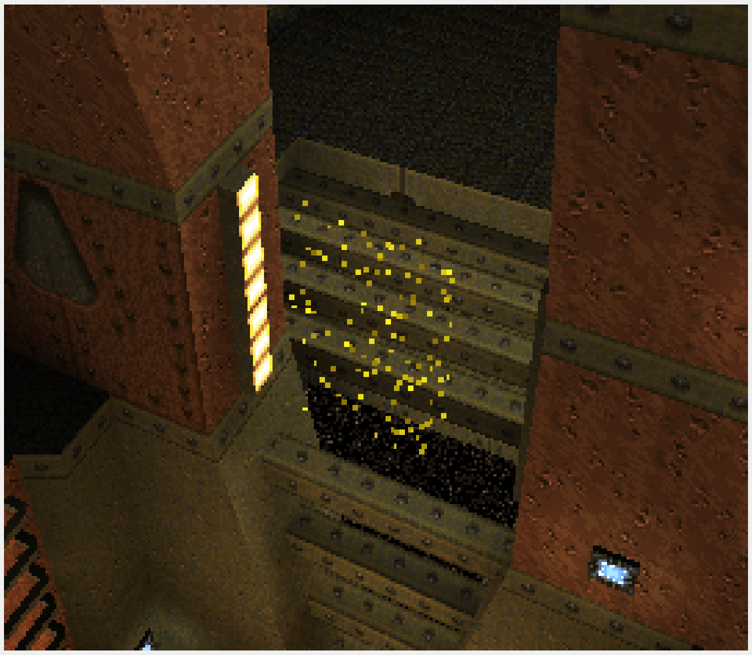
Produces a burst of yellow sparks at random intervals. If targeted, it will toggle between on or off. If it targets a light, that light will flash along with each burst of sparks. **NOTE:** targeted lights should be set to START_OFF. Spawnflags = SPARKS_BLUE: sparks are blue in color SPARKS_PALE sparks are pale yellow in color. *wait* is the average delay between bursts (variance is 1/2 *wait*). Default is 2. *cnt* is the average number of sparks in a burst (variance is 1/4 *cnt*). Default is 15. sounds 0 = no sound, 1 = sparks. *noise* is the path to a custom spark sound effect, *sounds* must be set to 1.

misc_particle_stream

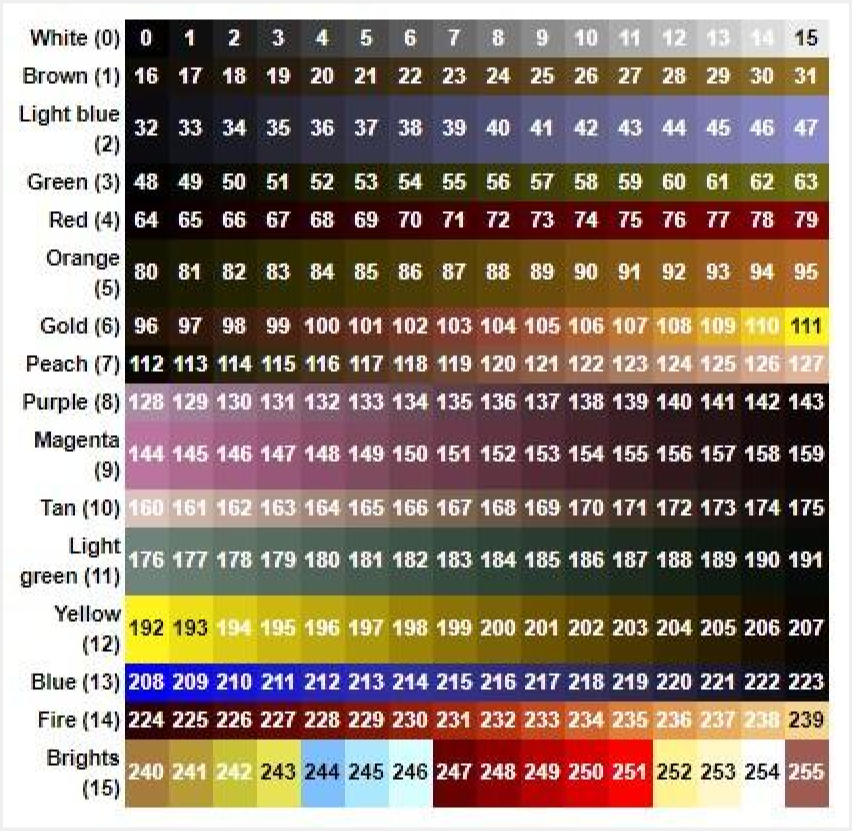
A particle stream! It appears when triggered. This entity is one end of the stream, target another entity as the other end-point. Usually an *info_notnull*, but you should be able to target anything (like monsters). *target* = This entity's origin is the end-point of the stream *dmg* = 1st Color, use this by itself if you want a single color stream *cnt* = 2nd Color, mixes particles of both colors. noise = Sound to play when triggered. See color palette reference below. **NOTE**: You can see this in action in the *pd_counter* sample map and at the end of the *pd_lasers* map.

func_particlefield

Creates a brief particle flash roughly the size of the defining brush each time it is triggered. You can see an example of this in the *pd_ladders* example map. In this case, the particle fields are animated in sequence to create a force field effect. *USE_COUNT* when the activator is a func_counter, the field will only activate when count is equal to cnt. Same as using a func_oncount to trigger. *cnt* is the count to activate on when *USE_COUNT* is set. *color* is the color of the particles. Default is 192 (yellow). *count* is the density of the particles. Default is 2. *noise* is the sound to play when triggered. <u>Do not use a looping sound here.</u> *dmg* is the amount of damage to cause when touched.



If you want to use another color for the particle field, refer to the Quake color palette below **NOTE**: not all colors will work:



White (0)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Brown (1)	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Light blue (2)	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Green (3)	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Red (4)	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Orange (5)	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Gold (6)	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Peach (7)	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
Purple (8)	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
Magenta (9)	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
Tan (10)	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
Light green (11)	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
Yellow (12)	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
Blue (13)	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
Fire (14)	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
Brights (15)	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

misc_particles

Produces a continuous particle splash for waterfalls and other effects. Can be triggered and toggled. Spawnflags = START_OFF The default behavior has the particles shimmering in an upward motion. *color* = color of particles. 0 through 15, corresponds to a row of the quake palette (see above for palette numbers). (default 0) *movedir* = average movement vector of particles (default 0 0 4) **NOTE**: Play with negative numbers to change the movement direction. *wait* = time between particle generation cycles. (default 0.1) volume = density of particles. (default 10)

misc_particlespray

Shoots particles either when triggered, or continuously when not triggered by anything. *color* is the palette color of the particles. (default 47) *movedir* is the vector distance that the particles will travel before disappearing. (in x y z) **NOTE**: Play with negative numbers to change the movement direction. *delay* is the delay between each triggering (default 0.1) *duration* is the amount of time that it will continue to release particles so that it can release a long stream of particles with only one triggering, *count* is the number of particles to make each time (default 15) *noise* is the name of the .wav file to play when triggered.

The two particle effects above are similar but there are some key differences. Take a look at the *pd_counter* map for some different examples.

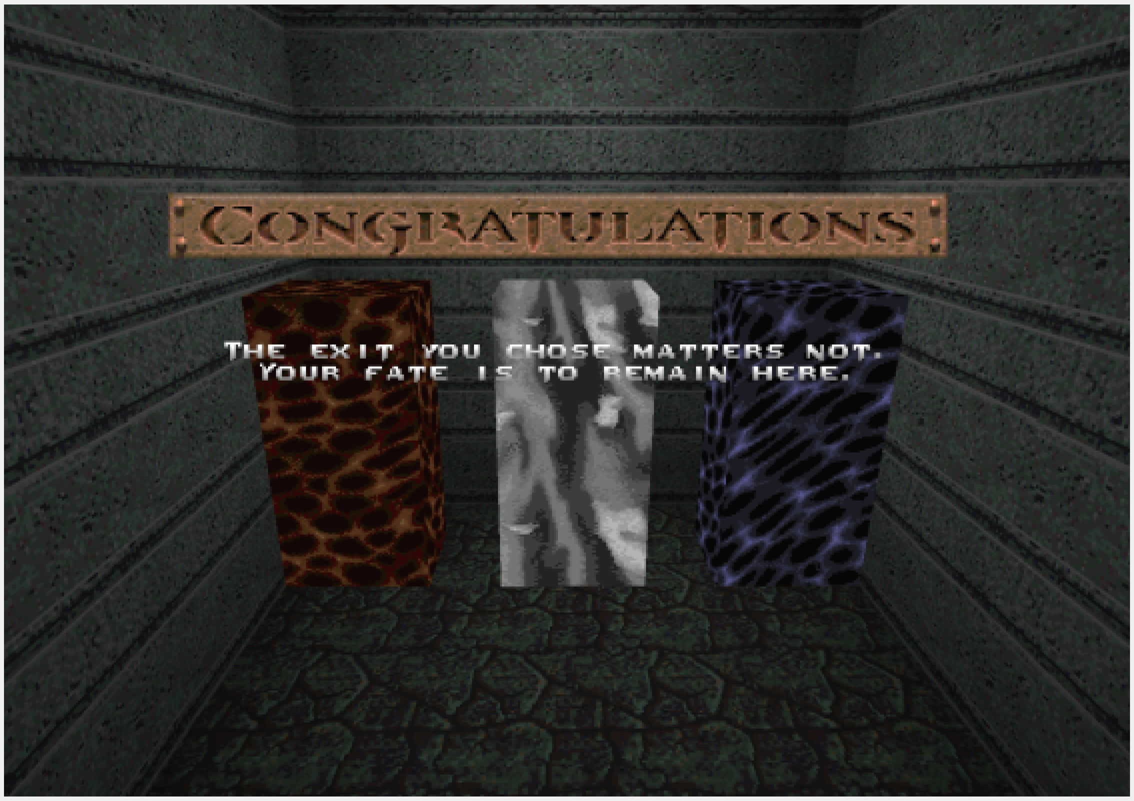
You can check out this video tutorial that goes into detail on most of the particle effects above.

Enhanced Text Tools

We've added enhanced text entities and other features, starting in progs_dump 3.0.0, that allow for many more storytelling opportunities. These include additions to *trigger_ changelevel* *info_intermissiontext* and the *textstory* entities.

info_intermissiontext

Created by ILike80sRock, this is used with a *trigger_changelevel* entity to display the "Congratulations" graphics and a custom message. These are displayed in the original game at the end of an episode when the player gets a rune, kills a boss or defeats the game. This entity can display multiple "pages" of text before the player warps to the next level. More information on how to use this functionality is described in the *trigger_changelevel* section above.



trigger_textstory

This new trigger from bmFbr shows long centerprint texts. Messages remain on screen while the player is inside the trigger volume. Can have custom sounds, or be made silent. Horizontal space is limited to 40 characters, so you must place linefeeds (\n) into your text. Use *fade_amt* to set the background fade to a custom value (default is 160). You can set it to show the message only when the player is facing a certain angle within the trigger. Set the central angle with *mangle*, and the maximum angle offset with *view_ofs* (both in 'pitch yaw 0' format). See illustration below.

