Beginner's Guide

This guide will cover the basics required to get started with level modding.

Section 1: Before you begin modding

Section 1 will cover everything you need to know before you can begin editing or making your own enemy encounters.

There will be non-essential steps that you don't technically need to do, but it will streamline the modding process and be good in the long run. Watch out for those.

Section 1: What is a level mod?

In order to untangle the mess that is teaching the pre-requisite knowledge of being able to create level mods on your own, it is best to lay down the process step by step.

This is something I've noticed this wiki does not do. It may have a lot of useful documentation such as eventcalls and miscellaneous entities such as damage triggers, particle effects, etc., but it does not put together a concise tutorial on how to use them properly.

This guide sets out to correct this.

First, a brief write-up of what exactly level mods are and how they work.

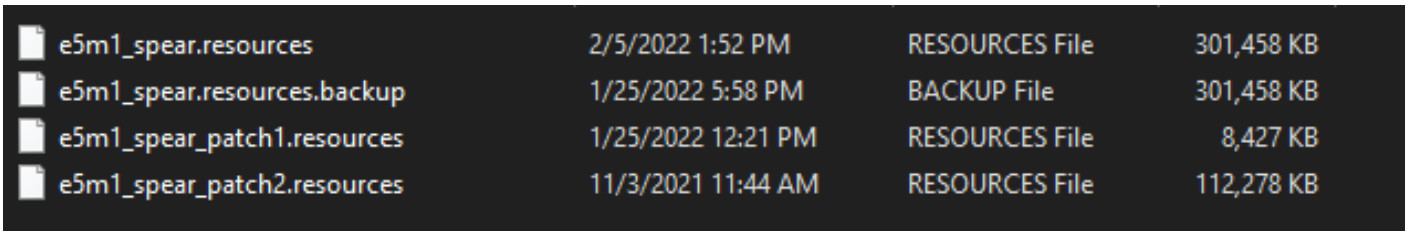
> TL;DR: A level mod replaces the "level file" within a map's .resources file(s), which contains the sum total of all of its assets.

A single map in Doom Eternal has the sum total of its assets and files located within its RESOURCES files, with the file extension ".resources".

A RESOURCES file is what you would imagine it would be, a file that is jam packed with a huge amount of assets and resources that, when combined and used by the map, form the entire experience of playing through it.

You can even see them for yourself. On the Steam/Windows version of Doom Eternal for example, the RESOURCES files for the World Spear for example are located in:

```
C:\Program Files (x86)\Steam\steamapps\common\DOOMEternal\base\game\dlc2\e5m1_spear
```



The _patch files are merely extensions of the main file and are noticeably smaller in size, although _patch2 is larger because it contains all of the World Spear's official Master Level assets.

These RESOURCES files, when combined, have everything the map needs to function properly. They have everything from particle effects to models to, most importantly, the file that determines how the map operates when loaded.

This file, with the file extension .entities, is what id Software themselves change when they push a new master level out.

This file is highly important, it takes all the individual assets in a map's RESOURCES file and mashes them together into a single playable experience. Particle effects, hazards, enemy encounters, triggers. Everything.

So all in all, these RESOURCES files, especially the .entities file, are pretty important then. The only thing they *don't* have is the map geometry itself.

Level mods edit these RESOURCES files directly. Specifically, they **replace** the .entities files within. You know, the thing id Software themselves change when they push a new master level out.

Next step, we will gather the required tools needed to create level mods in a fast and efficient way. Don't worry, it's not a lot!

Section 1: Required tools part 1

Now that we know that the file we actually edit to create a level mod is the .entities file, we will need a way to edit it.

And not just edit it, but find a way to edit it on the fly so we don't need to re-compress and re-inject the file after each and every change.

It is important to note a few, crucial details, each of which will have a corresponding tool or helpful asset to help with that.

1. As you have seen in the last page, the .resources files aren't exactly openable. We will need to extract the .entities file from them to edit it.

