Use the Particle Editor to Create a Flamethrower Ability (that burns enemies and starts fires in the world)

by Ross Carey - C26

Overview

This tutorial will teach you the basics of using the particle editor as well as integrating those particles into your level. It also includes setting the fire up to damage enemies, ignite env_fires and limiting the ability to a mana count. This tutorial's primary purpose is in teaching the particle editor, the sections after the implementation of the particles are all extra pieces to make it feel better and teach how other entities work.

The particle editor can be very confusing so be sure to pay close attention to the sections on using it and for more information about it, see the valve developer entry. (https://developer.valvesoftware.com/wiki/Particle_System)

Set Up

Flamethrower Entities:

Bonus Entities:

Entity Type	Entity Name (If Applicable)
☐ info_player_start	N/A
☐ logic_auto	N/A
☐ info_particle_system	Flamethrower_ParticleSystem
☐ env_entity_maker	Flamethrower_Spawner
☐ point_template	Flamethrower_Template
☐ game_ui	KeyBind
☐ logic_relay	Flamethrower_Relay
☐ logic_relay	FlamethrowerOff_Relay
☐ Trigger_Hurt (Special Entity created from a BSP brush)	Trigger_Fire
☐ npc_zombie	N/A

Entity Type	Entity Name (If Applicable)
☐ math_counter	ManaAmount
☐ logic_timer	Mana_FlamethrowerDrain
☐ logic_timer	Mana_Regen
☐ env_fire	N/A
☐ env_firesource	Flamethrower_FireSource

Creating the Flamethrower

Set Up the Scene

1. Create a Room to contain the player

First step is to create a room to play round in so that we can spawn a player and have a space to move around in and see the flamethrower at work.

- ☐ Using the Block Tool, create a BSP block (512x512x512)
- ☐ With the selection tool, right click that BSP in the wireframe view and select make hollow
- ☐ Enter -32 and select OK

This creates a block and makes it hollow with walls going out 32 units. This method quickly gives you a floor, walls and ceiling.

2. Add a Player Start to your room

To actually test out anything we are going to be doing, we're going to need a player start.

- ☐ Using the Entity Tool, create an info_player_start at any location inside the BSP room

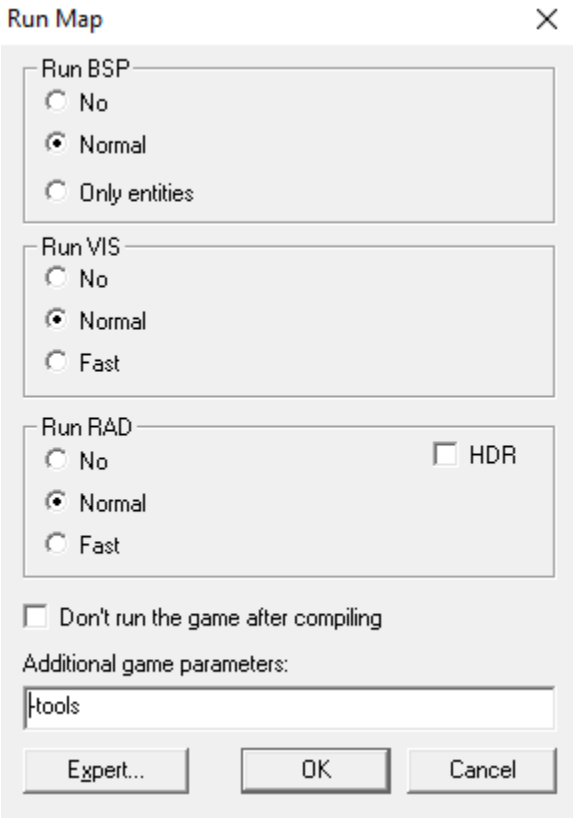
To the Particle Editor

1. How To Open the Editor

The particle editor is not part of Hammer itself. To get to this new editor, you have to launch the game with a special parameter.

- ☐

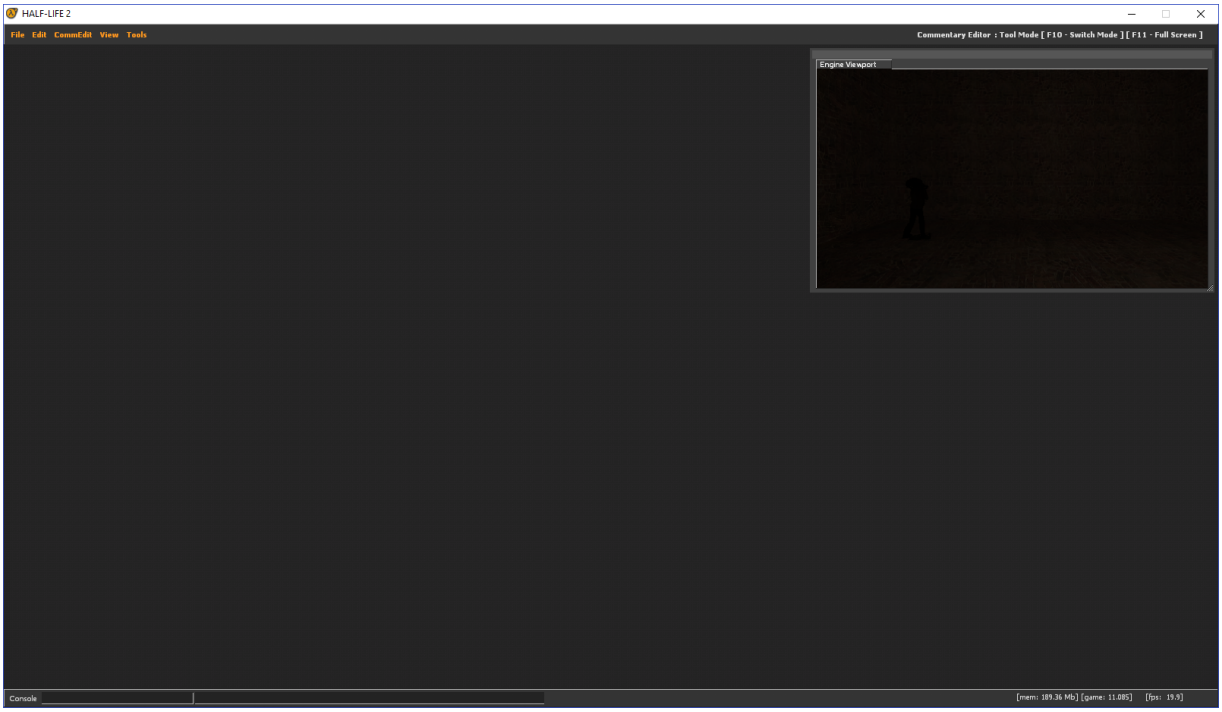
- [ ] Press the Run Map! button (or press F9) to bring up the run map dialogue
- [ ] In the additional game parameters section at the bottom, type in -tools



If you have nothing in that box already, it can be the only thing. If you have other parameters, just add it in at the end.

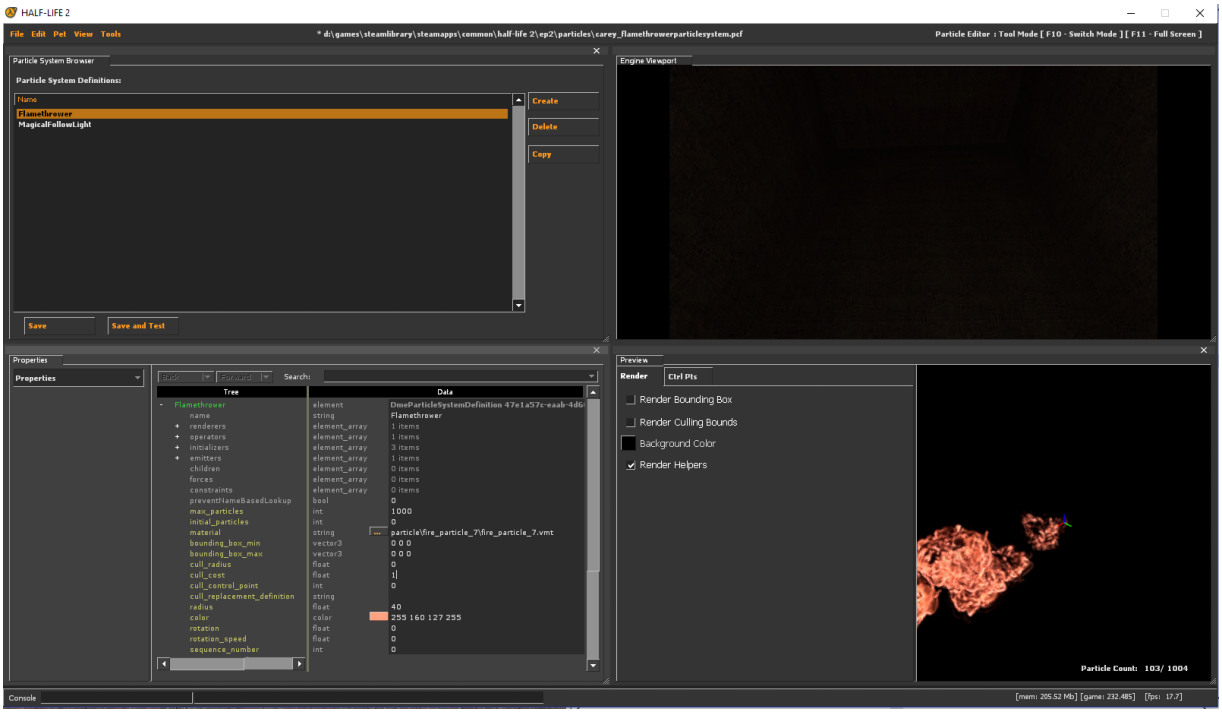
2. Intro to the Editor

When the game loads, you will see a mostly blank screen with a viewport in the top right and some options in the top left. This is the tools window, which includes the particle editor that we are looking for.



- [ ] To get to the particle editor, click on Tools -> Particle Editor.

This switches the mode to the particle editor where you can create and edit particle effects.



- [ ] To start creating your new particle, click on the File button in the top left corner and select new. This will open up the full particle editor and give you four different panels.

The top left panel contains the particles inside of the file you just created. One particle file can hold a number of different particle effects inside of it. For example, my project used two completely different particle systems that were saved into the same file.

The top right panel is the game viewport. Here the game is running and it allows you to quickly load your particle changes into game and see them running. Pressing F10 switches you from editor mode to game mode. Pressing the Save and Test button in the top right panel forces the level to restart with your changes so you can start playing and see the particles in your running level.

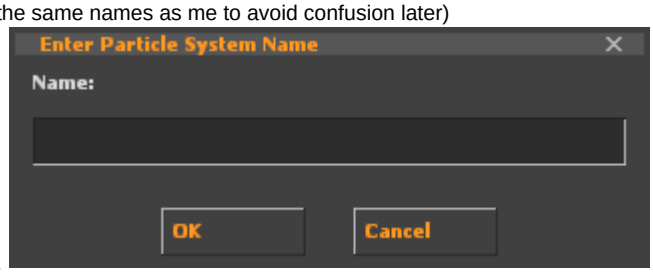
The bottom right panel contains the preview window. This automatically plays the particle effect whenever a change is made to it, allowing you to see changes in real time.

The bottom left panel is where you will spend most of your time. This panel is where all of the particle properties are. Here you can add new properties and change the values of existing properties to create the effect that you want.

3. Creating the Particle

Initial Properties

Focus on the top left panel first. Here we can press the create button to start creating our first particle system. The editor will now prompt you to give your system a name. I named mine Flamethrower. (You don't have to name yours this, but it might help you to use the same names as me to avoid confusion later)



.

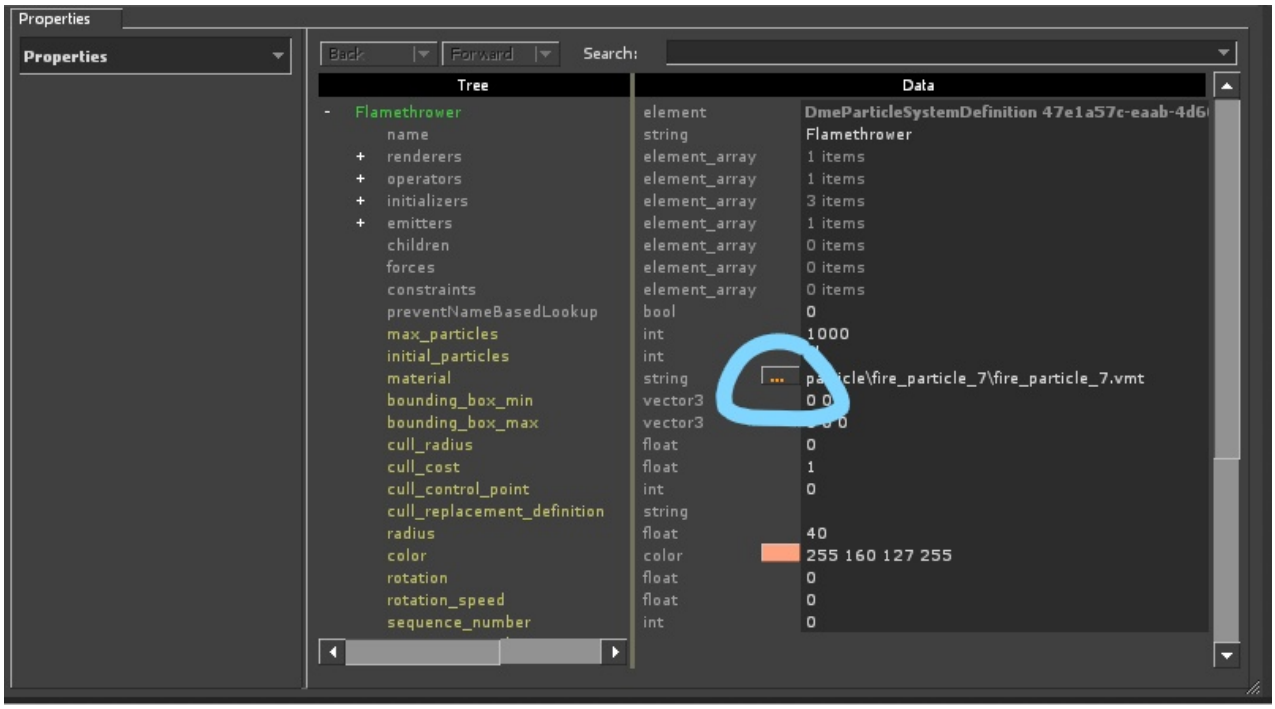
With the particle created, we can select it from the list. This populates the bottom left panel with property information that we can begin to edit.

(For the sake of this tutorial I will only be discussing the values I changed. There are a lot more values that could be changed and properties that could be added. I recommend trying them out and seeing what they do in the preview window.)

The default property is the properties section itself. This contains a number of basic properties that is consistent across all particle systems such as the size and material of the particles, to the number of particles.