Key	Details
fade_amt	Set a custom black level for background fade (default is 160)
mangle	Central point of the viewcone, in 'pitch yaw 0'

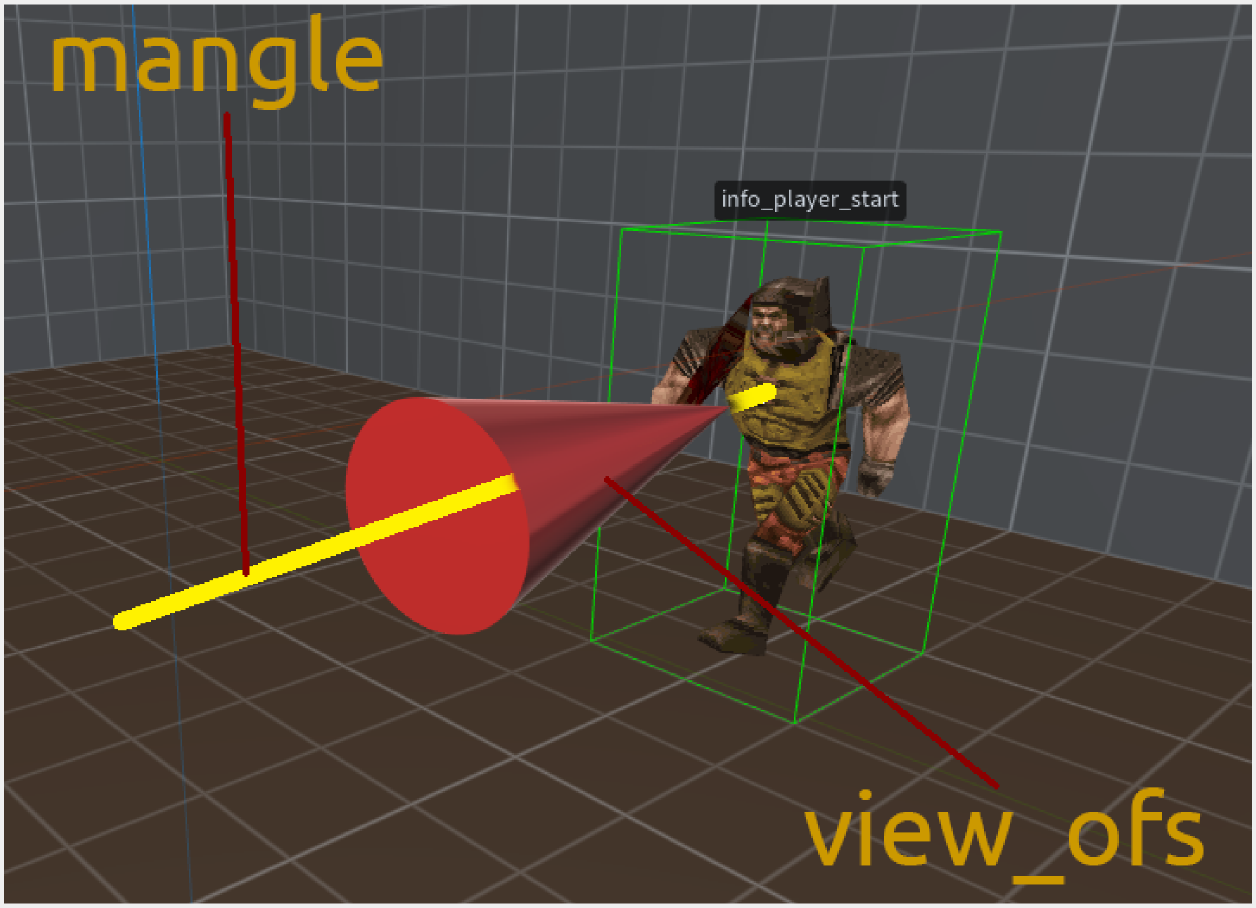
	format
message	Text of your message. 40 character limit then add \n for new line
noise1	Path to custom activation sound. Played when the message is shown. Uses 'misc/talk.wav' for the default centerprint beep.
noise2	Path to custom deactivations sound. Played when the message fades away.
view_ofs	Maximum angle offset. Angle offsets from the central viewcone point in each direction, in 'pitch yaw 0' format
Spawnflag	**Details**
Silent	No message sounds
No background fade	Background will not fade to black

target_textstory

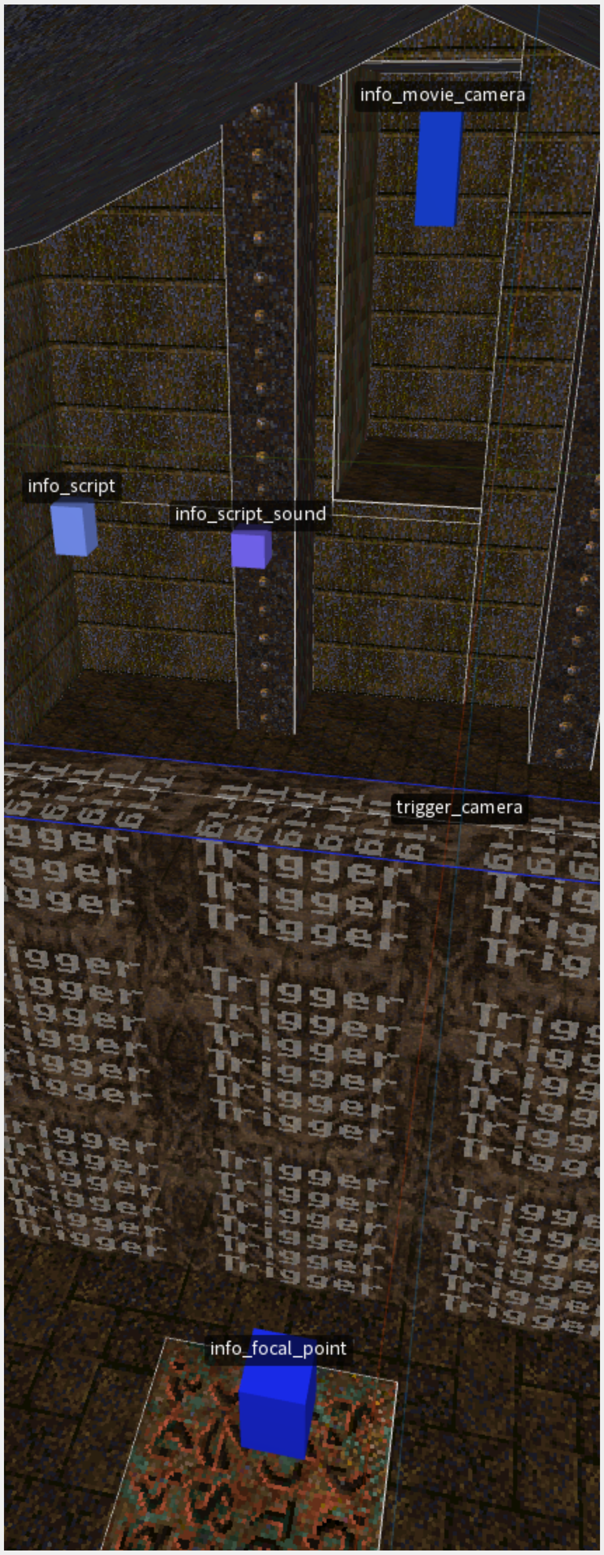
Point entity version of *trigger_textstory* that shows long centerprint texts. Message remains on screen for the duration of *wait*. Can have custom sounds, or be made silent. Horizontal space is limited to 40 characters, so you must place linefeeds (\n) into your text. Use *fade_amt* to set the background fade to a custom value (default is 160).

Key	**Details**
fade_amt	Set a custom black level for background fade (default is 160)
message	Text of your message. 40 character limit then add \n for new line
noise1	Path to custom activation sound. Played when the message is shown. Uses 'misc/talk.wav' for the default centerprint beep.
noise2	Path to custom deactivations sound. Played when the message fades away.
wait	Amount of time to display the message.
Spawnflag	**Details**
Silent	No message sounds
No background fade	Background will not fade to black

The yellow line in this image represents the *mangle* and the cone represents the *view_ofs.* I'm using a player to illustrate this but it's the same concept using the *trigger_textstory*.



Cutscenes



The cutscene system is taken from the Drake mod beta devkit. Scenes take a bit of testing and tweaking to set up, so please read this section *carefully* if you want to include them in your projects. One missing key | value or typo will blow up the whole operation! **It's best to start with a small test level and learn how they work before moving forward.** Also play very close attention to the "best practices" section below!

NOTE: unlike many Quake entities, there are key | value pairs that are required to be set even though they may not seem to do anything.

Your first step should be to play the sample map *pd_cutscenes*, then open the map in your map editor and take a look at the different setups. There's a secret area of the map that shows the most simple setup, with one message and one camera. There are also three other, more complex setups in the map. Cutscenes can be skipped by pressing a weapon key or any impulse command.

There are a minimum of four entities required to make a cutscene work.

First, a *trigger_camera* that the player will enter to begin the scene. You can also use a *trigger_camera_point* if you need to trigger your scene without the player touching the trigger. You will see both methods in the sample map.

The second required entity is an *info_movie_camera*. This will "aim" the camera at the third required entity: an *info_focal_point*.

The fourth required entity is an *info_script*. This controls how long the player is in the scene, triggers events and holds any text messages that will play during the cutscene.

trigger_camera

This will begin a cutscene when touched. Some of these keys need to match the corresponding fields in the targeted *info_movie_camera* and *info_script*. **IMPORTANT**: most of the following keys are required unless noted.

Key	**Details**
focal_point	Point the targeted camera at this point.
script	Match script_num field of the info_script
script_delay	The amount of time to stay on the first script page. **NOTE**: You can usually set this to 1 because the *script_delay* key of the matching *info_script* will override this value.
target	Targetname of the first camera in the cutscene.
targetname (optional)	If the *trigger_camera* has a *targetname*, it will be dormant until triggered.

info_movie_camera

This is the target of the *trigger_camera* and controls the viewport of the cutscene. When using multiple cameras in a sequence, you need at least three cameras (see “complex cutscenes” below).

Key	**Details**
focal_point	Point the camera at this point.
targetname	The name of this camera.
delay (optional)	When the camera moves, don't track the focal_point’s position, keep the initial view angle.
speed (optional)	This controls the rate of travel to this camera in (Quake units per second) from another camera.
wait (optional)	Wait here in seconds, before moving to next camera if part of a sequence
target (optional)	*targetname* of the next *info_movie_camera* in a sequence.

info_focal_point

This is the point that the camera will face. It should have a *targetname* value matching the *camera_trigger* and *info_movie_camera's focal_point* fields. When using multiple cameras the focal points can change (see "complex cutscenes" below).

info_script

This controls the on-screen timing and the optional message text fields.

Key	Details
script_num	This should match the *script* field of the *trigger_camera* or *trigger_camera_point*. **IMPORTANT: Every *script_num* in a map needs to be unique!**
next_script	This is the *script_num* field of the next *info_script*, if part of a sequence. Set to zero if this is the last script in a series. **IMPORTANT: This value must be set by hand even if the number is zero!** This is unlike almost every other entity field in Quake mapping! Your cutscene will fail without this set.
script_delay	How many seconds to stay on this script. This overrides the same key on *trigger_camera*.
message (optional)	Optional text that will stay on screen for the amount of time set in script_delay. It's safest to limit this to a max of 64 characters.
target1-4 (optional)	Use these fields to trigger other events in time with the current script. Use *trigger_relays* if you need to *killtarget* something.

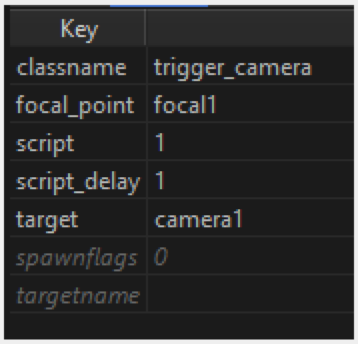
info_script_sound

You can use this optional entity to add a sound when text is displayed or you can even trigger custom sounds and add dialogue to your scenes!

Key	Details
sounds	Default Quake sounds for messages. Select 4 if you want to use a custom sound file.
noise1	Path to custom sound file. Requires sounds key set to 4.
targetname	Name of entity. You can use this multiple times in the same level but note the sound is directional.