2. If you were to open the .resources files of a map, including its _patch versions, you may notice multiple ones have their own .entities file. What gives?!

3. This .entities file is actually just text and can be opened with any text editor. Unlike id Software who uses their own software called idStudio to edit levels in a 3D environment, we are left with 2d text. An .entities file purely consists of defined entities, or entitydefs. Entitydefs can be anything from triggers to encounter managers to pickups.

Example 1: An entity that is defined to be a spawn point for a marauder.

```
entity {
	entityDef encounter_5_marauder1 {
	inherit = "target/spawn";
	class = "idTarget_Spawn";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = false;
	disableAIPooling = false;
	edit = {
		{  The stuff that usually goes here }
	}
}
}
```

Example 2: An entity that is defined to be shotgun ammo.

```
entity {
	entityDef pickups_pickup_ammo_shells_26 {
```

```
	inherit = "pickup/ammo/shells";
	class = "idProp2";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = false;
	disableAIPooling = false;
	edit = {
		[ The stuff that usually goes here]
	}
}
}
```

4. If you ever tried to open the .entities file within an existing level mod, you may notice that it's made up of nonsense weird characters. This is because the file is compressed. Decompressed files are the plainly text versions you see, but are too large to replace an existing .entities file, which is why they are compressed before being put into a level mod.

5. Assuming you want to spawn in an enemy on a brand new spawn point, you do not have any readily available way to have your current coordinate match the file's formatting.

Example 1: A basic spawnposition and spawnorientation. Does this look like you can manually type it out by hand, every single time?

```
spawnOrientation = {
	mat = {
		mat[0] = {
			x = 0.707107;
			y = -0.707107;
			z = 0.000000;
		}
		mat[1] = {
			x = 0.707107;
			y = 0.707107;
			z=0.000000;
		}
		mat[2] = {
			x = -0.000000;
			y = 0.000000;
			z = 1.000000;
		}
	}
```

```
}
spawnPosition = {
	x = -99.080002;
	y = -278.399994;
	z = 22.00265;
}
```

If these details made no sense to you, don't worry. I will provide each tool and let you know how they work in setting up your foundation to edit a level with.

They are:

1. SAMUEL to extract the .entities file from the .resources files.

Download link for SAMUEL: GitHub

2. A list of a map's current highest priority so you know *which* .resources file to extract from and subsequently edit. The game uses the .entities file from the map's highest priority patch. If you choose the wrong one and have your mod replace the .entities file in the wrong patch, it won't work.

Link to the list: Google Docs

3. NotePad++ to edit the .entities file with. You may also use EntitySlayer, a program that is specifically designed to work with these files, but this guide will not use it as I have no experience with it.

Download link for NotePad++: Official website

Download link for EntitySlayer: GitHub

4. idFileDeCompressor. You will not need to use this tool to decompress extracted .entities files as SAMUEL neatly gives them to you already readable and editable, but you will need it to re-compress them when packing them in the final .zip file in order to let level mods use it without giving you a memory error. EntitySlayer also has an option to compress files without the need of a dedicated compressor tool.

Download link for idFileDeCompressor: GitHub

5. meathook.dll. This tool is so useful for level modding that without it, I would say at least 99% of all master level mods would not exist today because they would've taken too long to make, ludicrously long. Meathook allows for on the fly editing and without it level modders would've been stuck re-compressing and re-injecting their edited .entities file over and over.

Download link for v7.2: Direct download

Next up, we will go over how to use these tools to set up your foundation for making a level mod.

Section 1: Required tools part 2

Now that you have the needed tools, let's go over how to use them to set up your level mod. Don't worry, you'll be editing encounters in a step or two.

First, do the following:

> Make sure you have the mod injector and all of its related contents installed beforehand, including file replacements, the mod injector, the mods folder, etc.