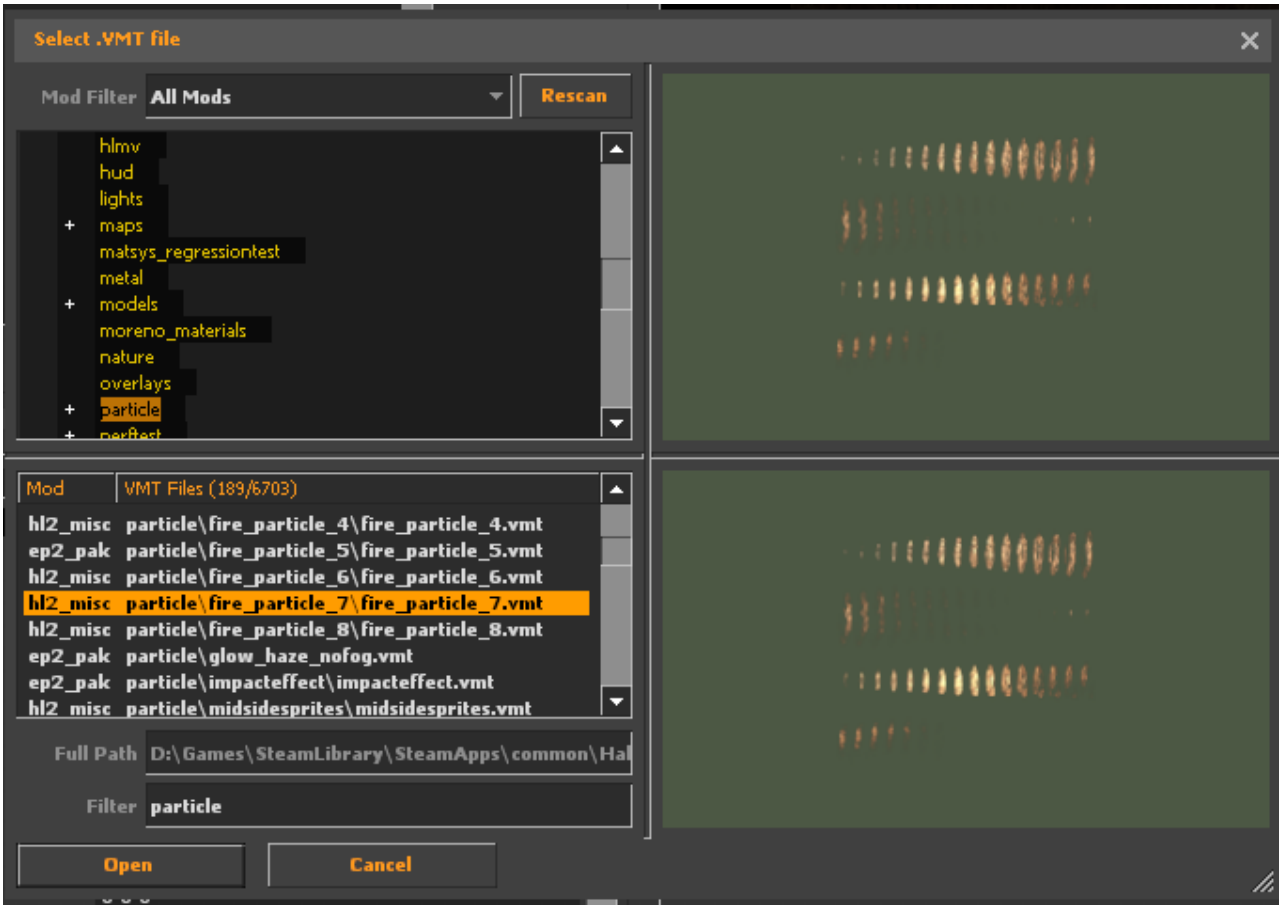
First we need to give our particle a material so that when it spawns it will be rendered out as a sprite that we can see.

- [ ] On the material line, click on the three dots



This opens up the material browser. Here you can see a list of materials that can be applied to the particles. The top left panel is a list of folders for you to search through by category (the default being particles, so you shouldn't have to touch it), and the bottom left panel being the list of materials inside that folder. Find the particle you want in there and either double click on it or click it and press OK.

- [ ] For this example choose fire_particle_7



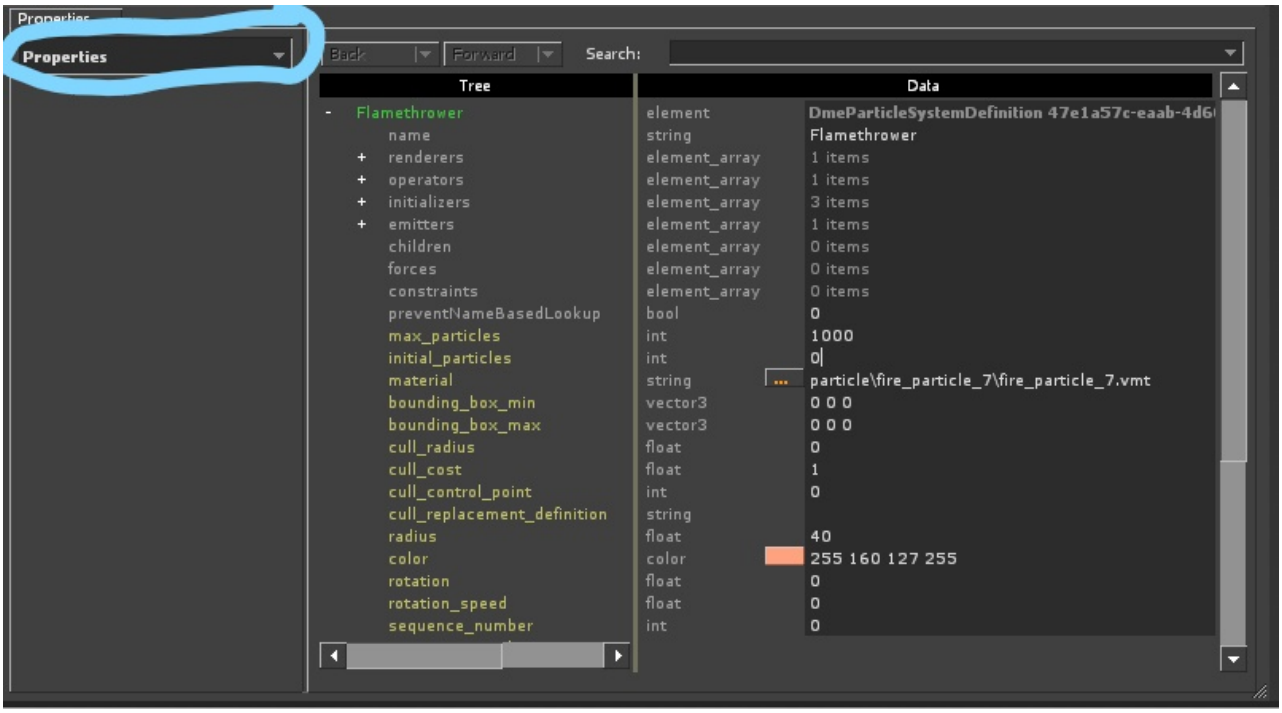
The last two values we are changing in the properties are the radius and color. The radius is the radius of the particle, so the size each

individual sprite will be once it is spawned and the color is a color change to the particle. The four numbers in the color are the RGB values plus an alpha value at the end. They are on a 0 to 255 scale.

- [ ] Change the radius to 40
- [ ] Change the color to 255 160 127 255 (Alternatively, choose a color by clicking on the color swatch)

Adding and Editing New Properties

Now that we have a fire particle, let's start to add custom effects to the system to make it spawn at a point and move away from it in a stream like a flamethrower. I'm going to take you through the different categories of properties we can add one at a time. Staying in the properties panel, click on the drop down menu in the top left corner. This will open up a list of different categories for us to edit and add. We'll start off with the renderer.

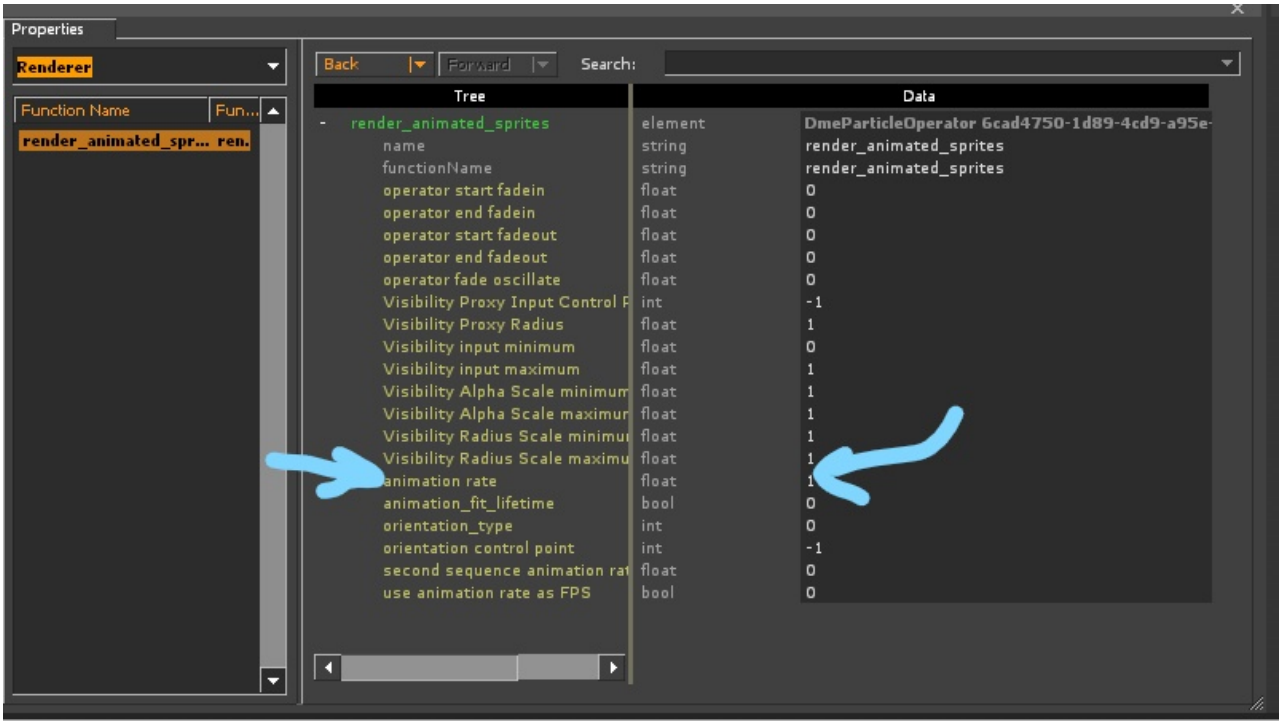


Directly under the new renderer tab is an empty box that says "No Particle Functions". This means that the system does not have any renderer functions on it yet. We're going to add one.

- [ ] Right Click inside that empty box and select Add
- [ ] Choose the render_animated_sprites option

This tells the system that we have an animated sprite and that it should run its animation. In this case it is in the form of a sprite sheet. Having this property automatically tells the particle to animate even if we don't change any default values. The default animation rate is very slow however, so let's speed it up a bit.

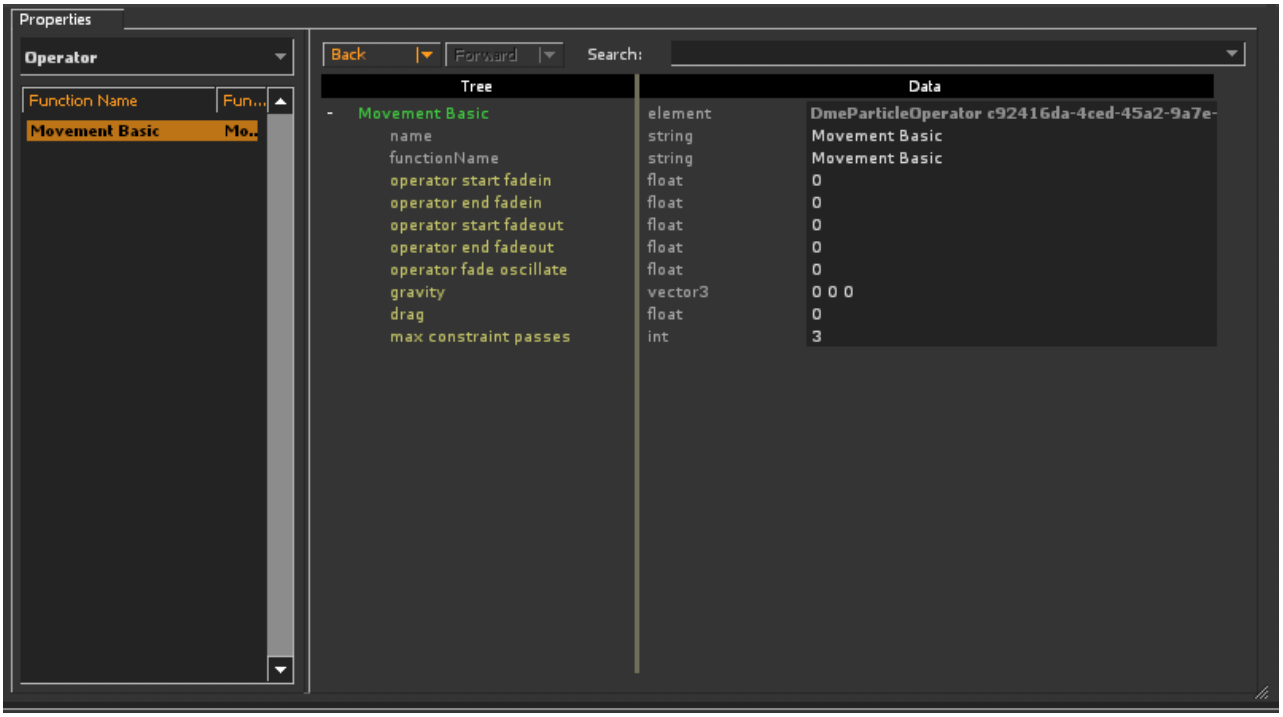
- [ ] Change the animation rate to 1



Now change from renderer properties to operator. This contains a large number of functions for how the particle should operate over its lifespan. We'll just add a property to make our particles move.

- [ ] Add the Movement Basic property (Use the same method as adding the render_animated_sprites property above)

This property again does everything we need it to by default. No further changes are needed.