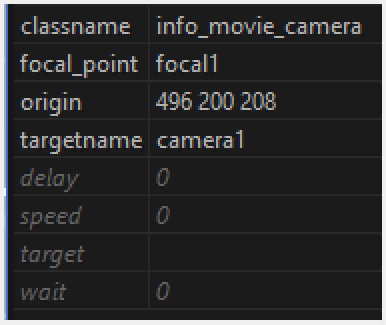
Creating a Simple Cutscene

Create a *trigger_camera* brush and give it these key | values:



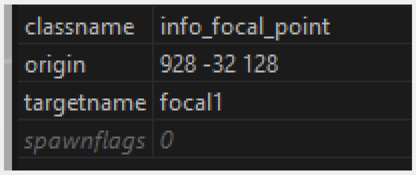
Key	
classname	trigger_camera
focal_point	focal1
script	1
script_delay	1
target	camera1
spawnflags	*0*
targetname	

Next add an *info_movie_camera* and give it these key | values:



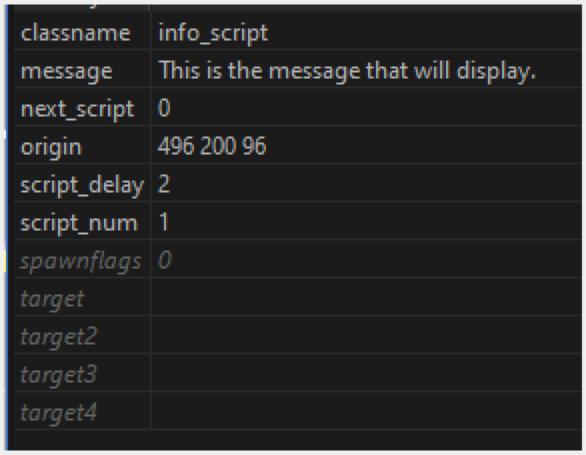
classname	info_movie_camera
focal_point	focal1
origin	496 200 208
targetname	camera1
delay	*0*
speed	*0*
target	
wait	*0*

Notice they both have the same *focal_point* key. Now create that *info_focal_point* give it these key | values:



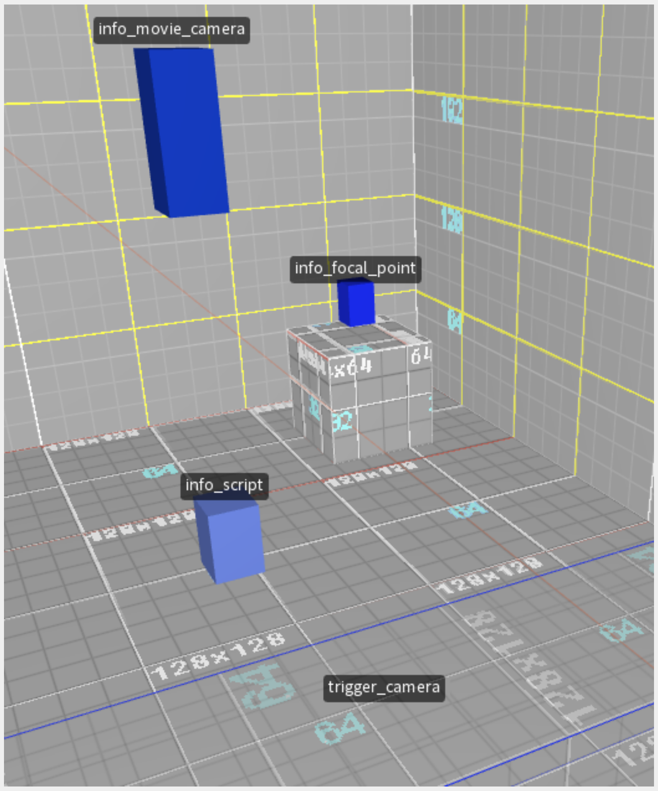
classname	info_focal_point
origin	928 -32 128
targetname	focal1
spawnflags	*0*

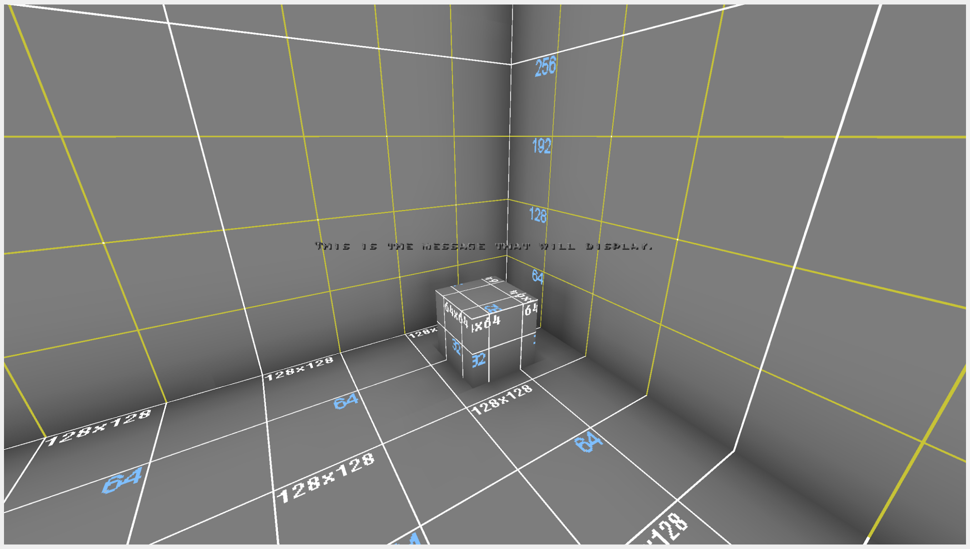
Now for the *info_script*. Note the *script_num* matches the *script* key from *trigger_camera*. The *next_script* value is <u>set by hand to zero</u>, this is **really** important to add or your cutscene will break! It's set to zero, because it's the last script of the scene.



classname	info_script
message	This is the message that will display.
next_script	0
origin	496 200 96
script_delay	2
script_num	1
spawnflags	*0*
target	
target2	
target3	
target4	

The length of the scene is controlled by the *script_delay* key in the *info_script*. Leave the message key blank if you just want a shot without text. Now you can move the focal point and camera around for your desired “angle”. The following screenshots show in-editor and then the in-game vantage points.





This is the message that will display.

Complex Cutscenes

You can add more cameras, scripts and focal points for a more complex scene. You can also animate the camera from one point to another but getting good looking “shots” takes a bit of time and tinkering.

I’ve created smaller demo levels for easy reference in addition to the *pd_cutcenes* map. These maps are not accessible from the *progs_dump* start map but you can load them via the console.

Here’s what each map demonstrates:

Map Name	**Details**
pd_cutscn_simple	A static camera, one message scene.
pd_cutscn_tracking	Animated camera between two points with two messages and a blank script between them for timing. Notice how the *delay* key is set on all the cameras and only one focal point is needed. This makes each camera focus in the same direction for each move.
pd_cutscn_cuts	Scene that “cuts” between three vantage points, with three separate focal points. Notice how the *speed* key is set to 999999 to make the move nearly instantaneous. You may still see a “flash frame” between the edits here. There’s no real way around this.

When moving the camera, even between just two points, you will need three info_movie_cameras. (This is due to some quirks in the original QuakeC and took me a long time to figure out!) If you only want two vantage points in your scene, simply use the *wait* key on the second to last camera. Set this to a longer amount of time than is controlled by the *script_delay* key in your *info_script* for that section of the cutscene. You can see this clearly in *pd_cutscn_tracking* and in *pd_cutscenes*.

Cutscene Best Practices

● Make small test maps to set up your cutscenes. Scenes require a lot of testing and tweaking. When they are working, paste them into your map and adjust as needed.

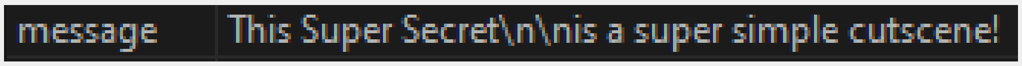
● **Do not quit the game while in a cutscene, this will reset your mouse sensitivity and console viewsize.** Add this info in the readme for your mod so players know not to quit!

● Keep your cutscenes as simple as possible. Things can break very quickly as you ramp up the complexity.

● Timing is controlled in two places when using multiple cameras. (*script_delay* in *info_script* and *wait* in *info_movie_camera*) That makes it harder to make small changes. Break up longer sequences into smaller parts.

● Camera moves in X and Y will display a bit of “player bob”. Play with the speed key to make the move faster and it won’t be as apparent.

● As in other Quake entities, you can add a line break in a message by adding \n with no space before the text of the second line. Here I’ve added two to make a blank line between sentences.



```
message    This Super Secret\n\nis a super simple cutscene!
```

● Do not reuse *info_scripts* for different cutscenes. Things will break. You *can* reuse *info_movie_cameras* as long as they are triggered by different *trigger_camera* entities.

● As mentioned above, *info_script_sound* entities can be reused. Be aware that because of the way Quake handles audio, the sound direction can change depending on where the entity is in relation to the camera. If you have sound coming from only one direction, center the *info_script_sound* on the focal point of the camera.

● Don’t overdo it. Quake is a fast paced game. Few players want to watch a three hour Quake movie with terrible voice acting!

Rotation Entities

By request, the Hipnotic (Quake Mission Pack 1) rotation entities have been added but are **unsupported**. Use at your own risk! The text below is taken from the Quake C code for the rotation system. Refer to the included sample map *pd_rotate* for examples. Also, here are the steps to create rotating entities. Enjoy!

func_rotate_entity

Creates an entity that continually rotates. Can be toggled on and off if targeted. TOGGLE = allows the rotation to be toggled on/off START_ON = whether the entity is spinning when spawned. If TOGGLE is 0, the entity can be turned on, but not off.

If "deathtype" is set with a string, this is the message that will appear when a player is killed by the train. The mapper can set "deathtype" on either an "inflictor" entity or an "attacker" entity (in the mission pack, it only worked when set on an attacker). The inflictor and attacker are often one and the same entity, but there are some cases where they are not: for example, if a player gets crushed by a func_movewall which is controlled by a func_rotate_entity, the func_movewall is the inflictor and the func_rotate_entity is the attacker.

"rotate" is the rate to rotate. "target" is the center of rotation, "speed" is how long the entity takes to go from standing still to full speed and vice-versa.

path_rotate

(Train with rotation functionality) Path for rotate_train. ROTATION tells the train to rotate at a rate specified by "rotate". Use '0 0 0' to stop rotation. ANGLES tells the train to rotate to the angles specified by "angles" while traveling to this path_rotate. Use values < 0 or > 360 to guarantee that it turns in a certain direction. Having this flag set automatically clears any rotation. STOP tells the train to stop and wait to be retriggered. NO_ROTATE tells the train to stop rotating when waiting to be triggered. DAMAGE tells the train to cause damage based on "dmg". MOVETIME tells the train to interpret "speed" as the length of time to take moving from one corner to another. SET_DAMAGE tells the train to set all targets damage to "dmg" "noise" contains the name of the sound to play when the train stops. "noise1" contains the name of the sound to play when the train moves. "event" is a target to trigger when the train arrives at path_rotate.

func_rotate_train

In path_rotate, set speed to be the new speed of the train after it reaches the path change. If speed is -1, the train will warp directly to the next path change after the specified wait time. If MOVETIME is set on the path_rotate, the train interprets "speed" as the length of time to take moving from one corner to another. "noise" contains the name of the sound to play when the train stops. "noise1" contains the name of the sound to play when the train moves. Both "noise" and "noise1" defaults depend upon the "sounds" variable and can be overridden by the "noise" and "noise1" variables in path_rotate.

Also in path_rotate, if STOP is set, the train will wait until it is retriggered before moving on to the next goal.

Trains are moving platforms that players can ride. "path" specifies the first path_rotate and is the starting position. If the train is the target of a button or trigger, it will not begin moving until activated. The func_rotate_train entity is the center of rotation of all objects targeted by it.

If "deathtype" is set with a string, this is the message that will appear when a player is killed by the train. *speed* (default 100) *dmg* (default 0) *sounds* 1 = ratchet metal

func_movewall

Used to emulate collision on rotating objects. VISIBLE causes brush to be displayed. TOUCH specifies whether to cause damage when touched by a player. NONBLOCKING makes the brush non-solid. This is useless if VISIBLE is set. "dmg" specifies the damage to cause when touched or blocked.

rotate_object

This defines an object to be rotated. Used as the target of func_rotate_door.

func_rotate_door

Creates a door that rotates between two positions around a point of rotation each time it's triggered. STAYOPEN tells the door to reopen after closing. This prevents a trigger-once door from closing again when it's blocked. "dmg" specifies the damage to cause when blocked. Defaults to 2. Negative numbers indicate no damage. "speed" specifies how the time it takes to rotate "sounds" 1 = medieval (default), 2 = metal, 3 = base 4 = silent

Sample maps

You can find these in the maps folder along with a wad file containing all the textures used in the development folder. You are welcome to copy and paste the entity setups and adjust as needed to use them in your maps. Please do not copy brushes or geometry from these maps. The following chart shows what examples exist in each map. Not all entities are demonstrated in the sample maps.

Maps prepended with *pd_* are playable maps. Prefab maps are more simple in presentation, but are specifically designed for you to copy and paste more complex setups.