1. Download SAMUEL and extract its contents to anywhere you want.

2. Open the list of the game's highest priority resources and have it on standby on a tab somewhere.

3. Install NotePad++.

4. Within IDCOMPRESSOR.zip, open the compressor files in which there should be two files, a .exe and a .bat. Place both files where you also put your mod injector, the DOOMEternal folder.

5. Place meathook, which is going to be a file named XINPUT1_3.dll, where you put the idFileDecompressor files, the DOOMEternal folder.

Now that we have all of our tools installed and put where they need to be, we can get started on setting up your mod.

We will only need the first 3 of these tools to begin, meathook is used during the modding process and idFileDecompressor is for when you're done with it.

We aren't going to setup the .zip files you see that are parts of mods, that will come at the very end.

Instead, we are going to setup an **overrides** folder that allows for the live editing of .entities files.

Here, we are going to set up an overrides file path for Cultist Base.

The process for each of them vary only in one step. You will see what I mean.

1. Decide what map you want to edit. In this case, we've already decided on the aforementioned map.

2. Find the names of the map as defined by the game. Cultist Base is e1m3_cult.

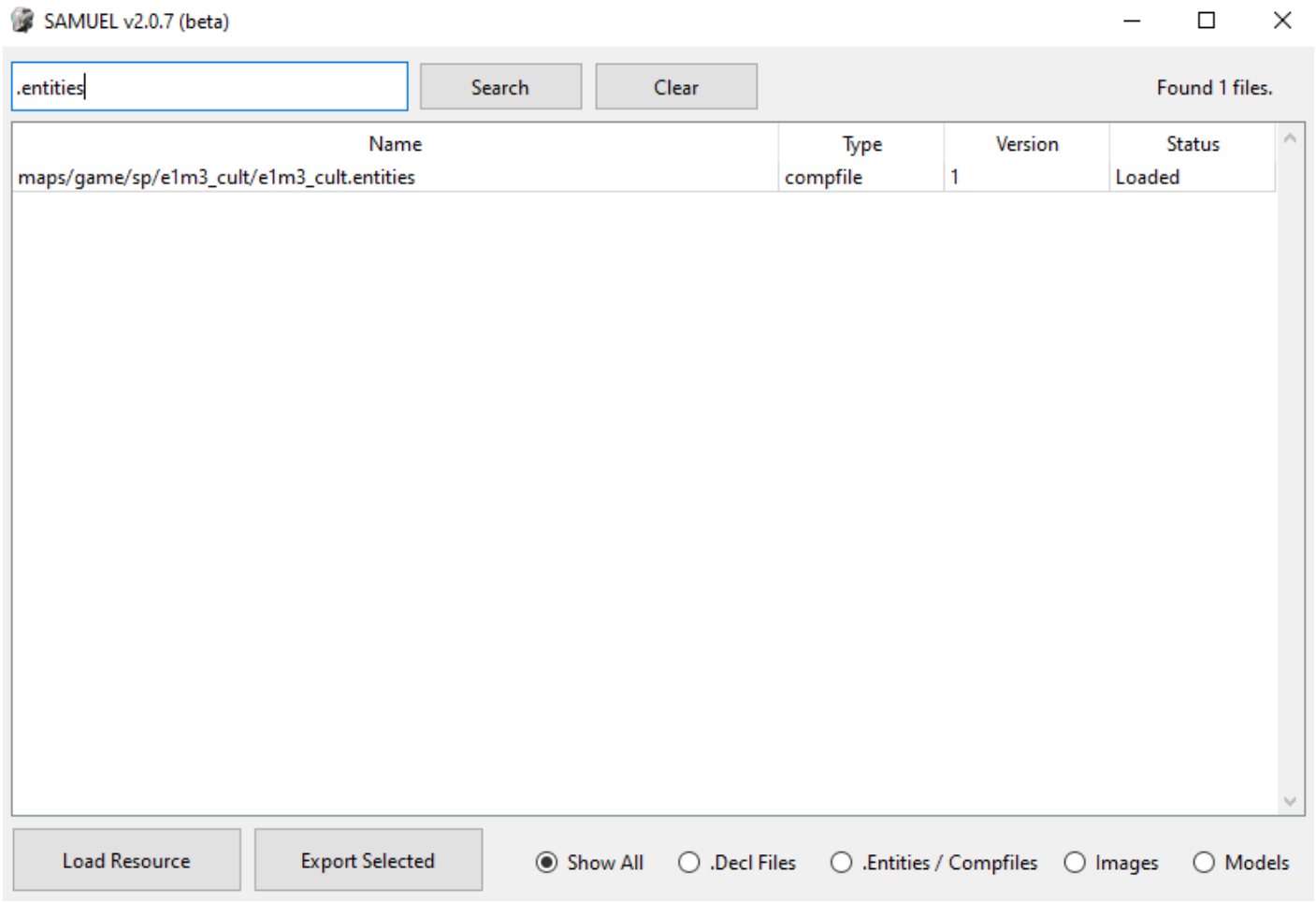
-You can find the names of every map **here.**