Next up is the initializer. Here we can add properties that add variance to our particles upon spawning to make it not look like it's repeating too much and feels more natural. To do that we are going to:

- [ ] Add the Position Within Sphere Random property
- [ ] Add the Rotation Random property

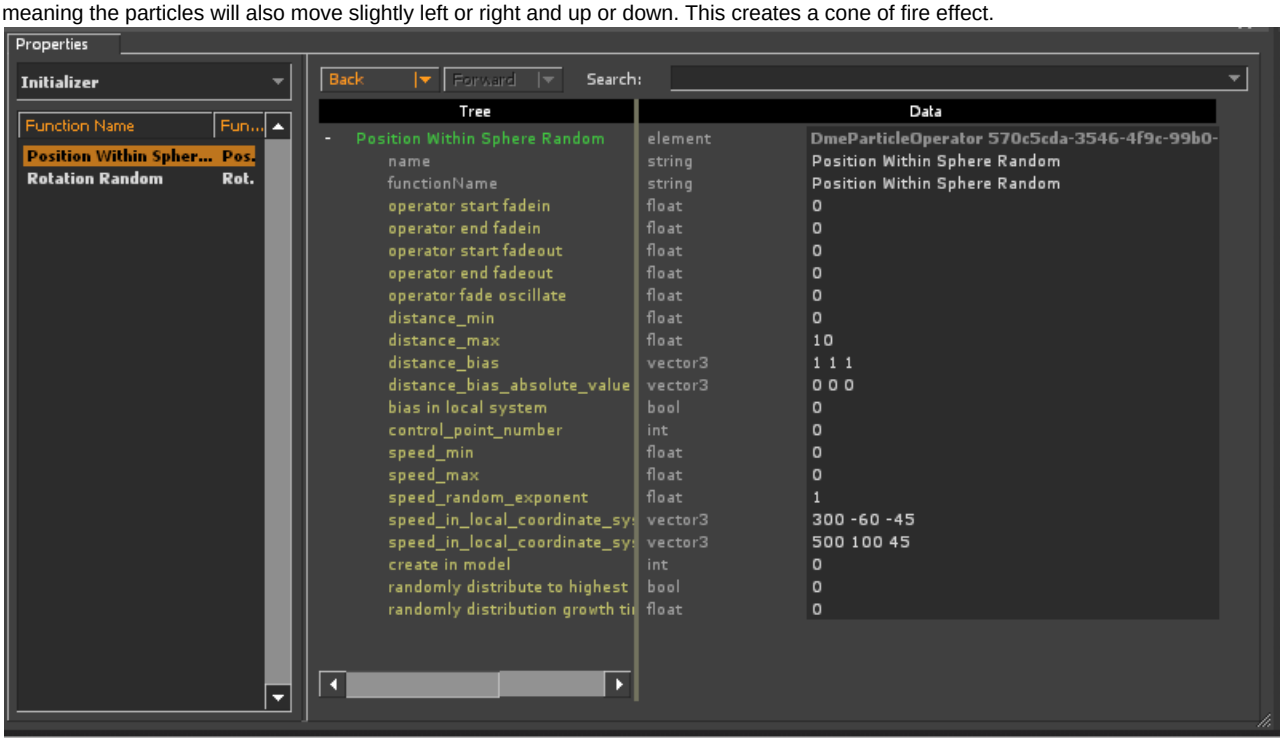
Both of these properties are self-explanatory, but basically they spawn your particles inside of a sphere that you define and give that a

random rotation. We will be editing these properties to get the best effect for a flamethrower.

First is the Position Within Sphere Random. We want to spawn the particles within a small sphere to add slight variance to the spawning and make the particles not stack on top of each other. This property also allows up to give the particles local speeds. This means that if I give it a speed in the x direction, it will move in it's local x or forward from wherever the emitter is.

- [ ] Change the distance_max to 10
- [ ] Change the speed_in_local_coordinate_system_min to 300 -60 -45
- [ ] Change the speed_in_local_coordinate_system_max to 500 100 45

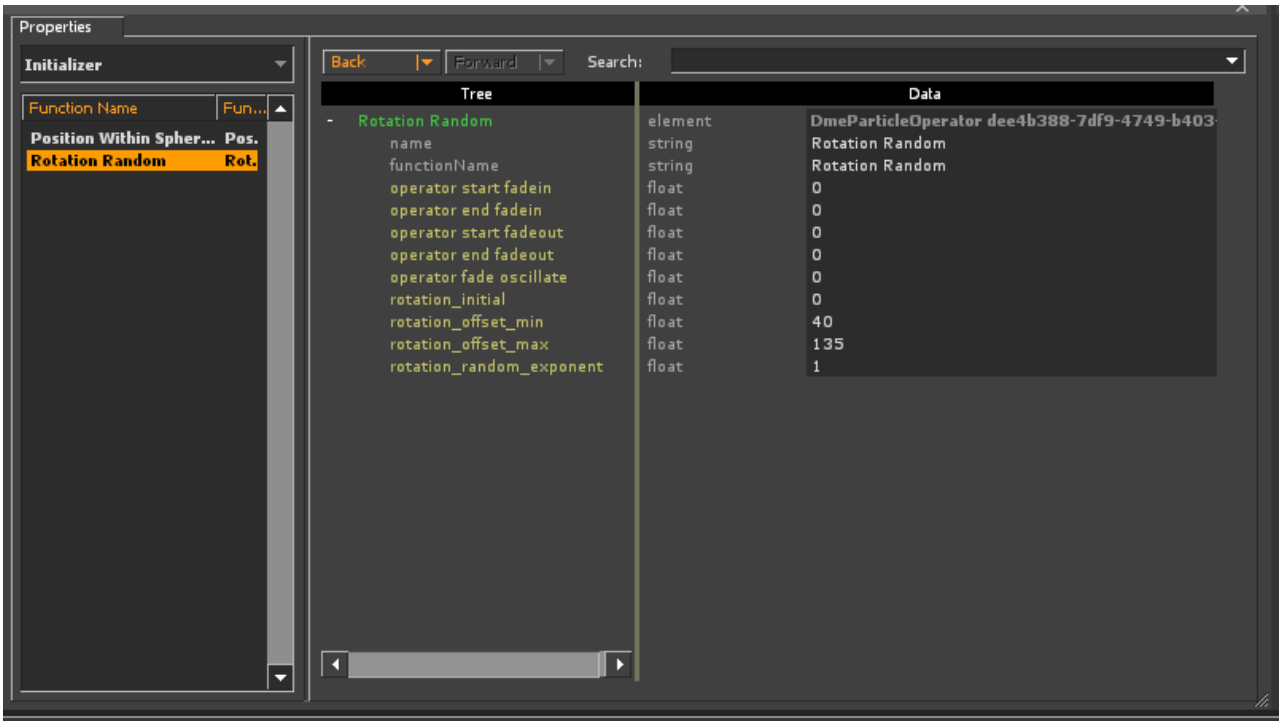
These speeds give it a range of speeds moving quickly forward for the x direction. The Y and Z range from positive to negative, meaning the particles will also move slightly left or right and up or down. This creates a cone of fire effect.



The rotation property rotates the particles left or right so that they don't look too consistent as they move away from the player. It is another way to add variance to the appearance of the particles.

- [ ] Change the rotation_offset_min to 40
- [ ] Change the rotation_offset_max to 135

This tells the particles to change their rotation to somewhere between 40 and 135 degrees when they spawn.

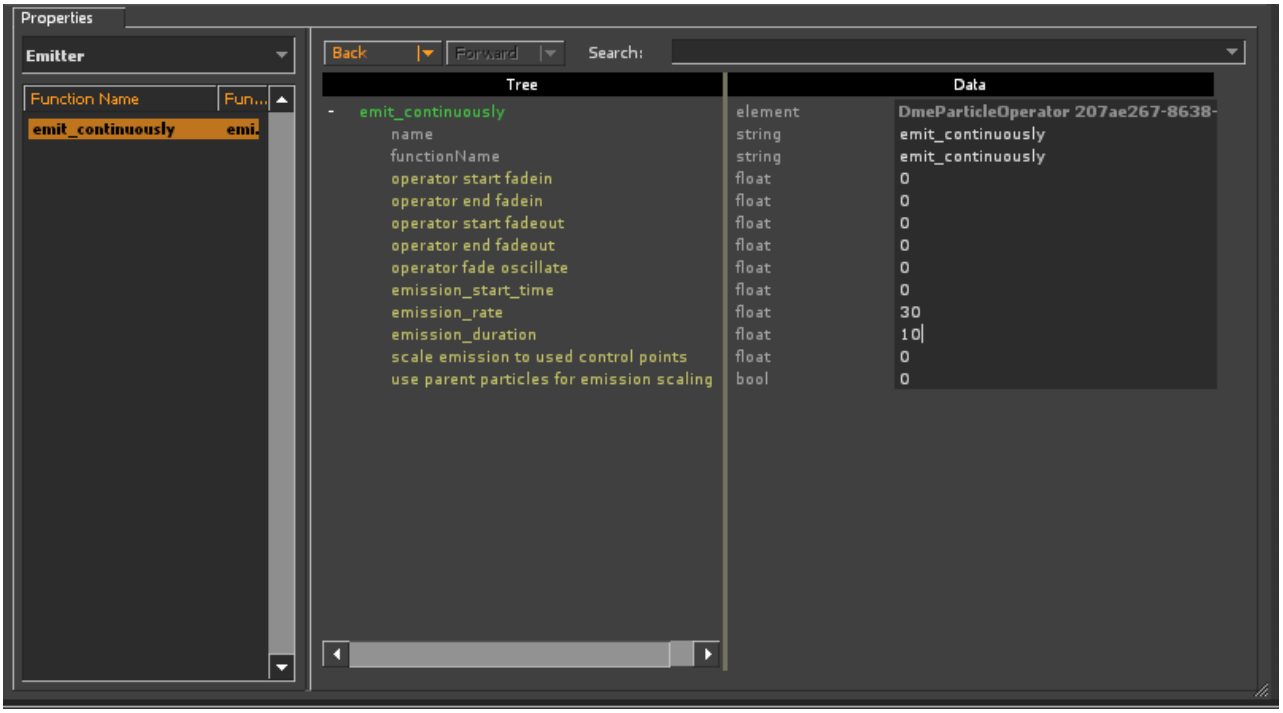


The last particle property we need to edit is the emitter. This category controls the actual creation of the particle. Once we add a property here, your fire particles will start spawning in the preview panel.

- [ ] Add the emit_continuously property

This causes the emitter to constantly spawn particles until it reaches a desired point or is turned off. We want to change the rate so it's not spawning so quickly and limit it with a duration so it has an upper limit of time it will emit.

- [ ] Change the emission_rate to 30
- [ ] Change the emission_duration to 10

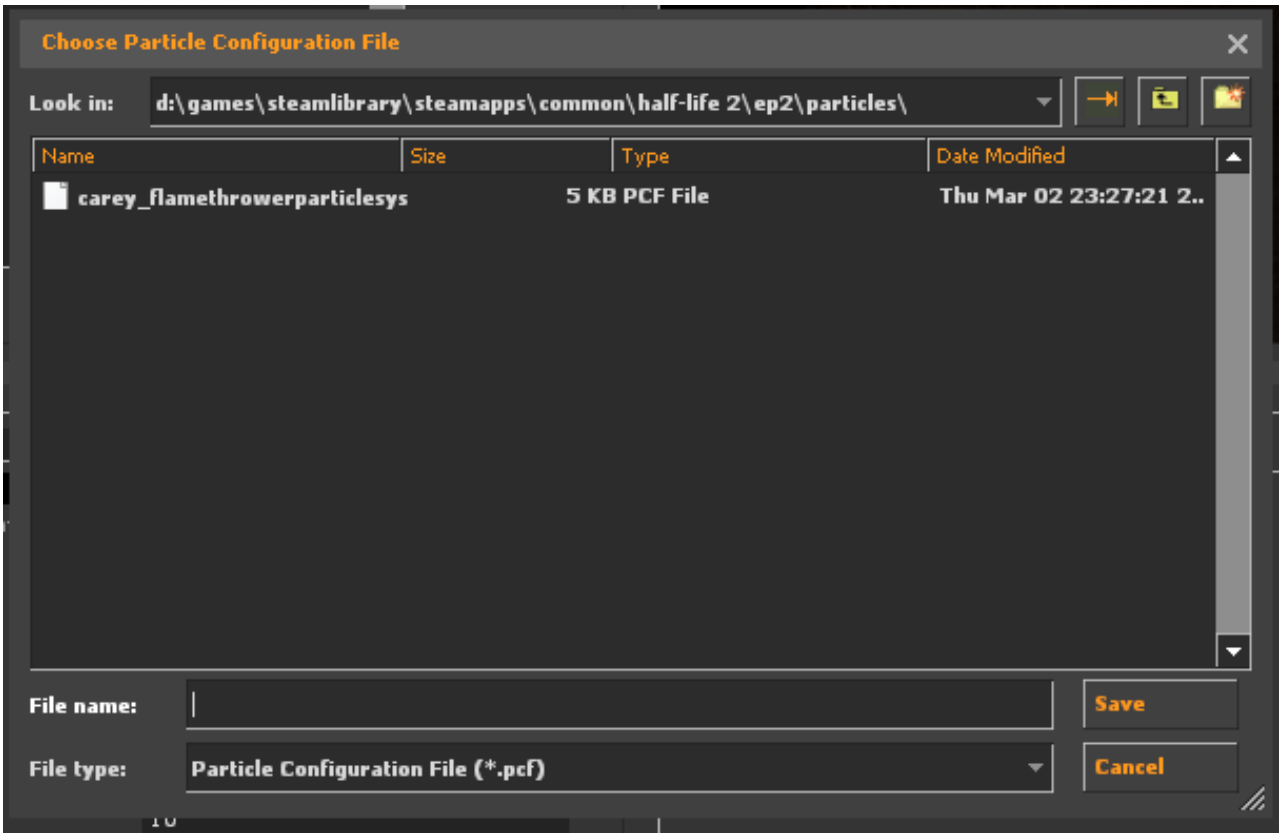


Congratulations! Your particle should be playing in the preview panel and you can see your awesome flamethrower particle at

4. Saving the Particle and Setting Up the Particle Manifest

Now that we've created our flamethrower particle, we have to save it out as a .pcf (Particle Configuration File) so that Half-Life can find it and use it in game.

- [ ] Click on File -> Save As and name your file (I named mine carey_flamethrowerparticlesystem)
- [ ] Make sure the file is saved in half-life2\ep2\particles (Create that folder there if it doesn't exist)



Now for the most confusing (but crucial) part, so bear with me if it doesn't make sense on your first attempt.

The way Half-Life finds these particle files and loads them into the game is with a text document called the particle manifest. Navigate to the following folder inside the folder you install your Steam games to: SteamApps/common/sourcesdk_content/ep2/particles.

If the sourcesdk_content folder is not there, download and launch the source SDK from the tools tab of Steam. When it launches it should download some files, including this folder. We are going to copy the particle manifest out of this folder and into a custom folder inside our Half-Life 2: Episode 2 folder.