Map	Entity Setup Examples	Notes
start	n/a	Modified version of *Celeritate satus* **by Danz**
pd_bosses	(killable bosses) monster_oldone2, monster_boss2, item_backpack, func_fall2, invisible func_breakable, play_mflash, misc_particlestream, play_sound_triggered, ambient_fire	
pd_breakables	func_breakable, misc_candle	
pd_change_dest	info_teleport_changedest, info_teleport_random, play_tele, trigger_shake, misc_sparks, play_sound_triggered, misc_teleporttrain	
pd_counter	func_counter, func_oncount, misc_particle, misc_particle_stream, misc_particlespray	
pd_cutscenes	trigger_look, info_movie_camera, info_focal_point, info_script, info_script_sound, trigger_changemusic, trigger_cdtrack	
pd_cutscn_cuts	sames as above	Shows how to make an “edited” sequence from multiple camera angles,
pd_cutscn_simple	sames as above	Shows how to make a simple “cutaway” shot.

pd_cutscn_tracking	sames as above	Shows how to make a shot that "tracks" from one position to another in a single shot.
pd_elevator	func_new_plat, func_elvtr_button	
pd_gallery		all entities from version 1.0.0
pd_gravity	trigger_setgravity, trigger spawned monsters	from 1.0.0
pd_ionous	is_waiting, trigger spawned monsters	**Map by Ionous** voice.of.the.nephilim@gmail.com @voiceovnephilim
pd_keys	item_key_custom	**Map by iw**
pd_ladders	trigger_ladder, func_particlefield, misc_sparks, func_breaklable, func_togglewall	
pd_lasers	func_laser, ltrail_start,ltrail_relay, ltrail_end, switchable light styles, misc_particle_stream, trigger spawned monsters	Can you find the YA secret?
pd_lava	play_lavasplash, func_fall, trigger_shake, misc_particle, func_train (triggered)	
pd_lightning	func_counter, ltrail_start,ltrail_relay, ltrail_end, trap_switched_shooter	
pd_meat	meat_shower, func_counter, gib_*, monster_dead_*	
pd_rotate	func_rotate_entity, path_rotate, func_rotate_train, func_movewall, rotate_object, func_rotate_door	This is Hipnotic's original sample map. Slight changes to lighting and renamed.
pd_turrets	various	Demonstrates different monster turret types.
pd_void	func_bob, suspended items, trigger spawn items, func_breakables	
pd_yoder	trigger_push_custom, multiple trigger names,	**Map by by Andrew Yoder**

	trigger_setgravity	AndrewYoder@live.com @Mclogenog
pd_zombies	func_counter, func_oncount, enhanced zombies, trigger spawned monsters	
prefab_ammo_backpack	item_backpack	This shows how you can use an item_backpack to mimic ammo boxes. You can set custom ammo, re-name the ammo. This offers more options than standard ammo.
prefab_assets	various	Shows most new models and a partial selection of built-in skins.
prefab_estate	various	Shows various examples of the estate (entity state) system.
prefab_fog	trigger_fogblend, target_fogblend	Many different fog setups. (Also the ugliest map in progs_dump!)
prefab_func_fall2	func_fall2	Shows most common setups with variations.
prefab_homing	homing and waitmin key | values	Shows off different homing settings.
prefab_infighting	misc_infight	Simple example of triggering a fight between monsters.
prefab_infointermissiontext	info_intermissiontext	Demonstrates how to display custom intermission messages. These are usually seen at the end of an episode.
prefab_monsterface	trigger_monsterface	Shows how this entity can change a monster’s path when the player is hidden from view.
prefab_shadows	_switchableshadow key	Different uses of this key | value along with misc_shadowcontroller.
prefab_spawner	func_monster_spawner, info_monster_spawnpoint	Two different ways to use monster spawners.
prefab_styles		See all monster styles and variants that are easy to configure. Check out this video showcase for more.
prefab_textstory	trigger_textstory, target_textstory	Simple examples of these entities and variations of fade_amt.
prefab_trains	misc_modeltrain	Powerful new train entities that can be triggered in different ways including the ability to animate monster models.
prefab_trig_filter	trigger_filter, trigger_everything, trigger_changetarget	An example of how to filter out projectiles to trigger a switch with a nail gun only.

Credits

QuakeC Sources

misc_model.qc, math.qc by Joshua Skelton
https://gist.github.com/joshuaskelly/15fe10fbaaa1bf87b341cba6e3ad2ebc

Trigger Spawned Monsters added via Preach’s excellent tutorial:
https://tomeofpreach.wordpress.com/2017/10/08/teleporting-monsters-flag/

various .qc from custents by Carl Glave
http://www.quaketastic.com/files/tools/windows/quakec/custents.zip

various .qc from Hipnotic’s Quake Mission Pack Scourge of Armagon
Original Code written by Jim Dose and Mark Dochtermann
http://www.quaketastic.com/files/tools/windows/quakec/soa_all.zip

various .qc from Rogue’s Quake Mission Pack Dissolution of Eternity
Original Code written by Peter Mack et al.
http://www.quaketastic.com/files/tools/windows/quakec/doe_qc.zip

Preach’s clean Quake 1.06 source courtesy of Joel B
https://github.com/neogeographica/quakec/tree/1.06_Preach

various .qc from Rubicon Rumble Pack Devkit by ijed / Louis
http://www.quaketastic.com/files/single_player/mods/RRP_DEVKIT.zip

Arcane Dimensions breakable and music code by Simon O’Callaghan et al.
http://www.simonoc.com/pages/design/sp/ad.htm

Honey source by czg
https://www.quaddicted.com/reviews/honey.html

Zerstörer QuakeC Development Kit - Dave 'Ace_Dave' Weiden and Darin McNeil
https://www.quaddicted.com/reviews/zer.html

various .qc code from Rubicon 2 copyright 2011 John Fitzgibbons.
https://www.quaddicted.com/reviews/rubicon2.html

deadstuff version 1.0 - Tony Collen
ftp://archives.gamers.org/pub/idgames2/quakec/level_enhancements/deadstuf.zip

Remake Quake code by Supa, ijed, et al.
https://icculus.org/projects/remakequake/

switchable lightstyles and DEF entries from Slipgate by Michael Coburn
https://github.com/c0burn/Slipgate

cutscenes and various .qc from Drakebeta by Patrick Martin
http://www.quaketastic.com/files/single_player/mods/drakebeta.zip

trigger_look by NullPointPalidin
https://nullpointpaladin.wordpress.com/

Copper style noclip from Copper by Lunaran
http://lunaran.com/copper/modding/

Nailgun origin fix from Seven and Sajt, courtesy of Greenwood.
http://shub-hub.com/files/mods_singleplayer/NailgunNailPosition.zip

Numerous Additions by bmFbr from the Alkaline mod
http://quake.great-site.net/files-n-links/

Lava Ogre QC and skin courtesy of JaycieErysdren
https://jaycie.erysdren.me

Additional code assistance and examples from iw, NullPointPaladin Qmaster, RennyC, Khreathor, Spike, ILike80sRock, Whirledtsar and c0burn.

A note about key | value pairs. You may notice some strange naming of key fields in progs_dump. For example, the *currentammo* in a lightning trail relay represents damage. This is because some of the code in progs_dump is from a time when the memory requirements for Quake were high for PCs of the day. Programmers would reuse certain keys to save memory, as each key field took up precious RAM. 25 years later, we have plenty of processing and RAM to dispose of. As a result, some of the newer keys are a bit more intuitive.

progs_dump Q logo by D.E.F.A.M.E. https://defameart.com/

Appendices

Appendix A: Included Assets

Here's a list of assets that are included in the mod with credits. You can also see a preview of the many new skins that are available below.

Models

Asset Name	Details	Credits
pd_bpack.mdl	revised version of id original backpack model with new skins.	starshipwaters
candle.mdl	candle	Quake Mission Pack 2 *Dissolution of Eternity* (a.k.a. *DOE*)
mervup.mdl	multi-grenade	same as above
lspike.mdl	laval spike	same as above
debris.mdl	model for breakables inspired by Rubicon 2 models	starshipwaters
g_shotgu.mdl	shotgun model	starshipwaters
spark.mdl	spark particle	starshipwaters
s_null.spr	empty sprite for various effects	dumptruck_ds
h_boss.mdl	Chthon head based on original boss.mdl	starshipwaters
s_flame.spr	new animated fire sprite	starshipwaters
spike.mdl	id original	added lava skin from *DOE* to original model
armshr.mdl	new armor shard inspired by ijed's from ReMakeQuake	starshipwaters
soldier.mdl	id original	additional skins by dumptruck_ds
ogre.mdl	id original	*DOE* version with additional skins by dumptruck_ds and JaycieErysdren
h_ogre.mdl	id original	*DOE* version
enforcer.mdl	id original	additional skins by dumptruck_ds and ijed
teleport.mdl	new replacement model with 30 additional skins, used for misc_teleporttrain but	starshipwaters

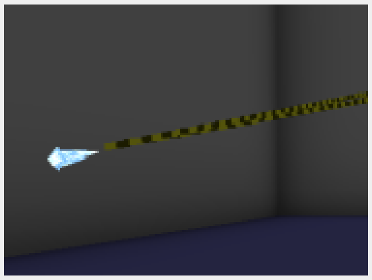
	intended as a general projectile model or decoration	
v_spike.mdl	new replacement model with 30 additional skins, used for Shalrath projectiles	starshipwaters
k_spike2.mdl	id original model with 26 additional skins, used for the Death Knight's explosive projectiles	starshipwaters
w_spike.mdl	id original model with 26 additional skins, used for the Wizards projectiles	starshipwaters with assistance from Kebby_
pd_r_key.mdl	remake of Rune key inspired by id original with 25 additional skins	starshipwaters
pd_b_key.mdl	remake of Base key inspired by id original with 25 additional skins	starshipwaters
pd_w_key.mdl	remake of Wizard key inspired by id original with 25 additional skins	starshipwaters
m_h15.mdl	rotten healthpack model	by Lunaran from Copper
m_h25.mdl	normal healthpack model	same as above
m_h100.mdl	mega healthpack model	same as above
pd_vial.mdl	health vial model	starshipwaters
m_cells1.mdl	box of cells	by Lunaran from Copper
m_cells2.mdl	large box of cells	same as above
m_nails1.mdl	box of nails	same as above
m_nails2.mdl	large box of nails	same as above
m_rock1.mdl	box of rockets	same as above
m_rock2.mdl	large box of rockets	same as above
m_shell1.mdl	box of shells	same as above
m_shell2.mdl	large box of shells	same as above

Sounds

Asset Name	**Details**	**Credits**
bricks1.wav metal1.wav metal2.wav stones1.wav wood1.wav wood2.wav	hard coded breakable sounds	dumptruck_ds
pd_water.wav	Underwater sound	dumptruck_ds
elec22k.wav	looping electricity sound	dumptruck_ds
rumble.wav	earthquake sound	Rogue mission pack
armsh1.wav	armor shard pickup sound	modified id sound
magic_01.wav	looping magical hum	dumptruck_ds (using Virtual ANS)
pd_pop2.wav	new pain sound for killable Shub	modified id sound