3. Using SAMUEL, we will extract the *correct* .entities file for each of the three. Open SAMUEL.exe, and click on Load Resource on the bottom left corner. A window will pop up asking you to select a file. We are going to extract Cultist Base first, so navigate to its .resources files.

```
C:\Program Files (x86)\Steam\steamapps\common\DOOMEternal\base\game\sp\e1m3_cult
```

4. When you open this folder, you will see multiple .resources files, most of which you will find has its own .entities file if you were to open them up. In order to extract the correct one, we will get it from the .resources file that has the highest priority. Open up the list of highest priorities, and CTRL+F "e1m3_cult". The **first** result that pops up is e1m3_cult_patch3, meaning that the .entities file in this specific .resources file will be the one the game uses. Open e1m3_cult_patch3.resources using SAMUEL.exe

5. Look for .entities in the search bar. There will be one result. A file named e1m3_cult.entities. Double click on it to export it.



6a. The file will be exported to the exports folder that is where SAMUEL.exe is located. Open the file with NotePad++, and make sure all future .entities files are opened by it.

6b. Export the file again to keep around as a reference point for the default .entities file. You don't have to do this, but it will make things easier.

7. Now that we have the correct .entities file for Cultist Base, we are going to create an overrides folder in the DOOMEternal folder and place it in there so that it, the editable text file, directly overrides what the game uses so we can save our changes and reload quickly and efficiently. More on reloading in section 2. Within the DOOMEternal folder, create a new folder called overrides.

8. You may have noticed there is a specific file path to the .entities file when you look it up in SAMUEL. We will have to recreate this exact file path within our new overrides folder. When you are done, your file path to your new editable file will look like this:

```
DOOMEternal\overrides\maps\game\sp\e1m3_cult\ <-- put the exported file in here
```

Your .entities file should now be here:

```
DOOMEternal\overrides\maps\game\sp\e1m3_cult\e1m3_cult.entities <--
```

> When making overrides for TAG1 and TAG2, "sp" must be replaced with "dlc" and "dlc2" respectively. This is because "sp" is specifically in reference to the base campaign. Examples of DLC level overrides are below.

```
DOOMEternal\overrides\maps\game\dlc\e4m3_mcity\e4m3_mcity.entities
DOOMEternal\overrides\maps\game\dlc2\e5m2_earth\e5m2_earth.entities
```

Let's test to make sure the overrides folder is added correctly. We're going to make an immediately noticeable change: switching the music. Open your new .entities file and CTRL+F idMusicEntity. Replace "cultist_base_music" with the music from Nekravol, "metal_hell_music".

More information on idMusicEntity(s) **here.**

Once you've made the change, open up DOOM Eternal and load up Cultist Base. If you already had Cultist Base open, type mh_force_reload in the console to reload the change. If the music is Nekravol's you've installed the overrides correctly and we can now start editing encounters. If it is not, please go over the last few pages again until you've successfully changed the music.