- [ ] Make a copy of the particles_manifest.txt
- [ ] Save it in your copy clipboard or somewhere easily accessible

DATA (D:) › Games › SteamLibrary › SteamApps › common › sourcesdk_content › ep2 › particles

Name	Date modified	Type	Size
bonfire.pcf	1/23/2017 9:58 PM	PCF File	9 KB
building_explosion.pcf	1/23/2017 9:58 PM	PCF File	4 KB
choreo_dog_v_strider.pcf	1/23/2017 9:58 PM	PCF File	103 KB
choreo_extract.pcf	1/23/2017 9:58 PM	PCF File	16 KB
default.pcf	1/23/2017 9:58 PM	PCF File	2 KB
demo_particle_light.pcf	1/23/2017 9:58 PM	PCF File	7 KB
devtest.pcf	1/23/2017 9:58 PM	PCF File	41 KB
door_explosion.pcf	1/23/2017 9:58 PM	PCF File	12 KB
dust_bombdrop.pcf	1/23/2017 9:58 PM	PCF File	3 KB
dust_rumble.pcf	1/23/2017 9:58 PM	PCF File	68 KB
electrical_fx.pcf	1/23/2017 9:58 PM	PCF File	36 KB
grub_blood.pcf	1/23/2017 9:58 PM	PCF File	8 KB
hunter_flechette.pcf	1/23/2017 9:58 PM	PCF File	6 KB
hunter_intro.pcf	1/23/2017 9:58 PM	PCF File	4 KB
hunter_projectile.pcf	1/23/2017 9:58 PM	PCF File	63 KB
hunter_shield_impact.pcf	1/23/2017 9:58 PM	PCF File	7 KB
particles_manifest.txt	1/23/2017 9:58 PM	Text Document	2 KB

In your Half-Life 2/ep2 folder (where we placed our particles folder previously) create a folder called Custom. Inside that custom folder we can create another folder (name is up to you, mine is called Carey_Assets). Finally, inside that folder create a folder called Particles.

- [ ] Paste your copy of the particles_manifest inside Half-Life 2/ep2/Custom/<NamedFolder>/Particles/
- [ ] Inside your copy of the particles manifest, add the name of your .pcf file as the last entry in the list

Follow the same format that all of the others use. I suggest copy and pasting an entry and changing the file name to the file name that you saved yours out as.

Make sure to keep the particles/ at the beginning.

```
        "file"          "particles/Vortigaunt_FX.pcf"
        "file"          "particles/devtest.pcf"
        "file"          "particles/electrical_fx.pcf"
        "file"          "particles/burning_fx.pcf"
        "file"          "particles/antlion_blood.pcf"
        "file"          "particles/grub_blood.pcf"
        "file"          "particles/grenade_fx.pcf"
        "file"          "particles/rocket_fx.pcf"
        "file"          "particles/impact_fx.pcf"
        "file"          "particles/carey_flamethrowerparticlesystem.pcf"
}
```

Now we need to direct Half-Life 2 to update its internal particles manifest based on your changes. It's helpful to have two explorer windows open at once for this section. Have the first one open to your custom folder that you just created (we're going to need to drag our folder (Carey_Assets in my case) onto an executable inside Half-Life's bin folder.

Navigate the second window to Half-Life 2/bin and find the file called vpk.exe

- [ ] Select and drag your created folder inside Custom (Carey_Assets in my case) onto vpk.exe

This runs the vpk.exe with your changes to the particle manifest. (The command prompt should flash on the screen as it runs)

Congratulations! Your new particle is now set up and integrated into Half-Life 2 and can show up when you create particle emitters in your level.

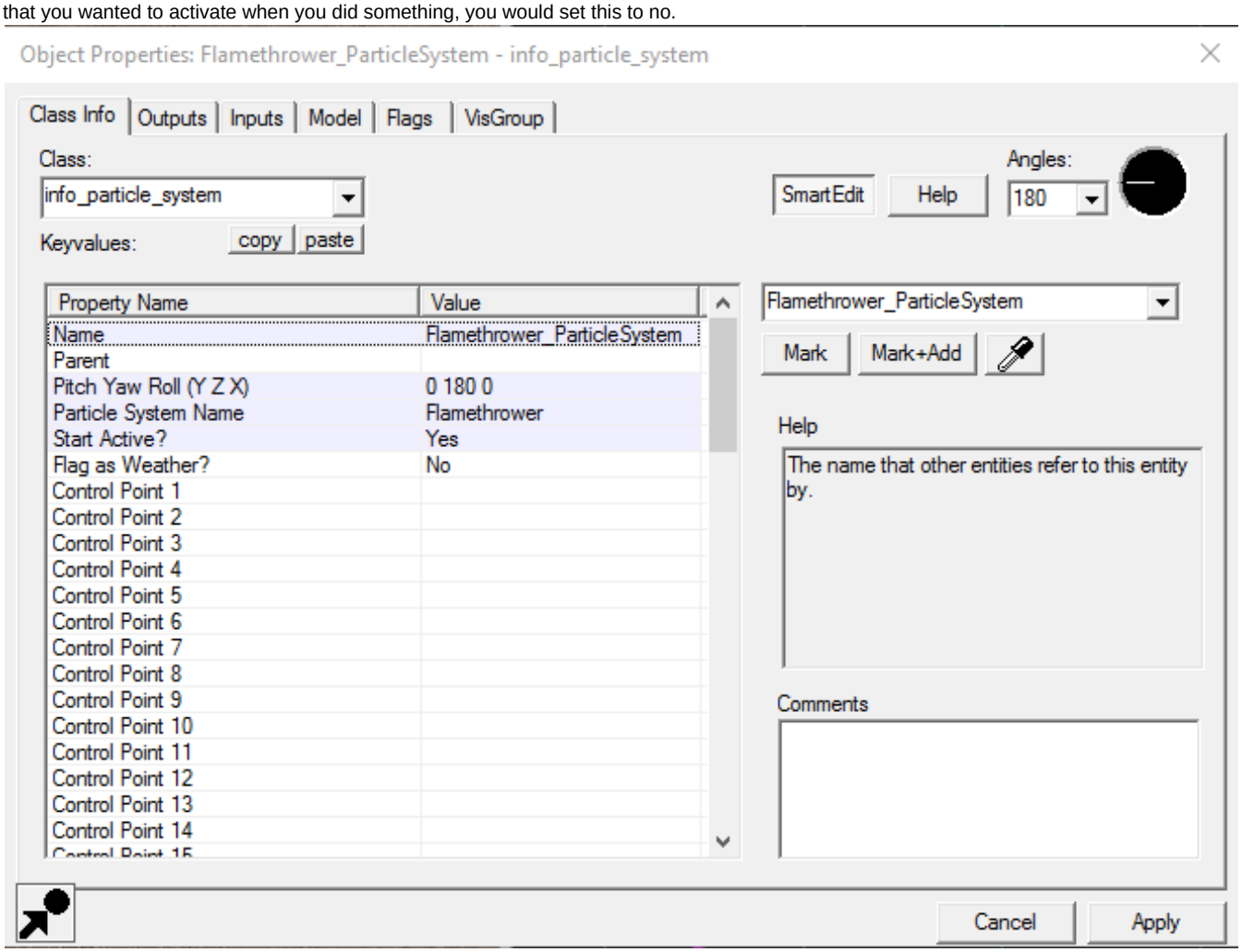
As a note: If you create new particles inside that same .pcf file, you don't need to update your particle manifest as there is no new .pcf file.

Create the Particle Emitter

Now that we have our particle created and loaded into Half-Life, we can go back into the Hammer Editor and start setting up our scene to use it. First let's place and name the entities we'll need to create and spawn the particle emitters. This will create a template that holds the particle emitter, and a spawner that is attached to the player that can spawn the template particle emitter. This way, whenever you tell the spawner to spawn your template, it will spawn right in front of the player.

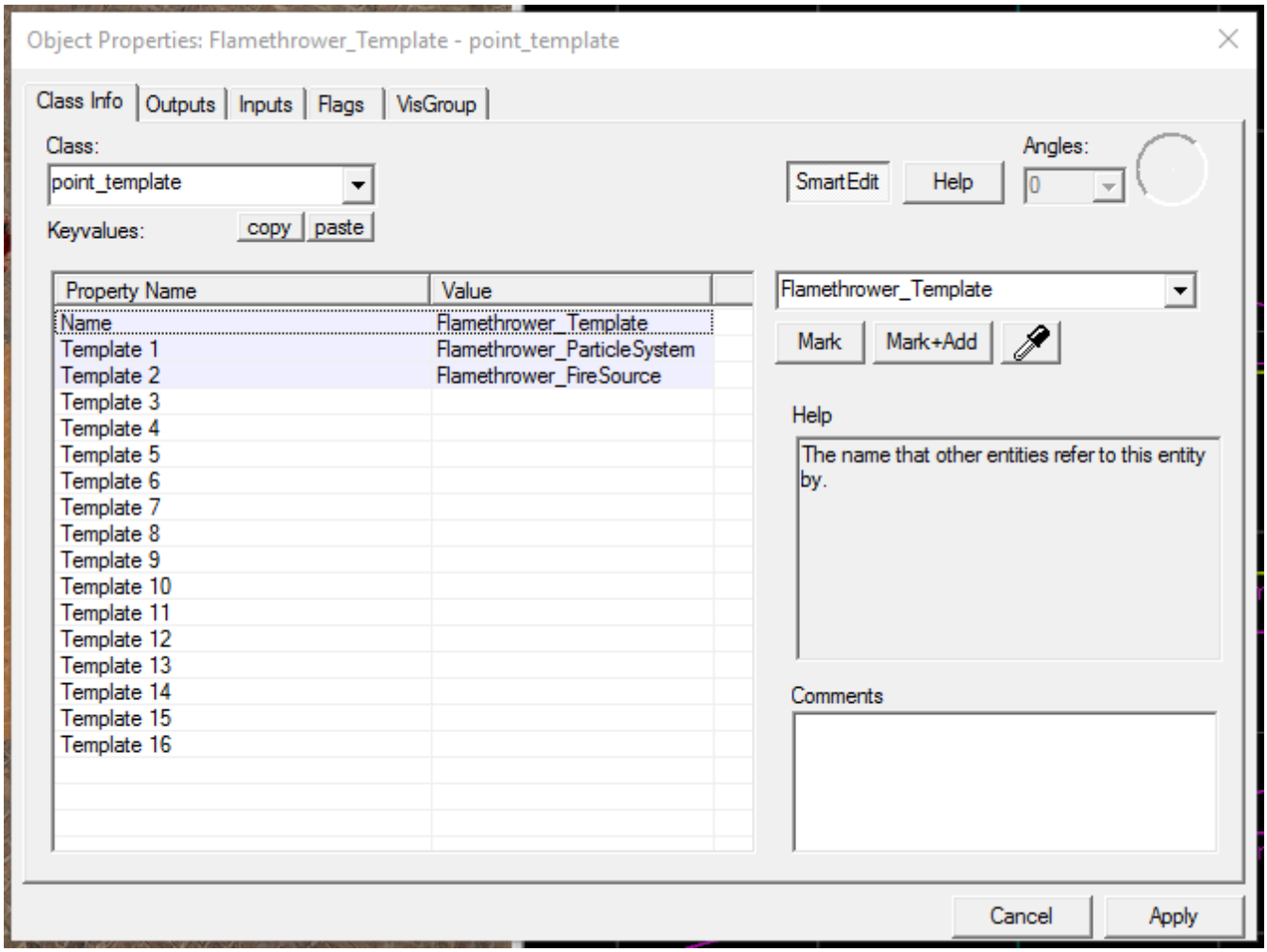
- [ ] Create an info_particle_system using the entity tool
- [ ] Open up the Object Properties window by double clicking it or pressing Alt+Enter
  - [ ] Name it: Flamethrower_ParticleSystem
  - [ ] Set the Pitch, Yaw, Roll: This is determined by where the player is facing. It should be the same direction that your info_player_start is facing
  - [ ] Set the Particle System Name to Flamethrower (This is the name of the particle you created inside your .pcf file, not the name of the .pcf file)
  - [ ] Set Start Active? to Yes

The name allows us to access it through the Input/Output system. The pitch, yaw, roll rotates it so that it is facing forwards. The Particle System Name is the name of the particle system it is going to spawn. In this instance, it is the name of the flamethrower particle we created in the particle editor. Start Active? tells the system that it should start emitting as soon as it is spawned. If you were to have emitters in the world that you wanted to activate when you did something, you would set this to no.



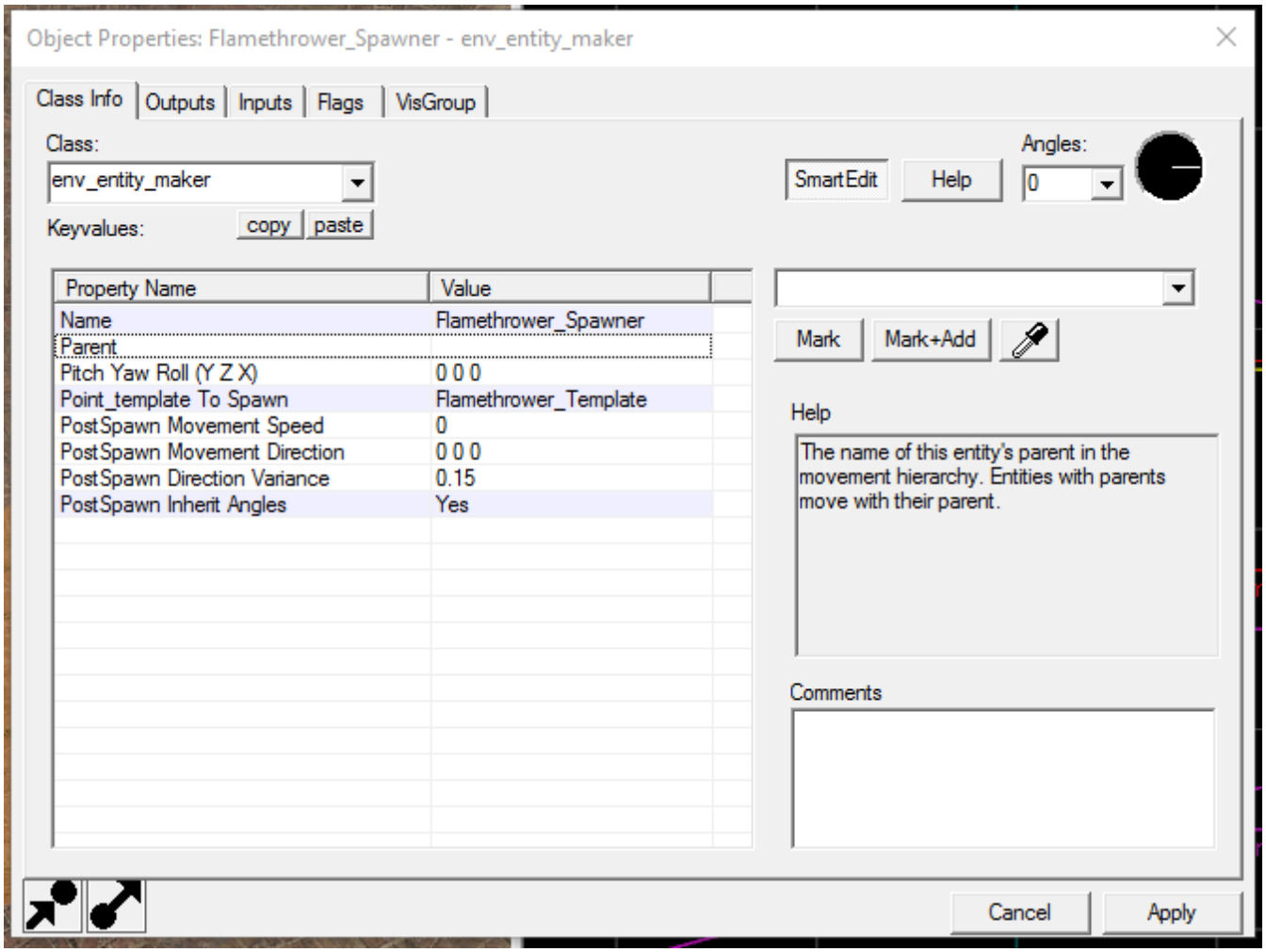
- [ ] Create a point_template using the entity tool
  - [ ] Name it Flamethrower_Template
  - [ ] Point Template 1 to your particle system (To do this either type the name into the field or click on the field and then click on the eyedropper button and click on the entity you wish to put in that field)
  - [ ] **Important: Move your particle system on top of the point template** (When your template is told to spawn, it spawns at the location of the entity maker, and any templates inside of it are spawned at their position in relation to their point template. For example if I had 2 particle systems in one template, one could be 64 units to the left and one 64 to the right and they will still be 128 units apart when the template is spawned.)