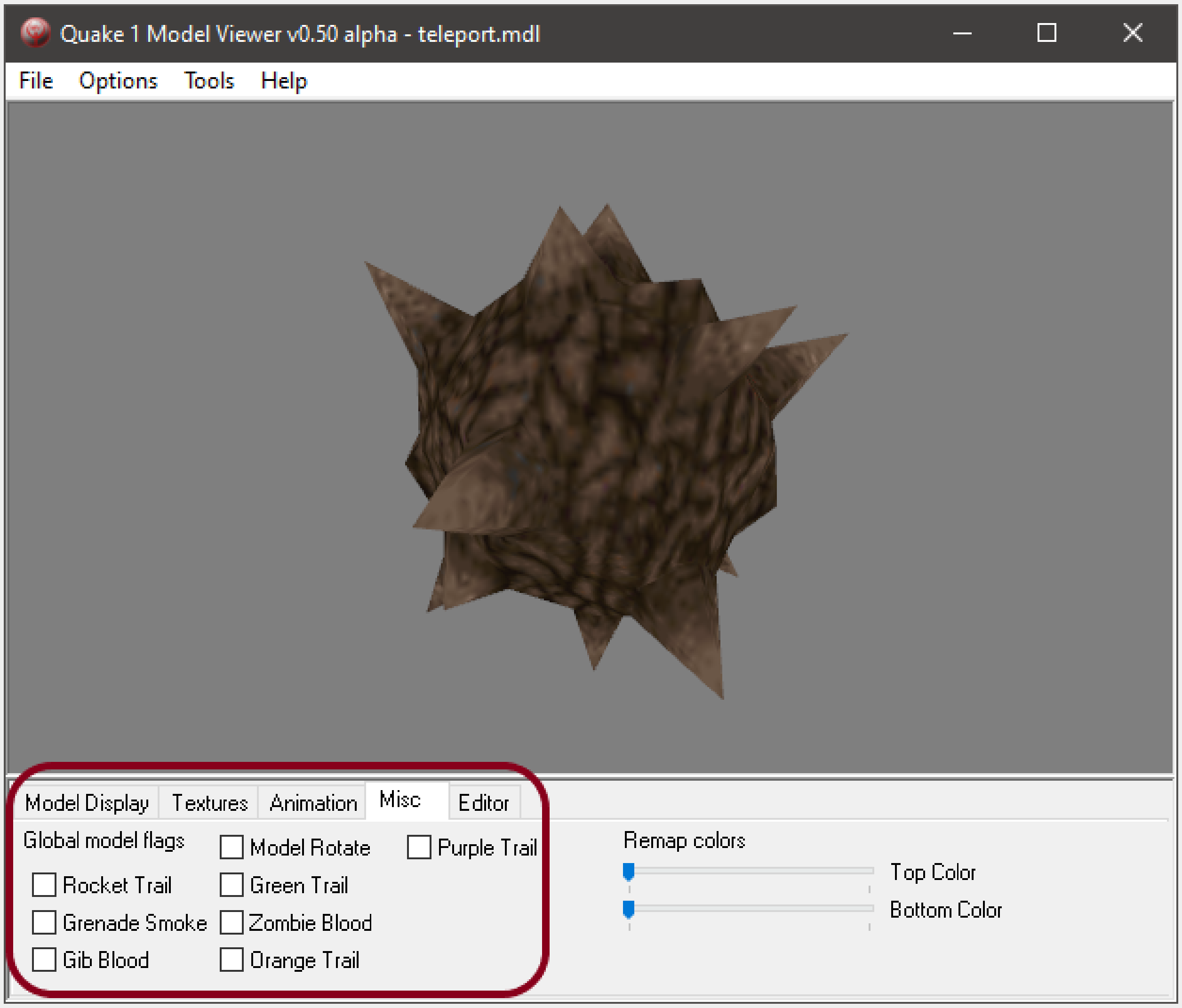
Using Model Flags

New assets in version 3.0.0 of *progs_dump* are intended to give mappers more options for decorations or alternate projectiles. Some models will have a flag set to display a certain visual effect. For example, the w_spike.mdl is the projectile used by Wizards. By default, there is a green trail on the model.



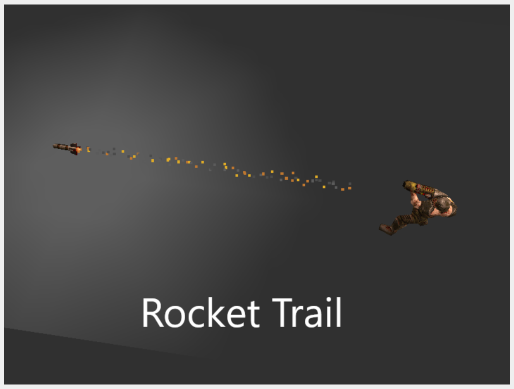
But what if you wanted to use a different skin on this model to make an “ice shard” projectile for another monster? The green trail would look out of place, so you’d want to remove the flag.

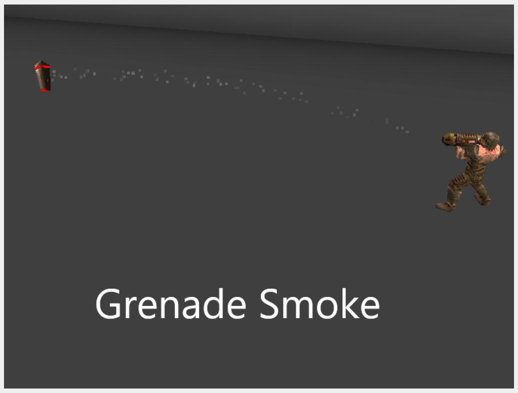
Using Quake Model Viewer or a similar tool, you can change a given flag for a model using the checkboxes shown here:

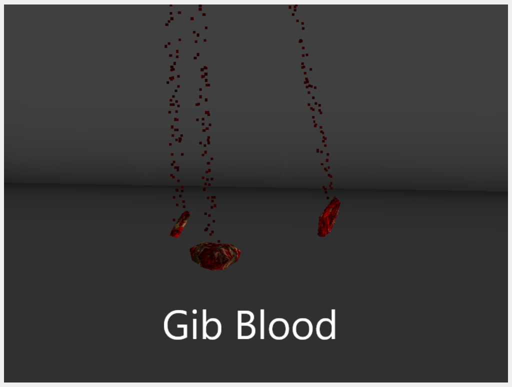


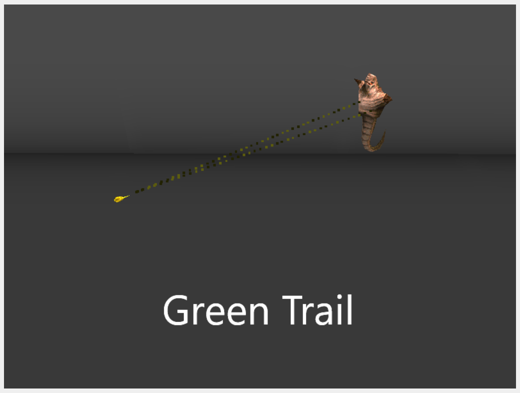
You should always save a copy of the model if you change its flag so you can use them both for their separate applications.

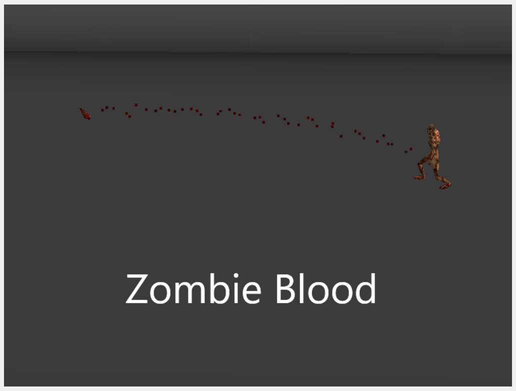
Here are examples of the different flags available. These are hard coded by the engine so you can not modify them, only enable or disable one flag at a time.