Of course, when you are ready to move on, feel free to revert the music. It was just a test.

Section 2: Let's begin modding

Just as a refresher, this section assumes you have an editable .entities file in the correct place in which you can make changes on the fly.

Here, we are going to do the very core of level modding, editing encounters. The full walkthrough will include:

1. Learning about idAI2(s), idTarget_Spawn(s), idTargetSpawnGroup(s), and idTarget_Spawn_Parent.
2. Using this knowledge to replace an existing demon in an existing encounter, then to...
3. Create and spawn a new demon in an existing encounter, then to...
4. Create an entirely new encounter.

Afterwards, we will move on to section 3, which will consist of just about everything else you can do in an encounter such as changing the music state, making and spawning our own orange barriers, event flags, and activating hazards.

Section 4 will cover all the miscellaneous stuff such as importing DLC ai and resources.

Section 2: idAI2s and spawns

Before we edit an encounter, we need to know about the behind-the-scenes elements going on.

Specifically, this section will cover idAI2s and spawns.

idAI2s

Earlier, I said that:

> An .entities file purely consists of defined entities, or entitydefs. Entitydefs can be anything from triggers to encounter managers to pickups.

An idAI2 is an entity for a specific demon.

What do I mean by that? Let's take a look at an idAI2 for an Imp.

```
entity {
	layers {
		'spawn_target_layer"
	}
	entityDef ai_custom_imp {
	inherit = "ai/fodder/imp";
	class = "idAI2";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = true;
	disableAIPooling = false;
	edit = {
		[  The stuff that usually goes here ]
	}
}
}
```

You can CTRL + F "ai/fodder/imp" to see what's usually in the edit section. It consists of important but irrelevant settings so I will not be including it here.

Think of idAI2s as the demon itself. These entities, however, will not do anything if you were to copy + paste a new one into your file. They need to be *spawned* in order to exist as you typically see them in-game.

How can we spawn them? We will get to that in this next section:

idTarget_Spawns

idTarget_Spawns are exactly what you think they are: they're spawn points for demons.

The catch is though: they must reference an idAI2 in some way.

> I cannot stress this enough, a target spawn *must* reference an idAI2 in some way, either by itself or by being part of a spawn zone that references a spawn parent that references idAI2s.

Making a spawn point is easy: copy + paste an existing one and edit the four important things:

- entitydefs
- aistateoverride
- spawnOrientation
- spawnPosition

It is these four, *and only these four*, that matter.

Here is an example of a custom spawn point, notice how it references our Imp idAI2 from earlier. I have also marked the other important things to edit with comments.

```
entity {
	entityDef custom_imp_spawn_1 {
	inherit = "target/spawn";
	class = "idTarget_Spawn";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = false;
	disableAIPooling = false;
	edit = {
		flags = {
			noFlood = true;
		}
		spawnConditions = {
			maxCount = 0;
			reuseDelaySec = 0;
			doBoundsTest = false;
```

```
		boundsTestType = "BOUNDSTEST_NONE";
		fovCheck = 0;
		minDistance = 0;
		maxDistance = 0;
		neighborSpawnerDistance = -1;
		LOS_Test = "LOS_NONE";
		playerToTest = "PLAYER_SP";
		conditionProxy = "";
	}
	spawnEditableShared = {
		groupName = "";
		deathTrigger = "";
		coverRadius = 0;
		maxEnemyCoverDistance = 0;
	}
	entityDefs = {
		num = 1;
		item[0] = {
			name = "ai_custom_imp"; // the idAI2 is referenced in here
		}
	}
	conductorEntityAIType = "SPAWN_AI_TYPE_ANY";
	initialEntityDefs = {
		num = 0;
	}
	spawnEditable = {
		spawnAt = "";
		copyTargets = false;
		additionalTargets = {
			num = 0;
		}
		overwriteTraversalFlags = true;
		traversalClassFlags = "CLASS_A";
		combatHintClass = "CLASS_ALL";
		spawnAnim = "";
		aiStateOverride = "AIOVERRIDE_TELEPORT"; // this specifies that this ai will spawn in
alongside the red teleport FX you see
		initialTargetOverride = "";
	}
	portal = "";
```

```
	targetSpawnParent = "";
	disablePooling = false;
	spawnOrientation = { // this is where the ai will face when it's spawned. it is ignored if
the ai is teleported in.
		mat = {
			mat[0] = {
				x = -1.000000;
				y = -0.000000;
				z = -0.000000;
			}
			mat[1] = {
				x = 0.000000;
				y = -1.000000;
				z=0.000000;
			}
			mat[2] = {
				x = -0.000000;
				y = 0.000000;
				z = 1.000000;
			}
		}
	}
	spawnPosition = { // this is the spawn position of the ai
		x = -30.209999;
		y = -11.000000;
		z = -10;
	}
}
}
}
```