Ignore the entry in Template 2 for now. That will be set up in a later section about starting fires.



- [ ] Create an env_entity_maker using the entity tool
  - [ ] Name it Flamethrower_Spawner
  - [ ] Choose the Point_Template to Spawn as Flamethrower_Template (The point_template we just created)
  - [ ] Set PostSpawn Inherit Angles
- [ ] Place the entity_maker 48 units in front of the player, at around chest height. This makes it seem like the fire is coming out of the player.

The entity maker is what does the spawning. It tells a point template to spawn whatever is saved inside of it. The PostSpawn inherit angles tells the maker to push its own rotation down to its spawned template. For us this means that the maker (which will be attached to the player) will always be looking forward from the player, that rotation will be sent to our particle system so it too will spawn facing the player's forward.



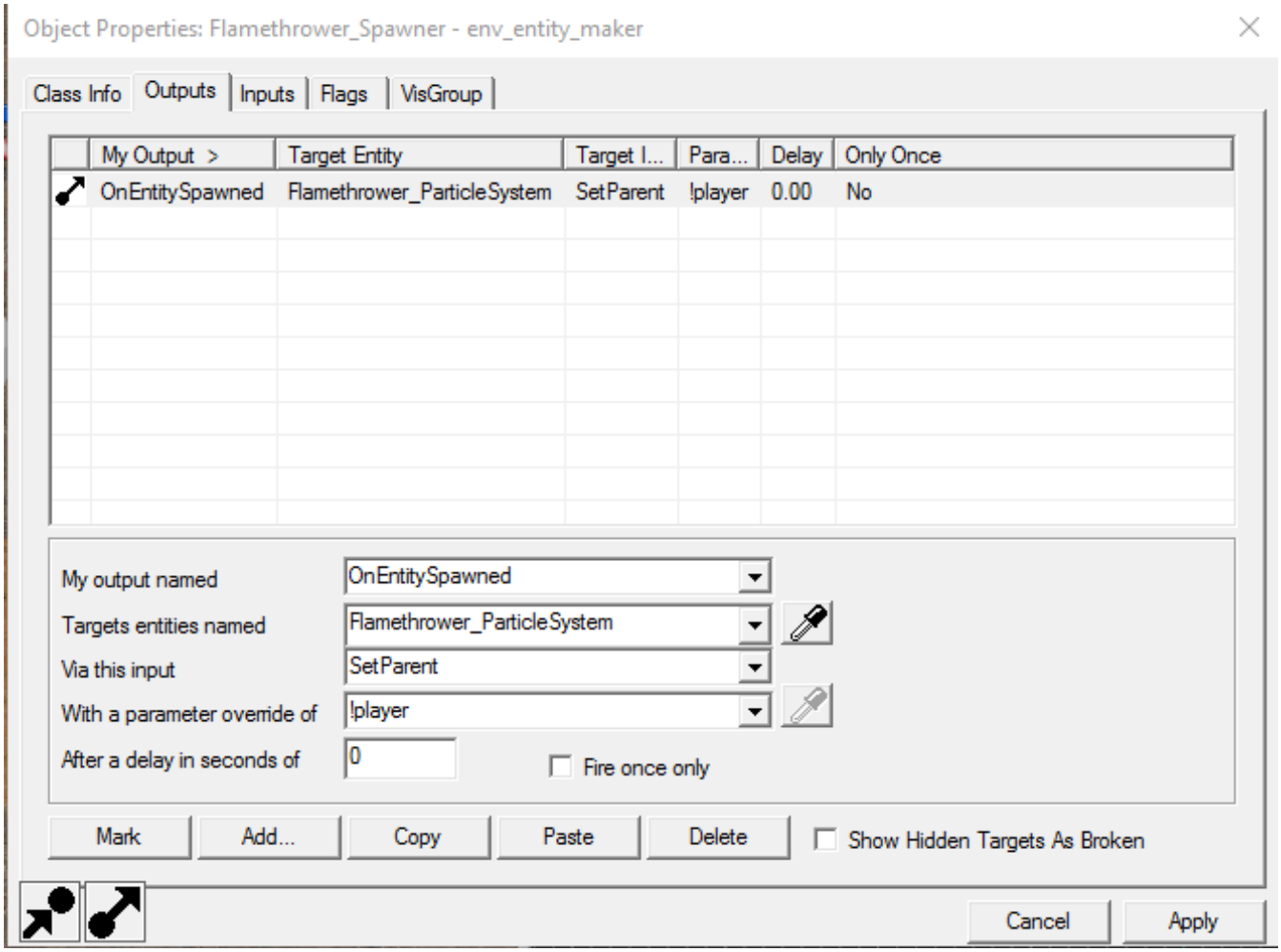
Lets attach this maker to the player now, go to the output tab of your entity_maker. To do that we need a logic_auto. This entity allows us to run scripts on the level load which lets us attach objects to the player. To do so we need the output tab. Here we can add outputs. They are scripts that are triggered on events (specific to each entity).

- [ ] Create a logic_auto using the entity tool
- [ ] In Outputs:
  - [ ] Click the Add... button
  - [ ] My output named: OnMapSpawn
  - [ ] Targets entities names: Flamethrower_Spawner
  - [ ] Via This Input: SetParent
  - [ ] Parameter Override: !player

We can do a similar operation on the entity maker to attach our created particle system to the player when it spawns.

- [ ] Click the Add... button
- [ ] On your new output change the following settings:
  - [ ] My output named: OnEntitySpawned
  - [ ] Targets entities names: Flamethrower_ParticleSystem
  - [ ] Via This Input: SetParent
  - [ ] Parameter Override: !player

This creates a new script and tells it whenever it spawns anything, find an entity called Flamethrower_ParticleSystem (Which is the name of the spawned entity in this case) and set it's parent to the player.



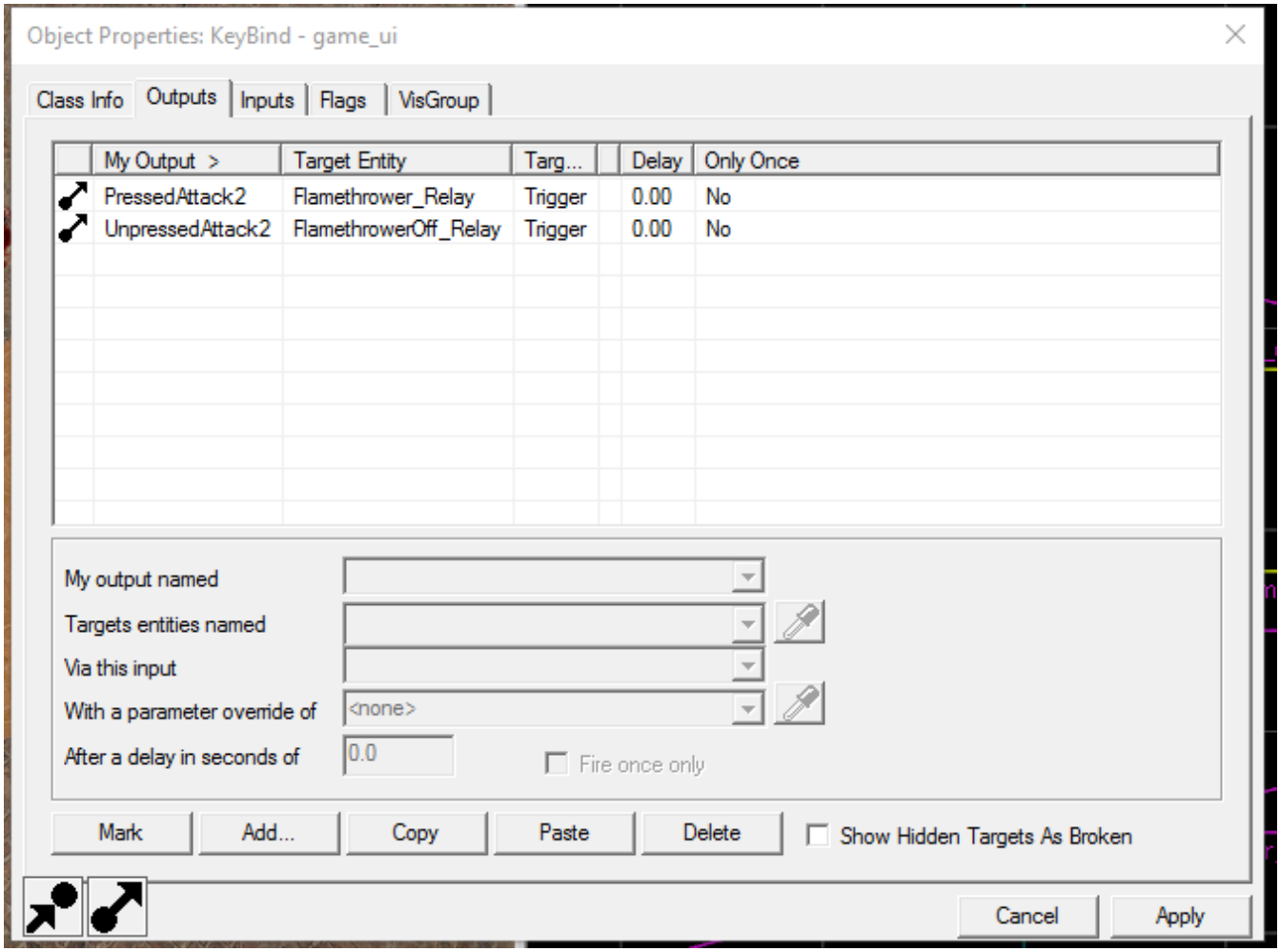
How to Control the Flamethrower

We now have a particle system that can be spawned and parented to the player from an entity maker that is parented to the player. Next step is to tell that entity maker to spawn the particle system so we can see the particles in action. To make this happen we need an entity to handle the controls and entities to turn the particle emitter on and off to start and stop the flamethrower.

- [ ] Create a game_ui using the entity tool
  - [ ] Name it KeyBind
- [ ] Create a logic_relay
  - [ ] Name it Flamethrower_Relay
- [ ] Create a second logic_relay
  - [ ] Name it FlamethrowerOff_Relay

The game_ui will be used to bind the right click to activating your flamethrower. The Logic relays are holders for inputs and outputs to help you organize your scripts. Let's set up the game_ui first:

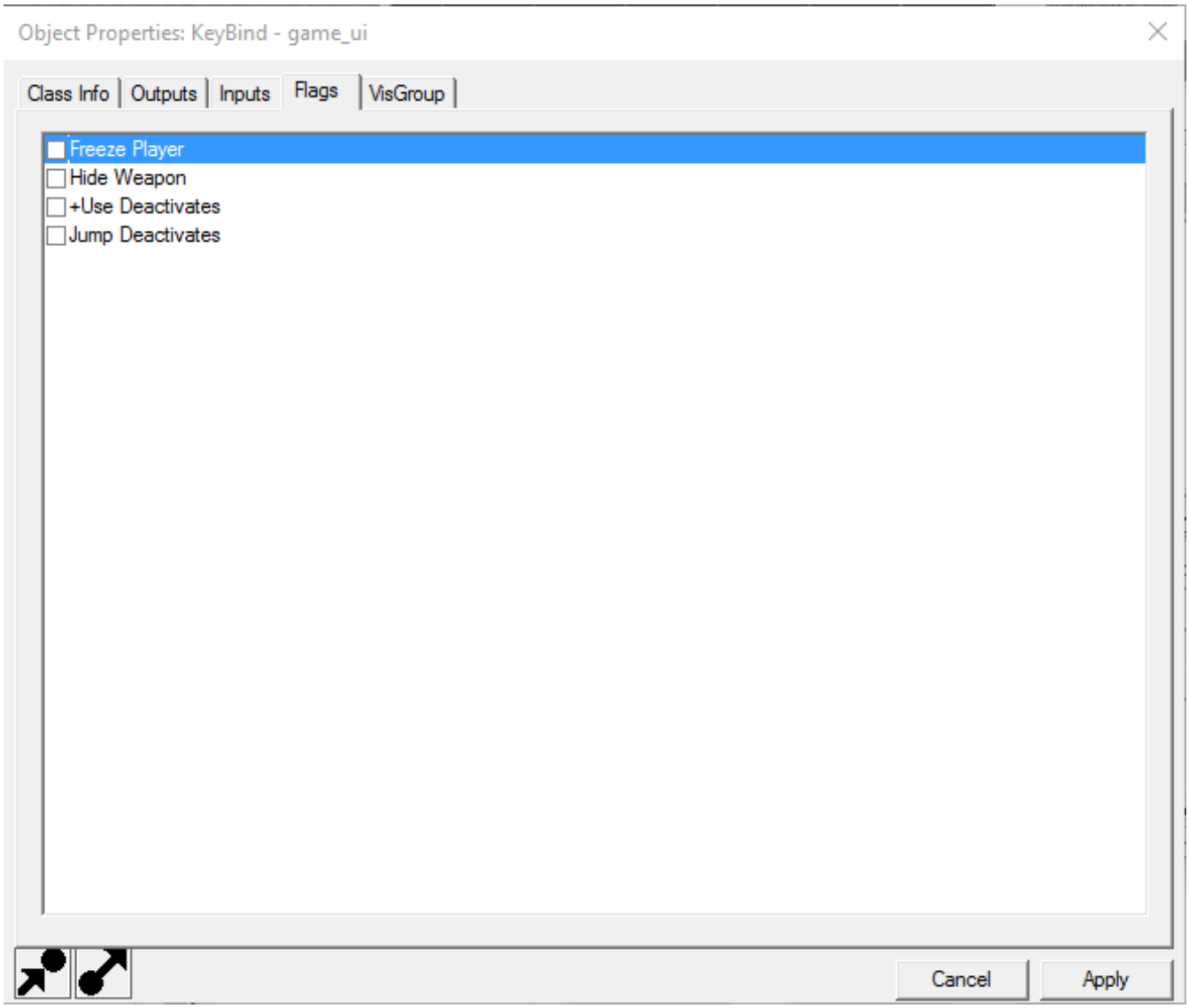
- [ ] Add...
  - [ ] Output: PressedAttack2 (This is the right click)
  - [ ] Target: Flamethrower_Relay
  - [ ] Input: Trigger
- [ ] Add...
  - [ ] Output: UnpressedAttack2
  - [ ] Target: FlamethrowerOff_Relay
  - [ ] Input: Trigger



This tells the on relay to trigger when I press the right click and the off relay to trigger when I release the right click. With that I can turn the flamethrower on and off with the right click.