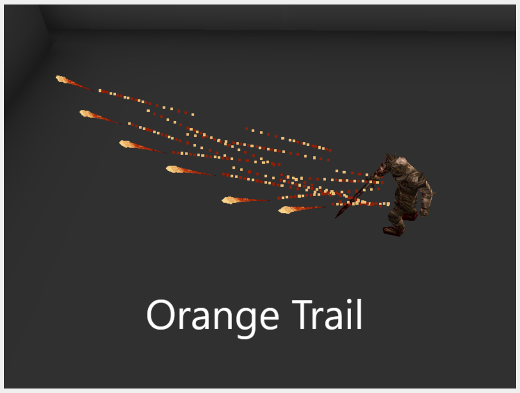


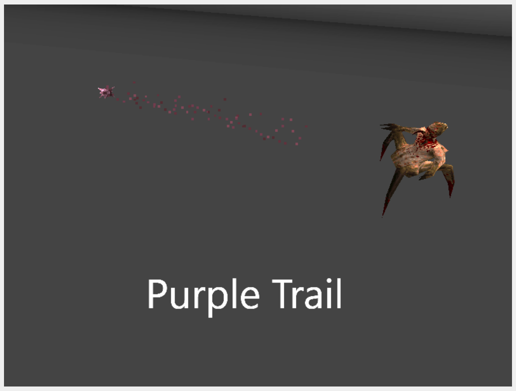






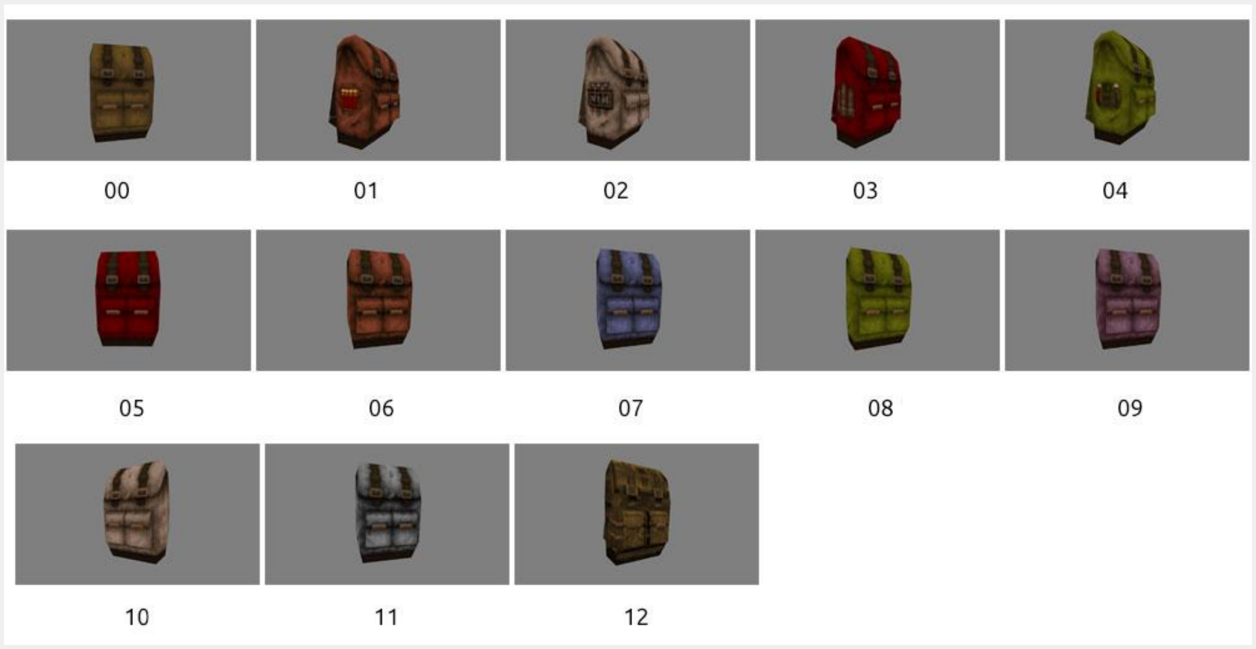






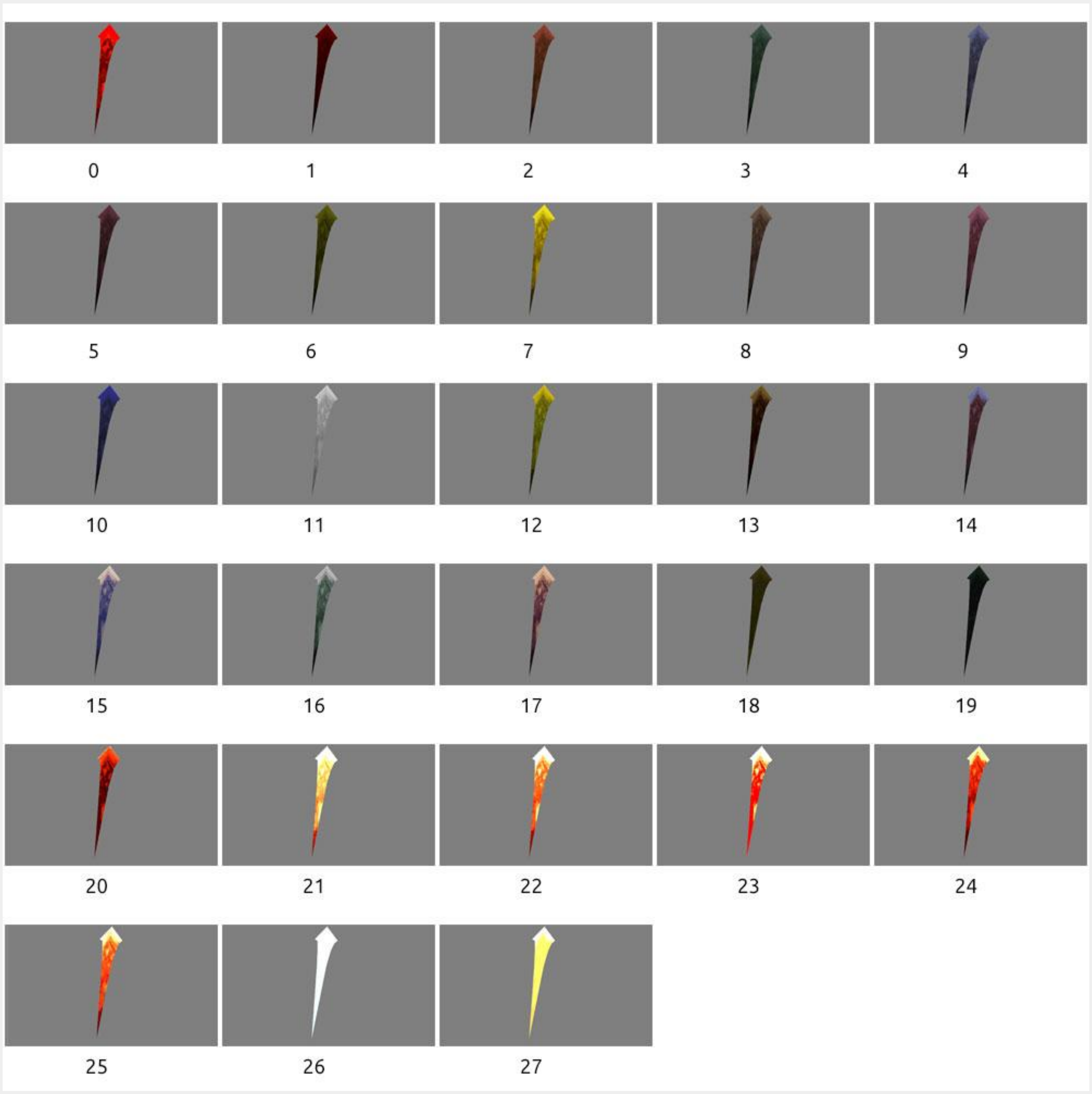
pd_bpack.mdl skins

This model has the *Model Rotate* flag set by default.



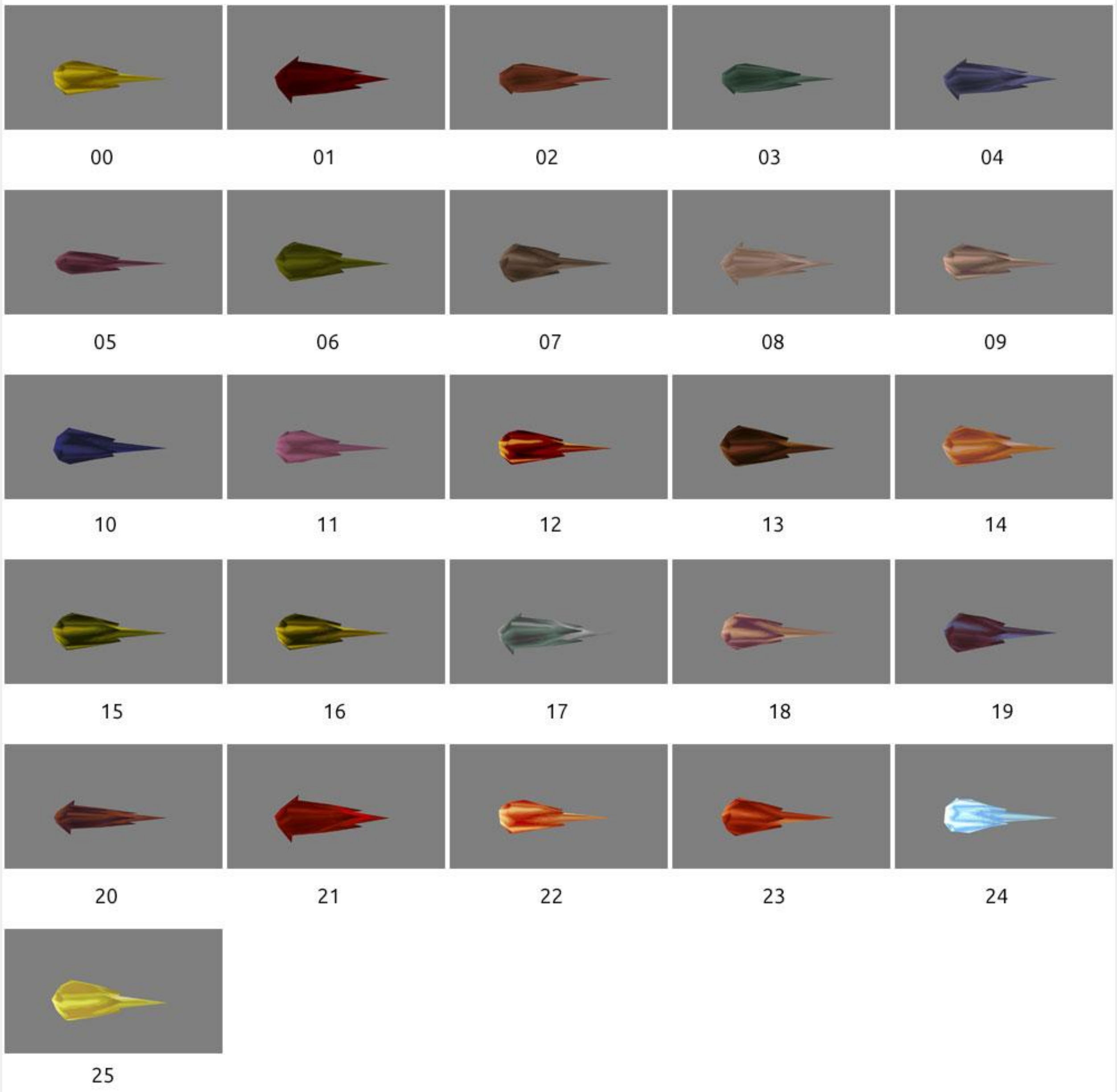
k_spike2.mdl skins

This model has the *Orange Trail* flag set by default.



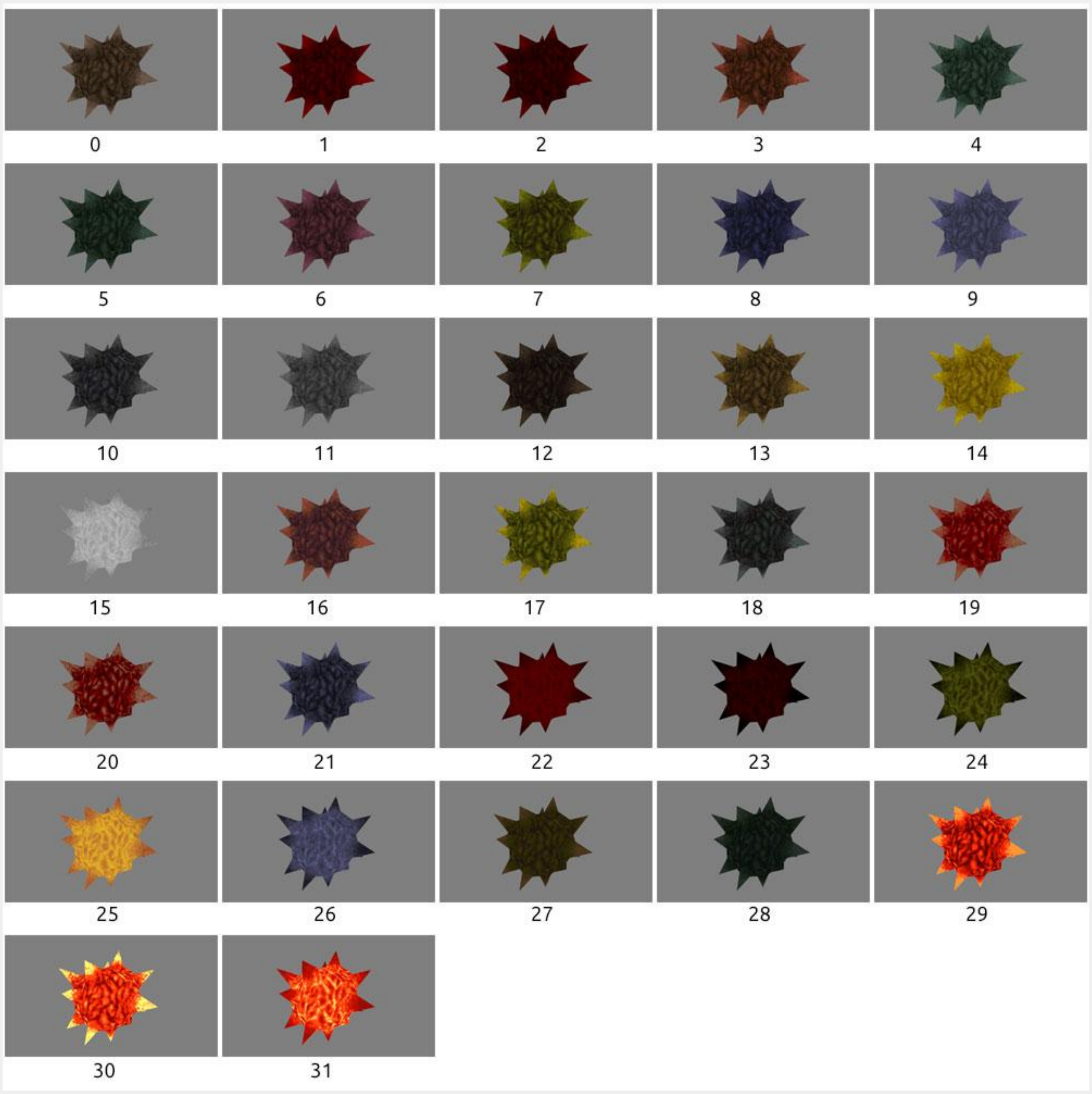
w_spike.mdl skins

This model has the *Green Trail* flag set by default. Special thanks to **Kebby_** for helping solve a technical issue with this one!



teleport.mdl skins

This model has no flag set by default.



pd_r_key.mdl skins

This model has the *Model Rotate* flag set by default.

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

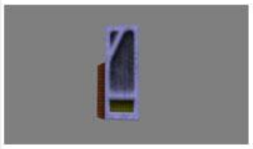
23

24

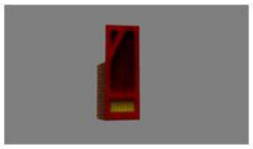
25

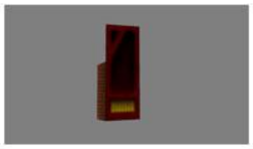
pd_b_key.mdl skins

This model has the *Model Rotate* flag set by default.