Everything else in the target spawn can, and should, be ignored.

You can use the same idAI2 for as many target spawns as you want. For example I can very much have every single new spawn point I make that I want to support an imp all use "ai_custom_imp".

This is how idAI2s and idTarget_Spawns work with each other. The idAI2 is just an entity for a specific demon that target spawns will take and place wherever you want, however many times you want.

Before going into encounter editing, we ought to go over two more entities that use idTarget_Spawns, idTargetSpawnGroups and idTarget_Spawn_Parent, and how they are used in an encounter as well.

After that, we will go over incorporating all of this into an encounter in the next page, including how to obtain usable positions for our custom spawn points.

idTargetSpawnGroups

Earlier, you may have noticed that I said target spawns must reference an idAI2 in one of two ways, either by itself (which you have been given an example of above) or by being part of a spawn group that references a spawn parent that references idAI2s.

Spawn groups are special entities that are a set of multiple target spawns. On their own, they may sound useless (why bother when I want to spawn individual demons in?) but they are crucial for things like maintained AI (which you must have for respawning fodder) or if you want to just spawn many demons at once without having a whole lot of spawnSingleAIs.

Here is an example of an idTargetSpawnGroup below:

```
entity {
	entityDef custom_spawn_group_1 {
	inherit = "encounter/spawn_group/zone";
	class = "idTargetSpawnGroup";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = false;
	disableAIPooling = false;
	edit = {
		spawnPosition = {
			x = 0;
			y = 0;
			z = -1000;
		}
		renderModelInfo = {
			model = NULL;
		}
		clipModelInfo = {
			clipModelName = NULL;
		}
		spawners = {
			num = 4;
			item[0] = "custom_ai_spawn_1";
			item[1] = "custom_ai_spawn_2";
			item[2] = "custom_ai_spawn_3";
```

```
		item[3] = "custom_ai_spawn_4";
	}
	targetSpawnParent = "custom_spawn_parent";
}
}
}
```

Individual target spawns in these groups may have zero entitydefs defined, for so long as the spawn parent contains the idAI2 you wish to use.

What did I just mean by *that?* Let's see an example of a spawn parent:

idTarget_Spawn_Parents

Much like how a spawn group is a collection of target spawns, a spawn parent is a collection of idAI2s. Groups and parents go hand in hand to allow for more flexibility when placing down demons in an encounter.