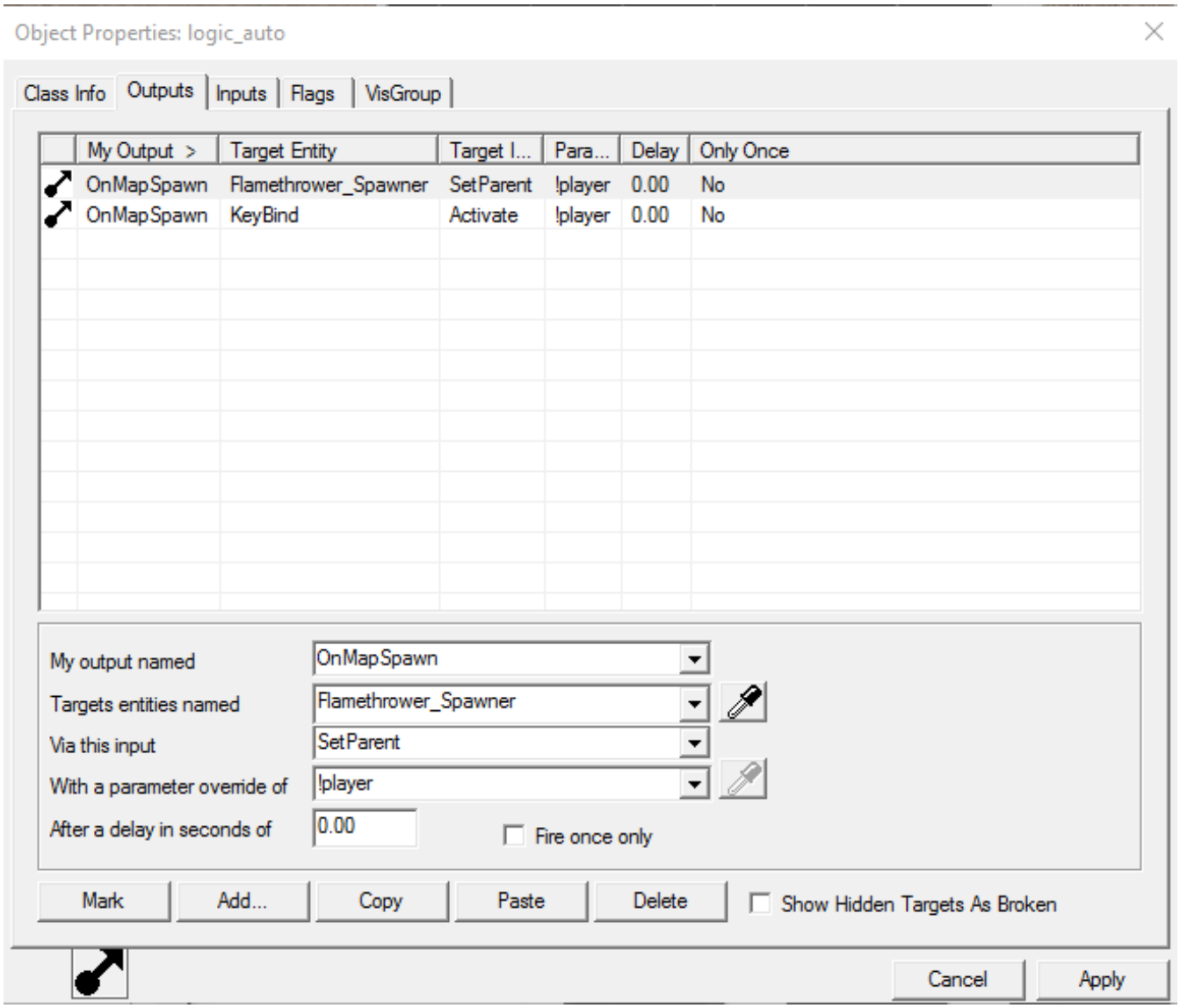
We also need to change some of the default flags of the game_ui. The defualt options turns off the game_ui and doesn't allow the player to move or shoot, so we need those options off.

- [ ] Go to the Flags tab of the game_ui
  - [ ] Uncheck all of the options here



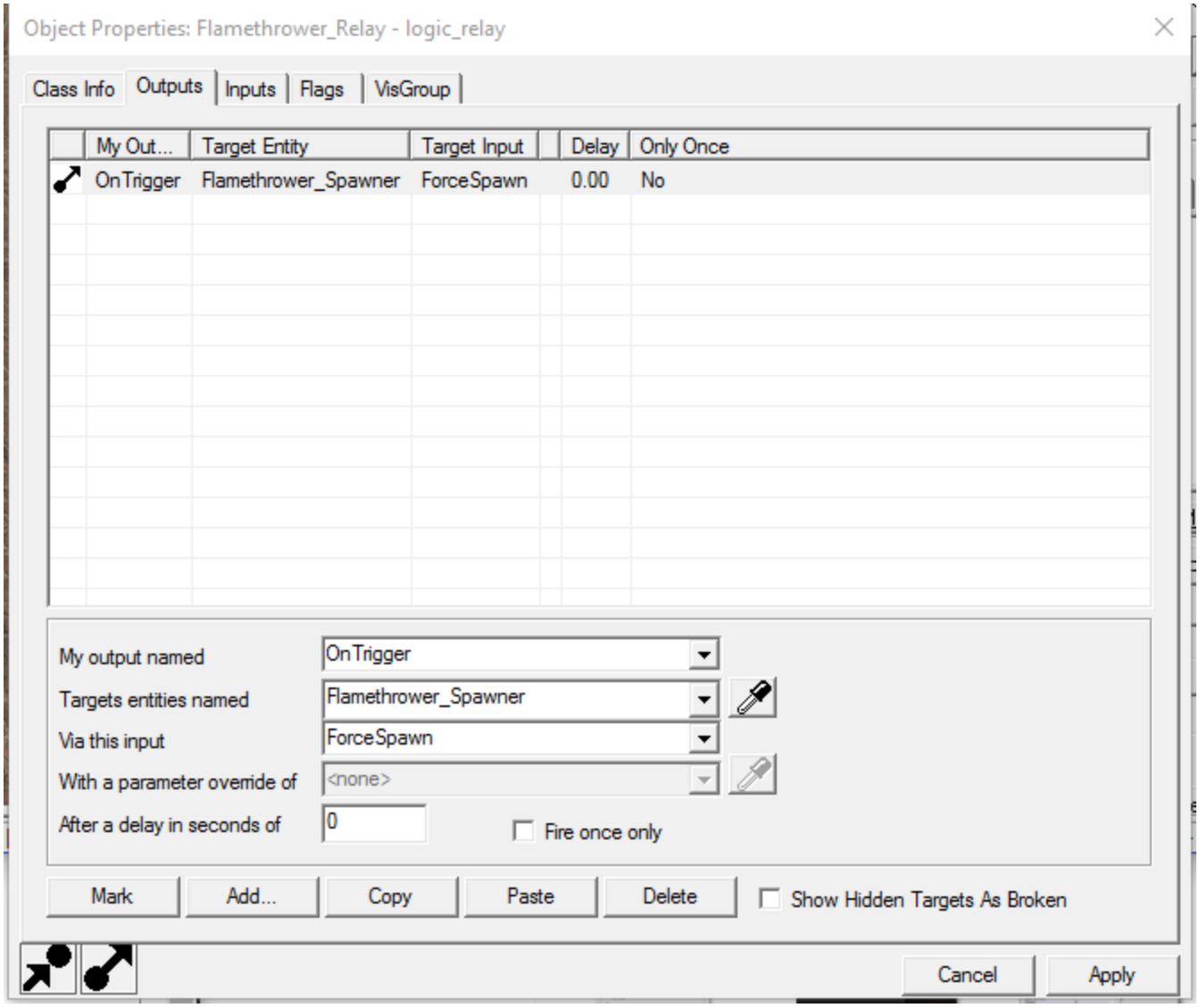
However, before all of this will work, it needs to be activated in our logic_auto at the beginning of the game.

- [ ] Go to Output tab in the logic_auto
- [ ] Add...
    - [ ]
    - [ ] Output: OnMapSpawn
    - [ ] Target: KeyBind
    - [ ] Input: Activate
    - [ ] Parameter: !player

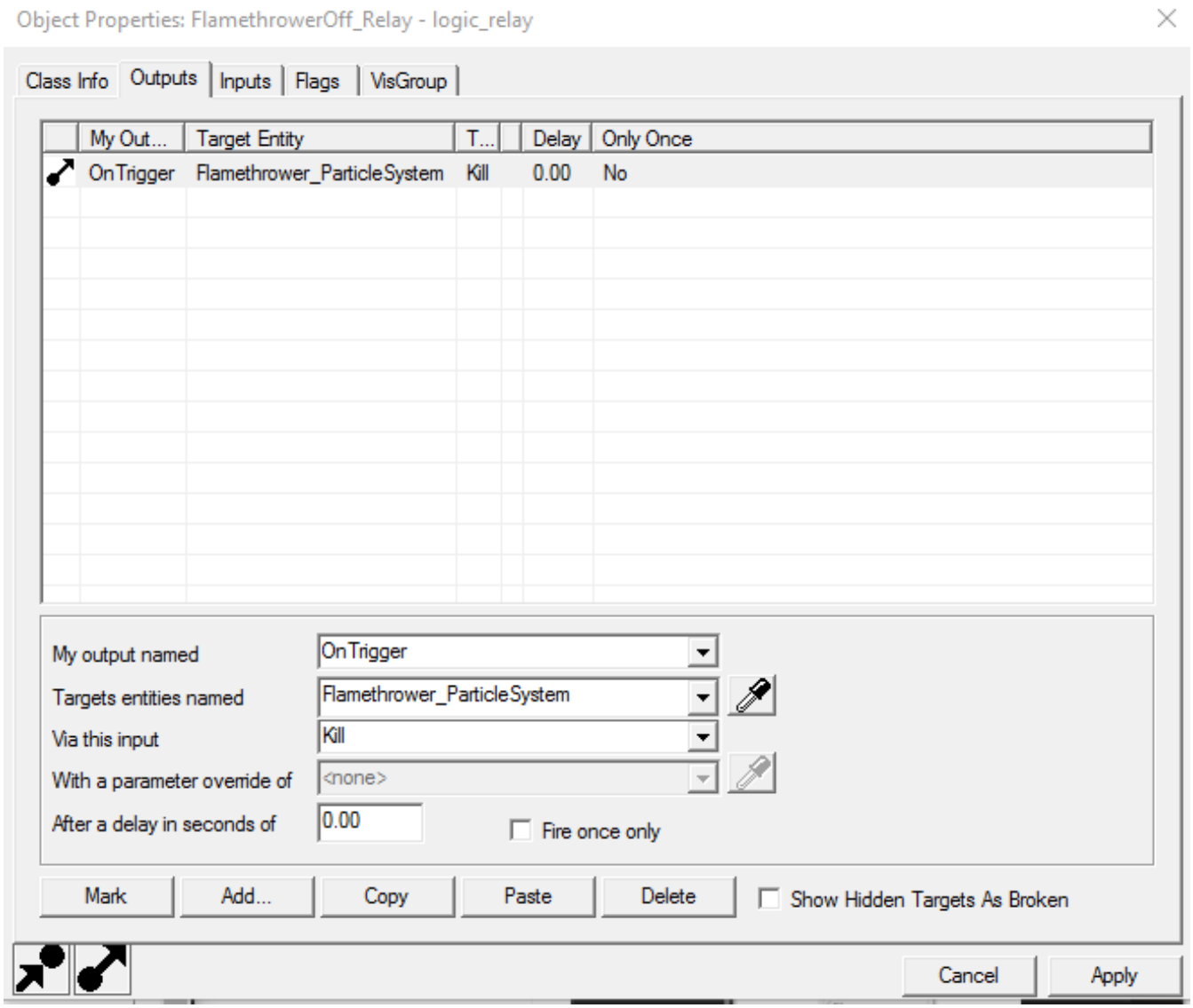


To set up the logic_relays we just have to tell them to have the spawner spawn a particle system and then destroy that particle system when I let go of the right click.

- [ ] On the Flamethrower_Relay: Go to Outputs
  - [ ] Add...
    - [ ] Output: OnTrigger
    - [ ] Target: Flamethrower_Spawner
    - [ ] Input: ForceSpawn



- [ ] On the FlamethrowerOff_Relay
  - [ ] Add...
    - [ ] Output: OnTrigger
    - [ ] Target: Flamethrower_ParticleSystem
    - [ ] Input: Kill

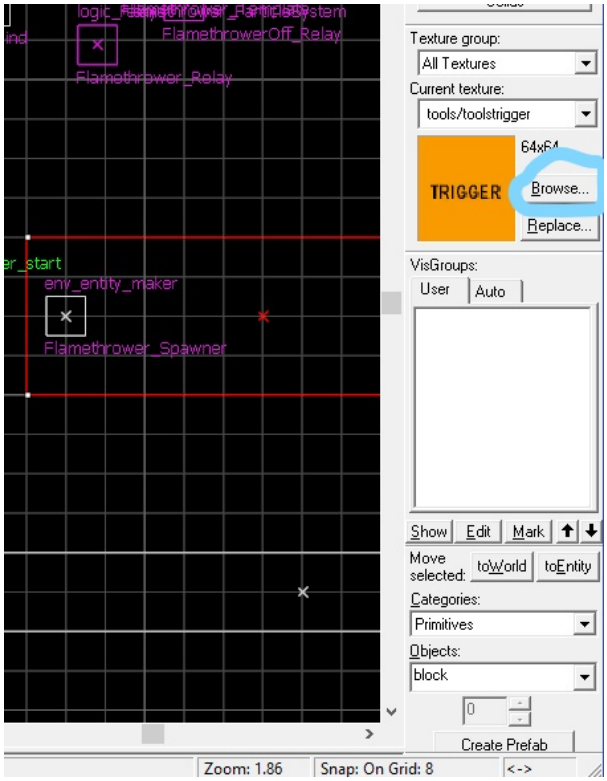


With all of that in, you will spawn a flamethrower whenever you press the right mouse button, and it will turn off when you let go. Feel free to test out your progress at this point by running the map. Remember to take out the -tools from your run parameters if you want to be brought directly into the level. Next up is adding in the ability for this ability to damage enemies.