0

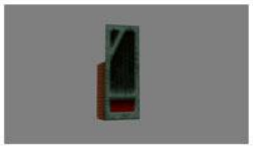
1


2


3

4

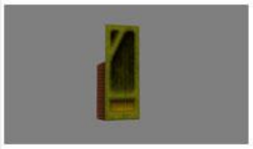
5


6

7

8

9


10

11

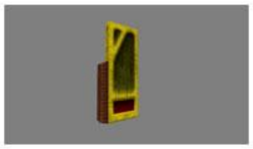
12

13

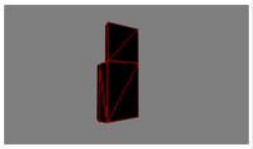
14

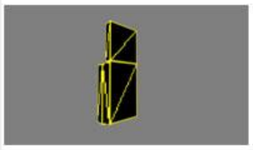
15

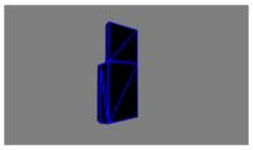
16

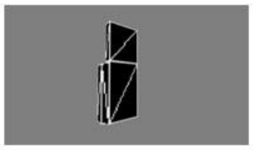

17

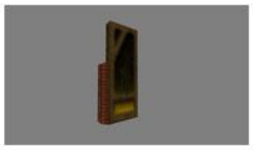
18

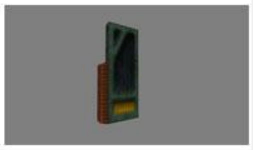

19


20


21


22

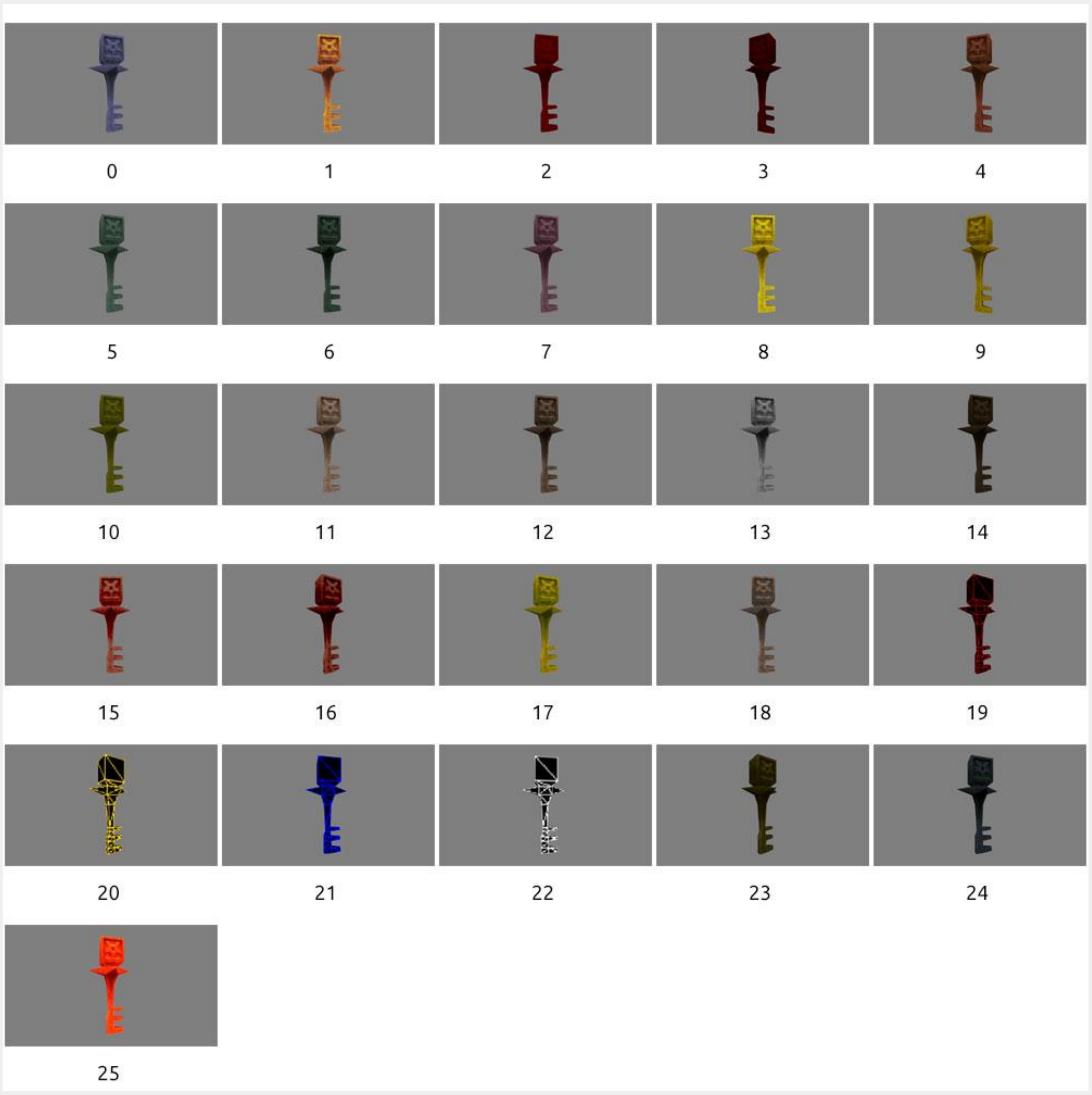

23


24

25

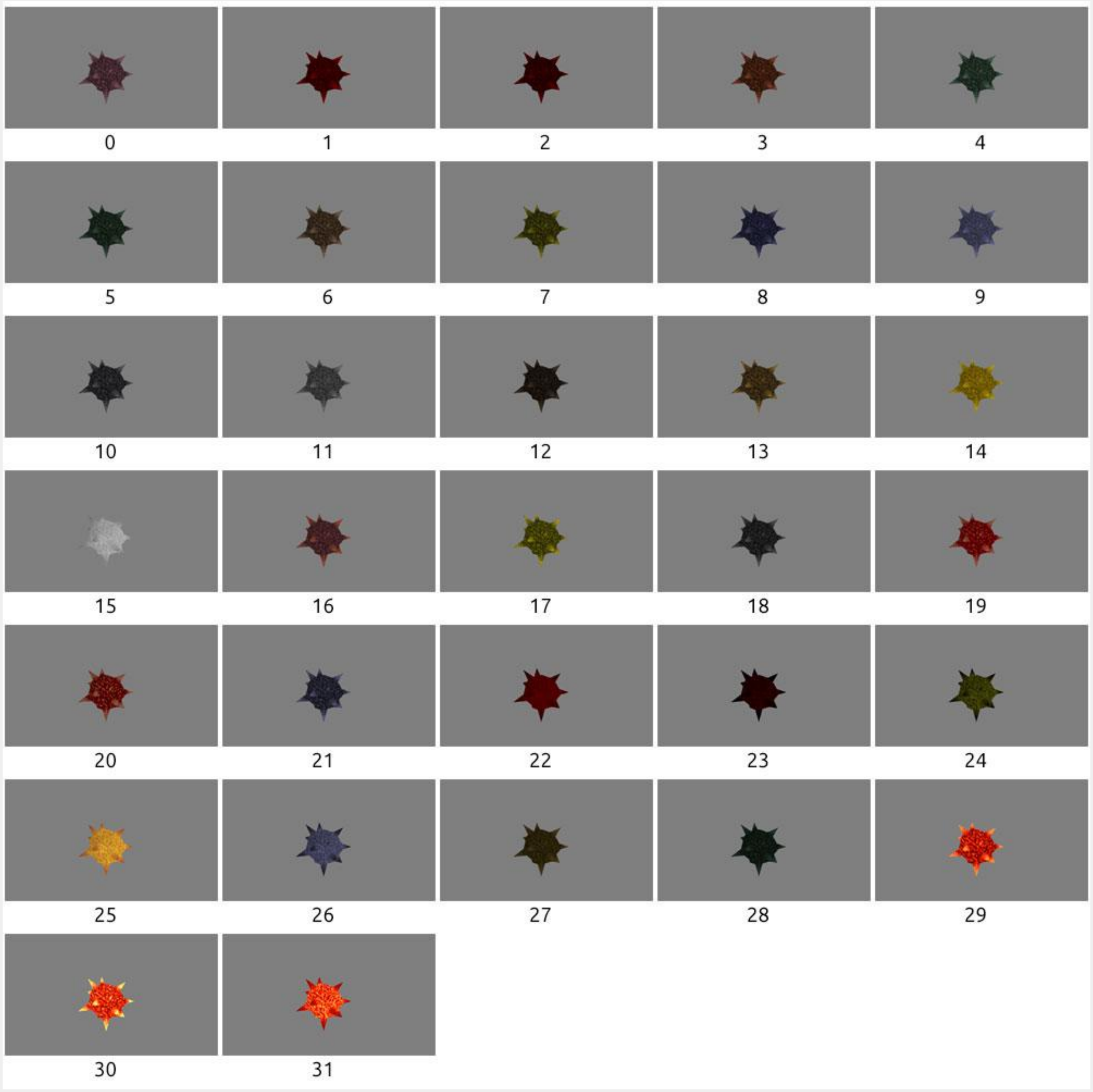
pd_w_key.mdl skins

This model has the *Model Rotate* flag set by default.

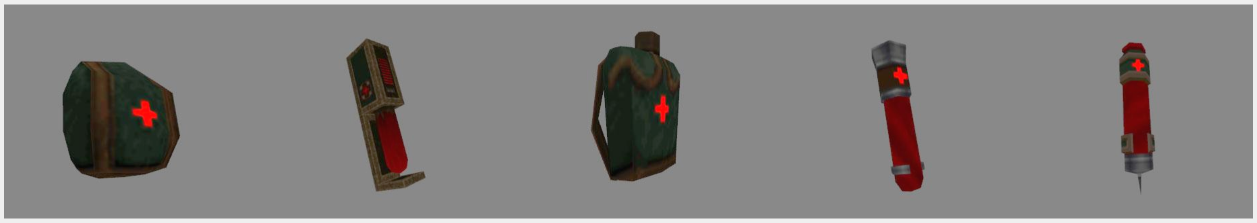


v_spike.mdl skins

This model has the *Purple Trail* flag set by default.



Alternate Health Vials



If you'd prefer a different style of health pack, you can download these and replace the default health vial using the *mdl_body* key as with other assets in the devkit. You can also set the default in the Worldspawn key. These were all created by starshipwaters. The models have the *Model Rotate* flag set by default.

Appendix B: Finding Custom Models

As this manual is being written, Quake is entering its 25th year. Over those years, hundreds of Quake mods have included custom models, many of which are compatible with the stock Quake assets. Below are *some* of the resources we've used to test with *progs_dump*. Following that, a short overview of how to extract models and check them for compatibility.

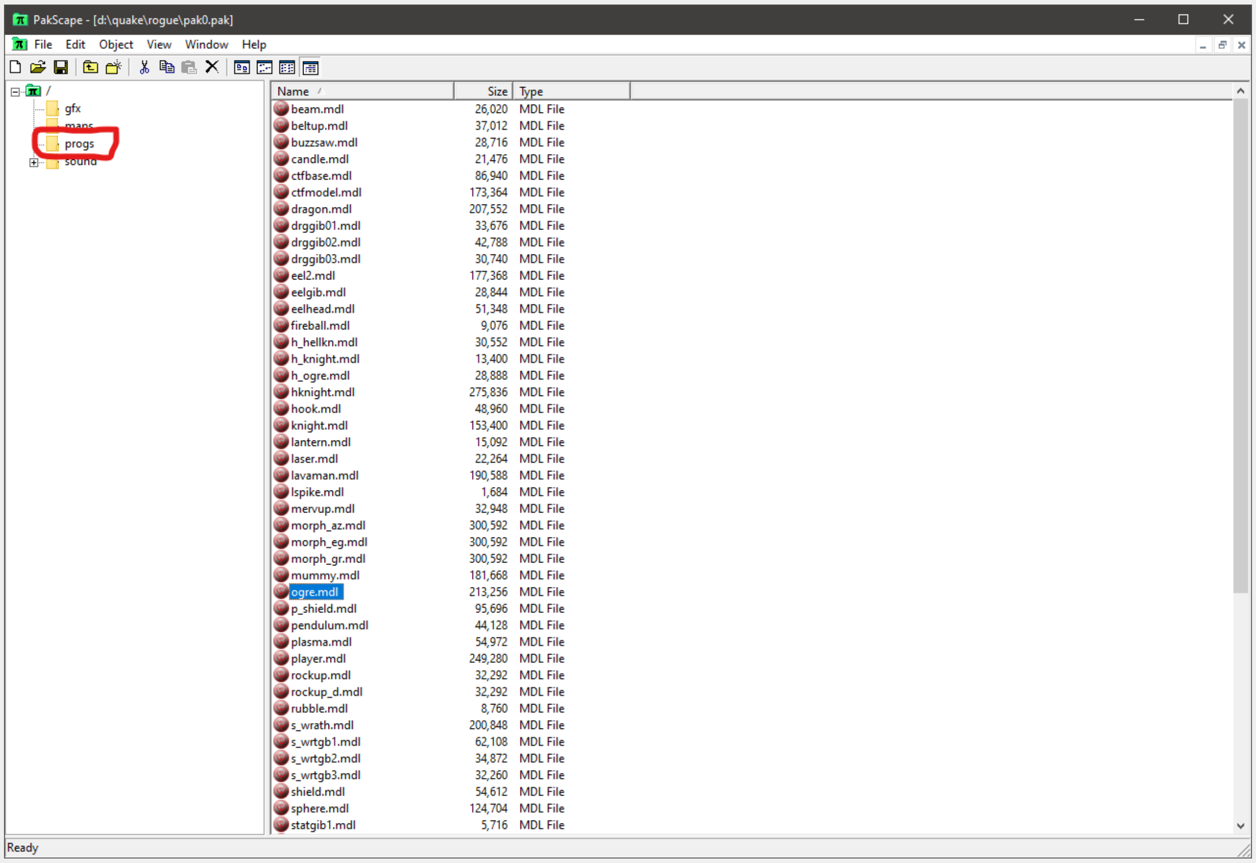
Don't forget to check if you have permission to use mod assets in your projects. Usually this is stated in the readme.