Here is an example, pretend these idAI2s are already present in your file.

```
entity {
	entityDef custom_spawn_parent_1 {
	inherit = "encounter/spawn_group/parent";
	class = "idTarget_Spawn_Parent";
	expandInheritance = false;
	poolCount = 0;
	poolGranularity = 2;
	networkReplicated = false;
	disableAIPooling = false;
	edit = {
		flags = {
			noFlood = true;
		}
		spawnConditions = {
			maxCount = 0;
			reuseDelaySec = 0;
			doBoundsTest = false;
			boundsTestType = "BOUNDSTEST_NONE";
			fovCheck = 0;
			minDistance = 0;
			maxDistance = 0;
```

```
		neighborSpawnerDistance = -1;
		LOS_Test = "LOS_NONE";
		playerToTest = "PLAYER_SP";
		conditionProxy = "";
	}
	spawnEditableShared = {
		groupName = "";
		deathTrigger = "";
		coverRadius = 0;
		maxEnemyCoverDistance = 0;
	}
	entityDefs = {
		num = 2;
		item[0] = {
			name = "ai_custom_imp";
		}
		item[1] = {
			name = "ai_custom_pinky";
		}
	}
	conductorEntityAIType = "SPAWN_AI_TYPE_ANY";
	initialEntityDefs = {
		num = 0;
	}
	spawnPosition = {
		x = 17.0000057;
		y = 1209.99939;
		z = 185.500961;
	}
	targets = {
		num = 2;
		item[0] = "ai_custom_imp";
		item[1] = "ai_custom_pinky";
	}
}
}
```

When I say flexibility, I mean that when I use these two, I can change the ai spawned in an encounter on the fly without having to go into their individual spawn points and change the linked idAI2.

Instead, hypothetically, every spawn point can be part of a spawn group that is linked to a spawn parent that contains the idAI2s of every ai supported in the level (if DLC ai is not supported in the level, adding a DLC idAI2 into the file will not make it work on its own.)

You can reference the same spawn parent multiple times, so I use a "master parent" that does indeed contain every supported idAI2 in the specific level I'm working on at the time.

When you're spawning in individual demons in an encounter, you may choose to forgo putting in specific idAI2s in your spawn points in favor of this system if you so choose, but it is mandatory when having respawning enemies or spawning multiple enemies at once in its own specialized function rather than having multiple spawnSingleAIs.

Next step, we will bring our knowledge of idAI2s, idTargetSpawns, idTargetSpawnGroups, and idTarget_Spawn_Parents to finally be able to edit an encounter in a way you want to.

Section 2: Editing an encounter: Replacing

Most people who want to make level mods, myself included, first got to the "let's open the .entities file" part of the process, saw the unorganized chaos it presented, and backed out.

With our pre-requisite knowledge, perhaps now it won't seem as overwhelming.

We're going to begin editing our first encounter by replacing one single enemy within.

Extract the .entities file of the level of your choice and get all your things in order - having a backup file, knowing where the file to extract them from is alongside which version of the file is the correct one, etc.

Do the music change text to make sure you have everything setup, and refer back to section 1 if you have any difficulties.

Load in the level itself and head to the encounter you wish to edit. This is where meathook will begin to shine as an important level modding tool. First, enable 'notarget' in the console to make life easier, then type in 'mh_active_encounter' to grab the name of the encounter itself. CTRL+F the name of said encounter in your .entities file.

If you know how the specific encounter you've chosen works, it should be easy to line up what you see in game with what the listed steps are in the file.

Now, you might think it's as easy as simply replacing, say, ENCOUNTER_SPAWN_ZOMBIE_TIER_1 with ENCOUNTER_SPAWN_IMP, and it is. Mostly.

But don't forget. The eventdef that spawns a demon not only has the type of demon to spawn, but also which spawn point to use. Refer to the previous section. If you use a spawn point that doesn't support the idAI2 you want it to spawn, the game will crash.

Luckily, this problem is easily solvable, simply add the idAI2 of your desired demon type into the spawn point. The name of an existing idAI2 can be simply found elsewhere in the file, or one can be made by you.

For more documentation on types of eventdefs, please refer to the Event Calls section of this wiki.

Section 2: Editing an encounter: Adding

I'm going to be frank, I think ProdeusDoom does a much better job of explaining it, check it out here:

https://www.youtube.com/watch?v=HgzoFr9Zaf8