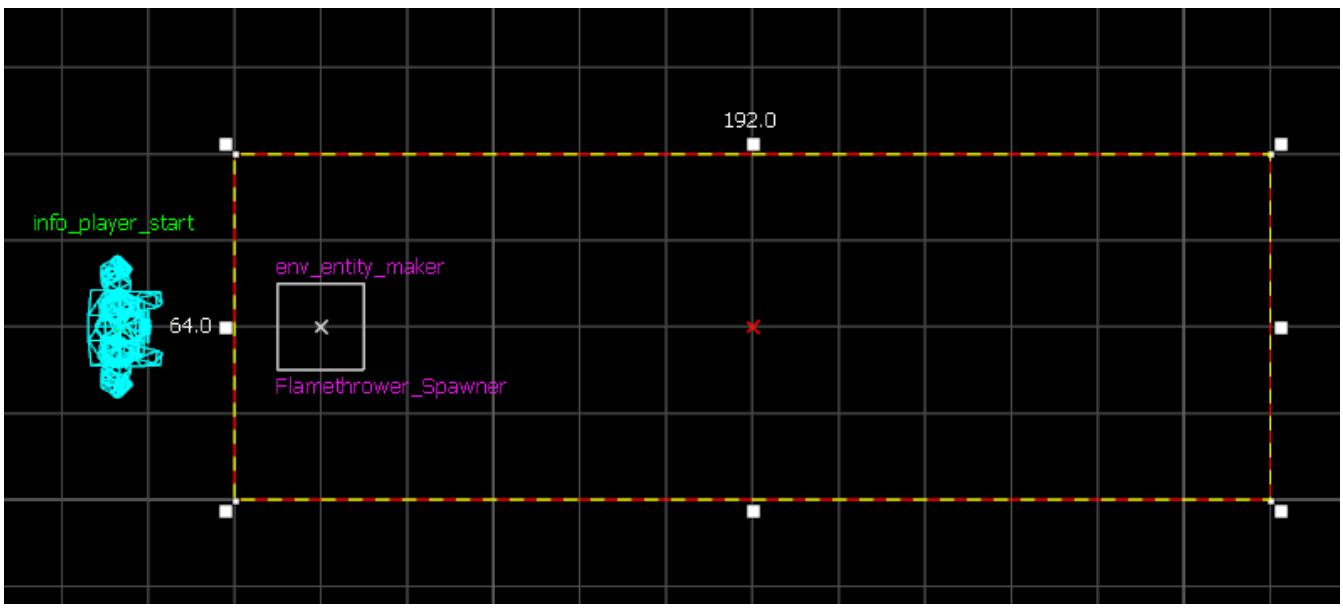
Give the Ability Some Damage

Adding in damage is a simple process compared to the rest of what we've done. First we need to create a piece of BSP in front of the player that will be our damage area.

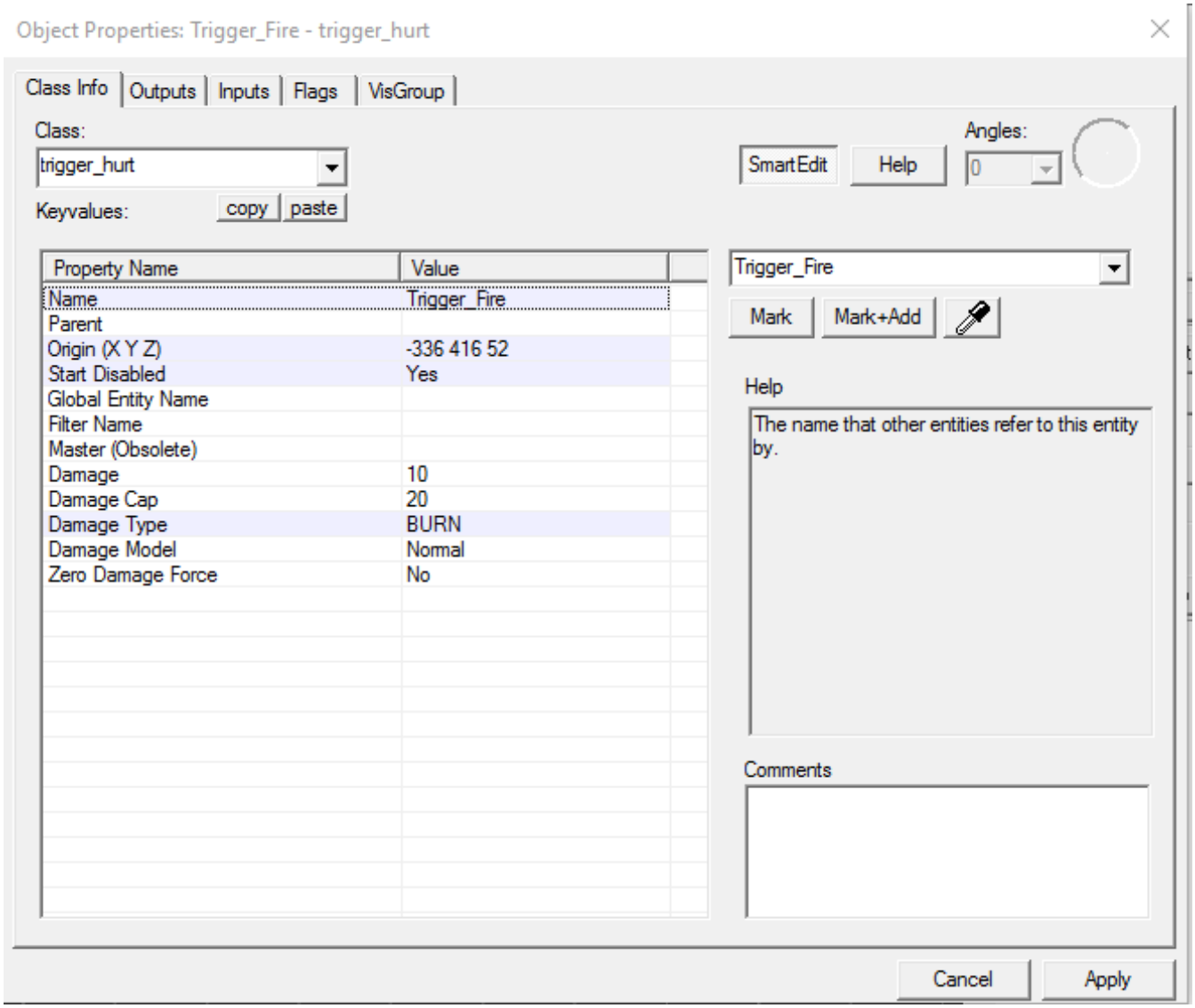
- [ ] Change your selected texture to the ToolsTrigger texture
  - [ ] Click the browse button under the texture setting in the right panel
  - [ ] Find the ToolsTrigger texture



- [ ] Create a BSP in front of the player using the trigger texture
  - [ ] It should be 192 x 64 x 72(tall)
  - [ ] Place it 16 units in front of the player_start



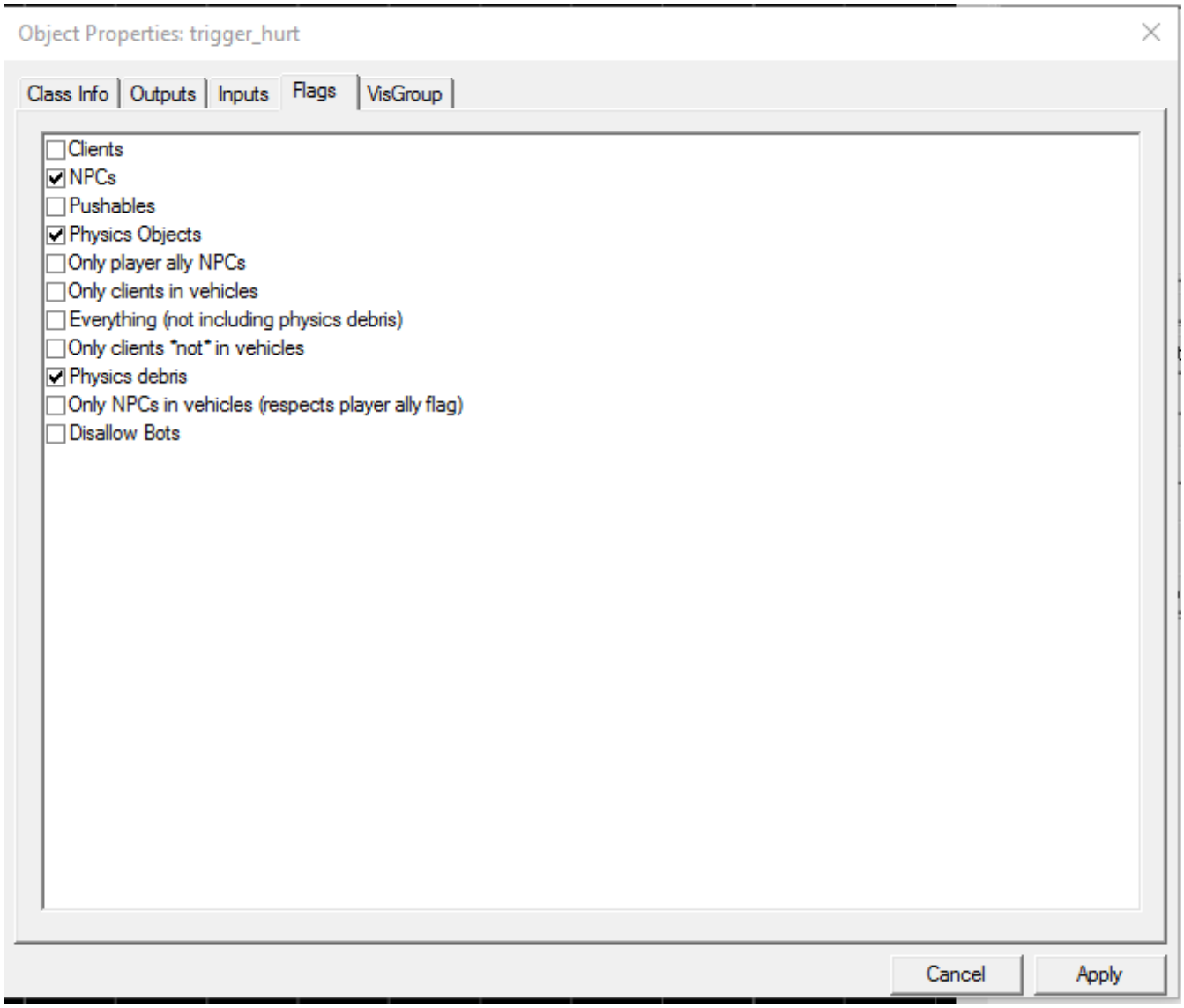
- [ ] Right Click the BSP and choose Tie to Entity (Or Press CTRL + T while it is selected)
- [ ] Change the class info from func_Detail to trigger_hurt
- [ ] Set the following values:
  - [ ] Name: Trigger_Fire
  - [ ] Start Disabled: Yes
  - [ ] Damage Type: BURN



This creates a trigger that hurts enemies that walk into the area defined by the BSP, using burn damage. By setting it to start disables, we can turn it on with out flamethrower_relay when the particles turn on and back off when the flamethrowerOff_Relay triggers. First we have to finish up the trigger by setting some flags for it.

- [ ] Set the following flags:
  - [ ] NPCs
  - [ ] Physics Objects
  - [ ] Physics Debris
- [ ] Uncheck the following flags:
  - [ ] Clients

These settings allow the trigger to hurt NPCs as well as any physics objects in the world like wooden crates and it ensures that it will never hurt the player.



We'll want to turn the damage on and off with the logic_relays. We need to update their outputs to enable and disable this new trigger.

- [ ] Add new output to Flamethrower_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Trigger_Fire
  - [ ] Input: Enable

OnTrigger	Trigger_Fire	Enable	0.00	No

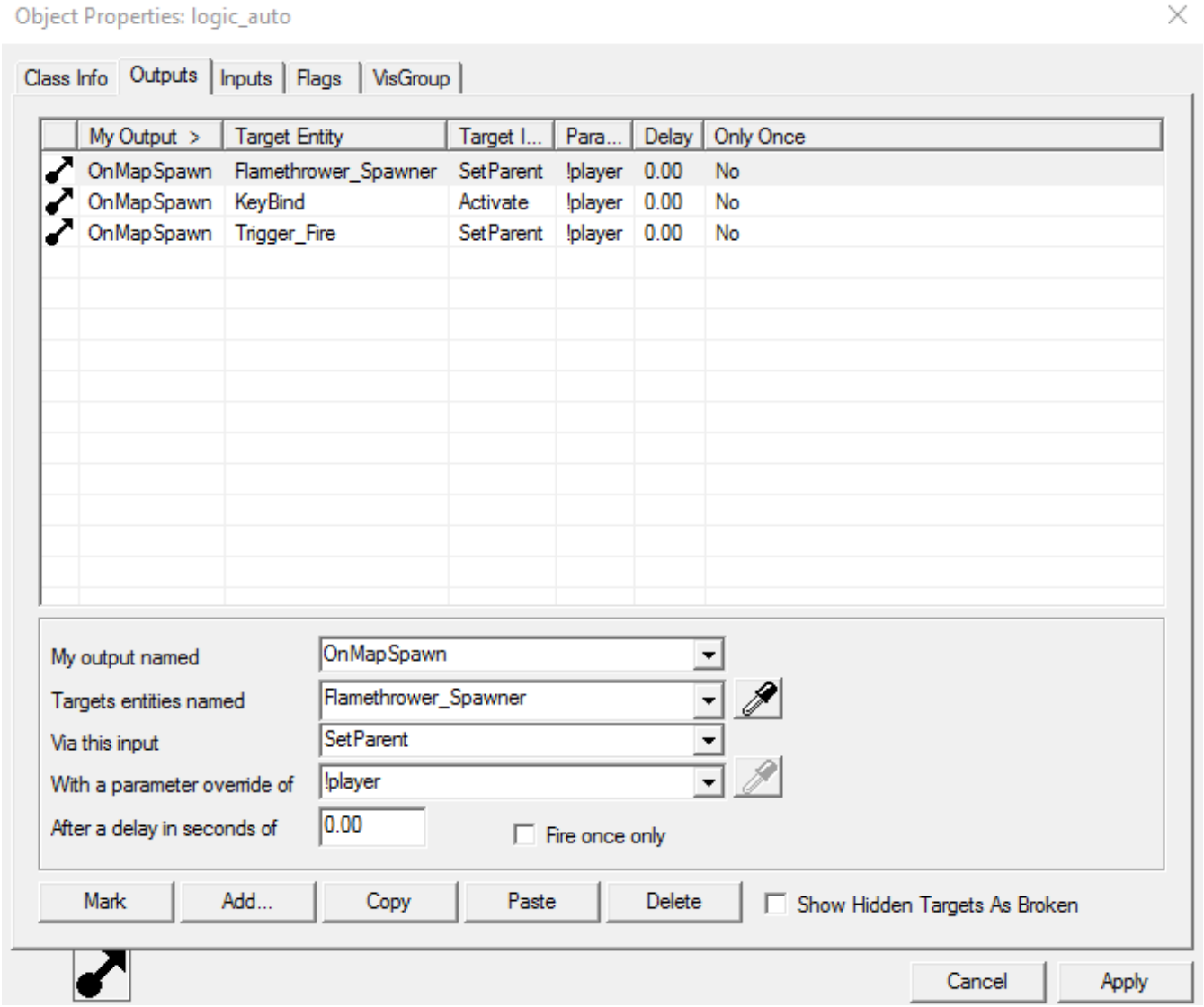
- [ ] Add new output to FlamethrowerOff_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Trigger_Fire
  - [ ] Input: Disable

OnTrigger	Trigger_Fire	Disable	0.00	No

Lastly we need to parent this box to the player so that it follows you around as you move about the space and is always covering the area you are looking. This is done the same way we parented the entity_maker to the player: with the logic_auto.