Name	Type	Details
The Keep mod	mod	A huge mod based on the Arcane Dimensions codebase. It's still in development as of December 2020. This mod takes hundreds of monsters and other assets from scores of mods and lumps them into one giant release. This one resource makes it *really* easy to find monster replacements for the stock id creatures from a variety of mods. More info.
Arcane Dimensions	mod	Features all kinds of skins for all the traditional id monsters. Versions 1.8 and above, use PAK files. You'll need PakScape to extract assets. Or use 1.7 and below for unpacked assets.
Chillo's Model Pack 1.7	mod	Replacement models for almost every monster.
Quoth	mod	Similar to Arcane Dimensions. Some good models and skins that are compatible with the original monsters. Plus lots of other models and sprites.
ReMakeQuake	mod	canceled mod, assets are free to use per the author
Drake (beta)	mod	Fun mod with dragons and all kinds of monster skins and misc models.
pub/idgames2	ftp	Massive directory of id related content dating back 25+ years. Models, mods, total conversions and more.
Quaketastic models	http	Another massive collection of models to peruse.
Quaketastic mods	http	A decent collection of Quake mods.

If you come across custom content for Quake that uses PK3 and/or MD3 files, it's likely that that modification will *only* work with the *DarkPlaces* engine. We don't recommend using *DarkPlaces* with progs_dump for a variety of reasons.

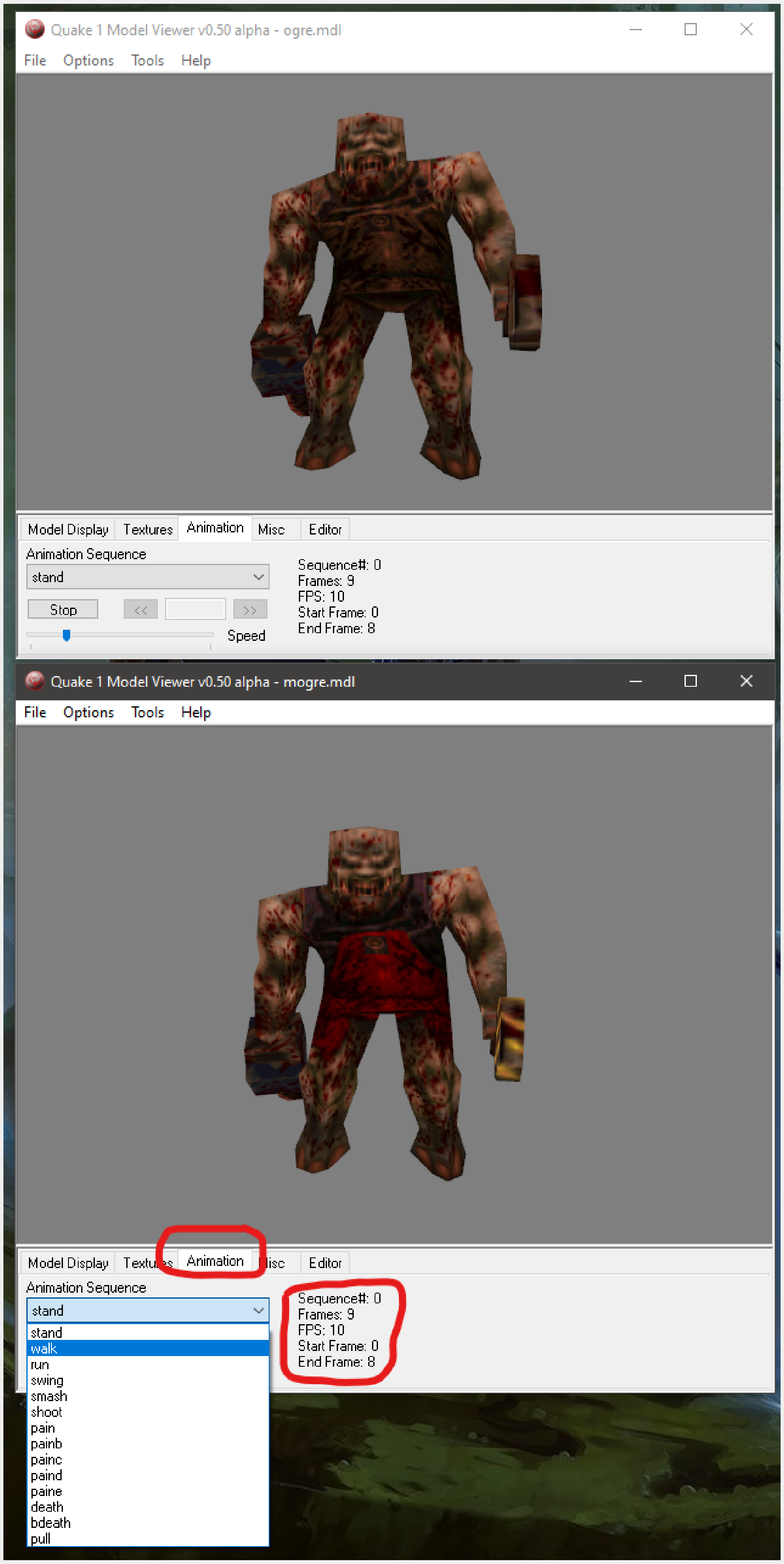
Once you have a model you want to use in *progs_dump* you may need to extract it with PakScape. Just open the PAK file and click on the progs folder. Drag and drop the model to extract it. Models should be placed under the progs directory in your mod but you can use sub folders to keep them organized.

Remember that you can also use sprites (.spr) in *progs_dump* with a variety of entities. And of course, sound (.wav) files.



Name	Size	Type
beam.mdl	26,020	MDL File
beltup.mdl	37,012	MDL File
buzzsaw.mdl	28,716	MDL File
candle.mdl	21,476	MDL File
ctfbase.mdl	86,940	MDL File
ctfmodel.mdl	173,364	MDL File
dragon.mdl	207,552	MDL File
drggib01.mdl	33,676	MDL File
drggib02.mdl	42,788	MDL File
drggib03.mdl	30,740	MDL File
eel2.mdl	177,368	MDL File
eelgib.mdl	28,844	MDL File
eelhead.mdl	51,348	MDL File
fireball.mdl	9,076	MDL File
h_hellkn.mdl	30,552	MDL File
h_knight.mdl	13,400	MDL File
h_ogre.mdl	28,888	MDL File
hknight.mdl	275,836	MDL File
hook.mdl	48,960	MDL File
knight.mdl	153,400	MDL File
lantern.mdl	15,092	MDL File
laser.mdl	22,264	MDL File
lavaman.mdl	190,588	MDL File
lspike.mdl	1,684	MDL File
mervup.mdl	32,948	MDL File
morph_az.mdl	300,592	MDL File
morph_eg.mdl	300,592	MDL File
morph_gr.mdl	300,592	MDL File
mummy.mdl	181,668	MDL File
ogre.mdl	213,256	MDL File
p_shield.mdl	95,696	MDL File
pendulum.mdl	44,128	MDL File
plasma.mdl	54,972	MDL File
player.mdl	249,280	MDL File
rockup.mdl	32,292	MDL File
rockup_d.mdl	32,292	MDL File
rubble.mdl	8,760	MDL File
s_wrath.mdl	200,848	MDL File
s_wrtgb1.mdl	62,108	MDL File
s_wrtgb2.mdl	34,872	MDL File
s_wrtgb3.mdl	32,260	MDL File
shield.mdl	54,612	MDL File
sphere.mdl	124,704	MDL File
statgib1.mdl	5,716	MDL File

Next, take a look at the animations and compare them to the original id models in Quake 1 Model Viewer using the animation tab. You can also just load the models in *progs_dump* and see if they work!



```
Sequence#: 0
Frames: 9
FPS: 10
Start Frame: 0
End Frame: 8
```

```
Sequence#: 0
Frames: 9
FPS: 10
Start Frame: 0
End Frame: 8
```

Appendix C: Development Folder

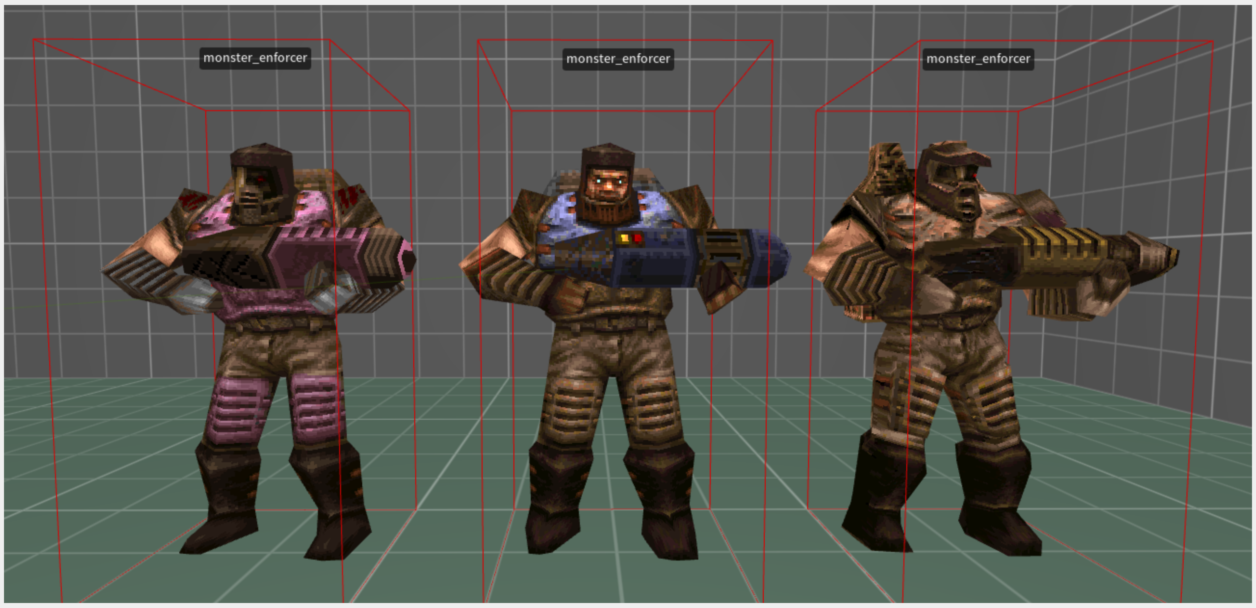
The development folder in the main *progs_dump* mod contains the source files for the sample maps and prefab maps, the entity definition files, a texture wad file and the QuakeC source.

fgd_def

FGD and DEF files are used by map editors to display entity information. *progs_dump* has three of these for different applications. Great care has been taken to update these files with as much information from this manual as possible. Also, the light and worldspawn entities have expanded definitions for ericw's compiling tools.

File	Details
pd_300.def	Quake Definition file for use with *x*Radiant editors like QuakeEd3, GTKRadiant and NetRadiant-Custom. TrenchBroom also supports the.def format.
pd_300_JACK.fgd	Used with J.A.C.K.
pd_300.fgd	Used with TrenchBroom

If you use the *pd_300* fgd you will see custom models in TrenchBroom including skin selections.



map src

This folder contains the .map files for all the sample and prefab maps as well as the custom debris .map files as described in the custom breakables section above.

quakec scr

This folder contains the QuakeC source files for the mod. You are free to use this as a basis for a new mod. Please give us credit and include the source files for your mod in the public release. These QuakeC files are also included in the *mod_template.zip* folder and should always be included with your mod whether you modify them or not.

wads

pd_300.wad is a single wad file containing all of the textures used in the sample maps. This includes key door textures for the included pd_key models. Use this when you refer to the sample files in a map editor. You are free to use any included texture in your own projects.

go map!