- [ ] Go to Outputs on the logic_auto
- [ ] Add...

- Output: OnMapSpawn
- Target: Trigger_Fire
- Input: SetParent
- Parameter: !player



To test out that it's working, add an NPC zombie to your world and set him on fire!

Congratulations! You made it this far and created a working flamethrower with custom particles. More advanced ideas of how to use it follow in the next sections, but you have successfully created a working ability.

Making the Flamethrower More Interesting

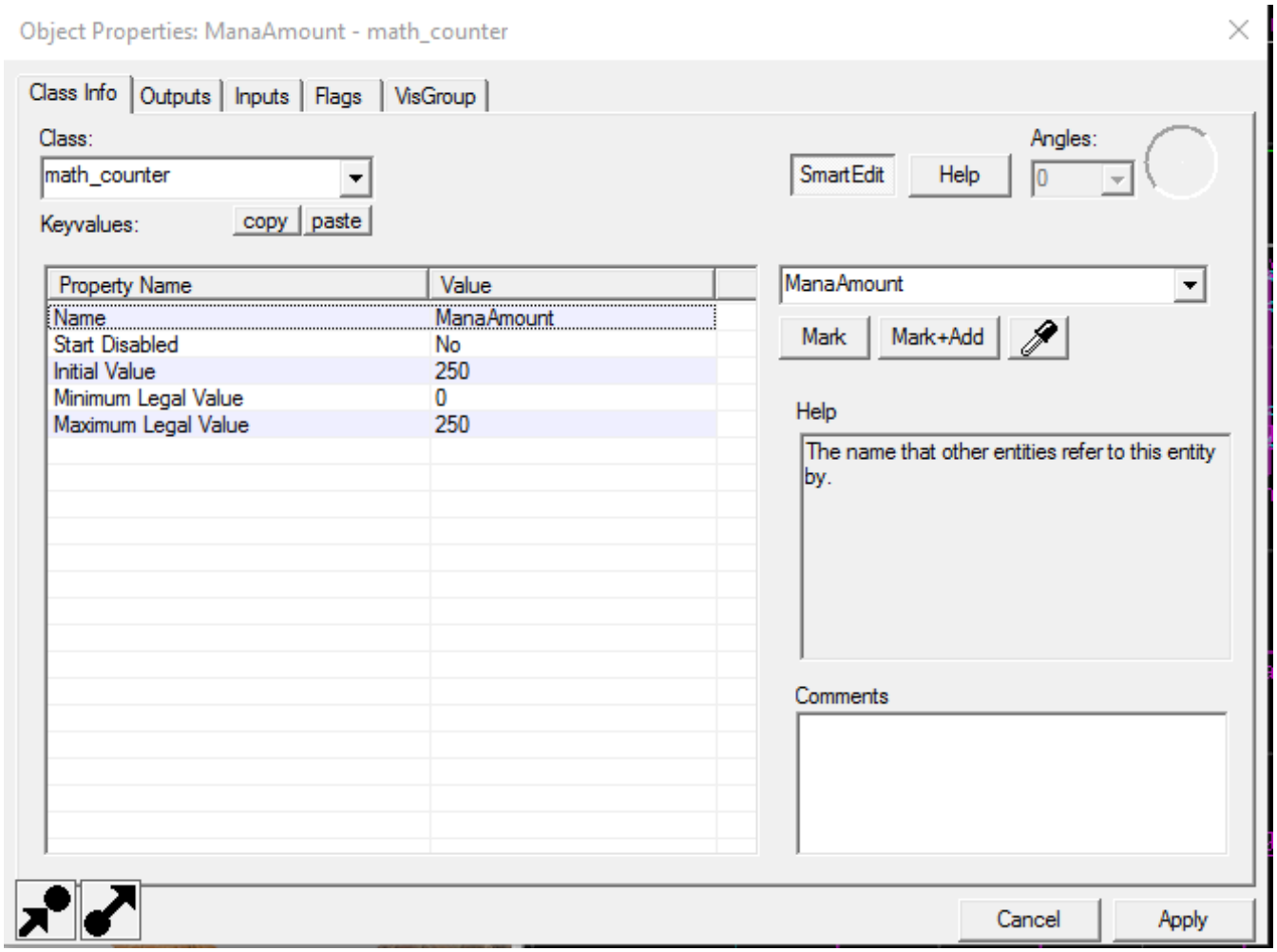
The following topics are additions to the flamethrower that use different entity types and modify the way it works to be more interesting for gameplay. They are not dependent on eachother.

Connecting the Ability to a Mana Cost

This gives your ability a mana cost. It involves a counter that gets subtracted from based on a timer and is replenished by a separate timer.

First step is creating the math_counter to keep track of your player's mana so that we can edit it later.

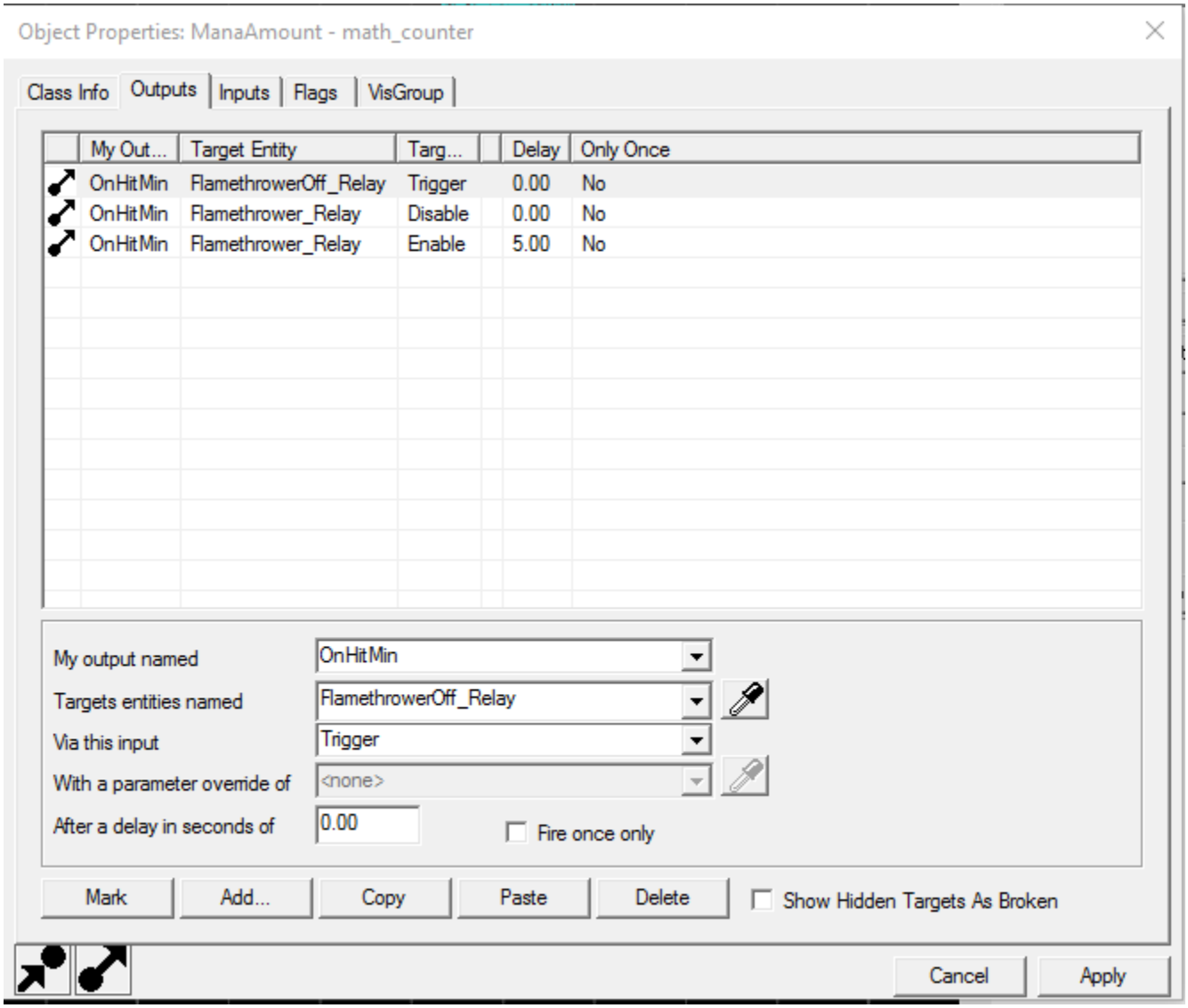
- [ ] Create a math_counter with the entity tool
- [ ] Change the following values:
  - [ ] Name: ManaAmount
  - [ ] Initial Value: 250
  - [ ] Maximum Legal Value: 250



These values sets a maximum amount of mana that you can hold at once as well as setting your current mana to the max. Next we can set up the outputs for the counter so that when you reach the minimum amount (set to zero above) you trigger the flamethrower off relay (turning the flamethrower off) and disabling the flamethrower temporarily.

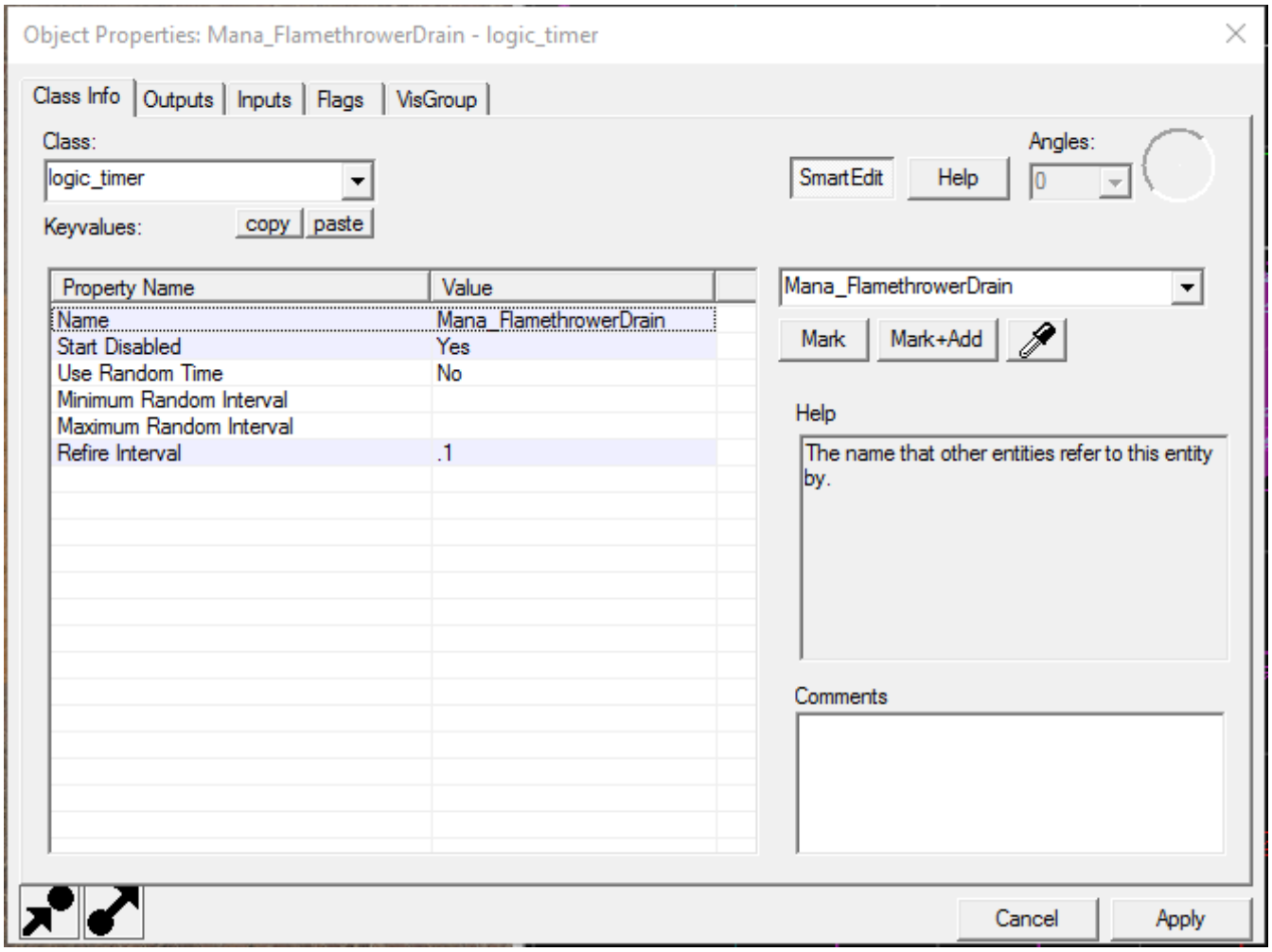
- [ ] Go to the Outputs Tab
  - [ ] Add...
    - [ ] Output: OnHitMin
    - [ ] Target: FlamethrowerOff_Relay
    - [ ] Input: Trigger
  - [ ] Add...
    - [ ] Output: OnHitMin
    - [ ] Target: Flamethrower_Relay
    - [ ] Input: Disable
  - [ ] Add...
    - [ ] Output: OnHitMin
    - [ ] Target: Flamethrower_Relay

- Input: Enable
- Delay: 5.00



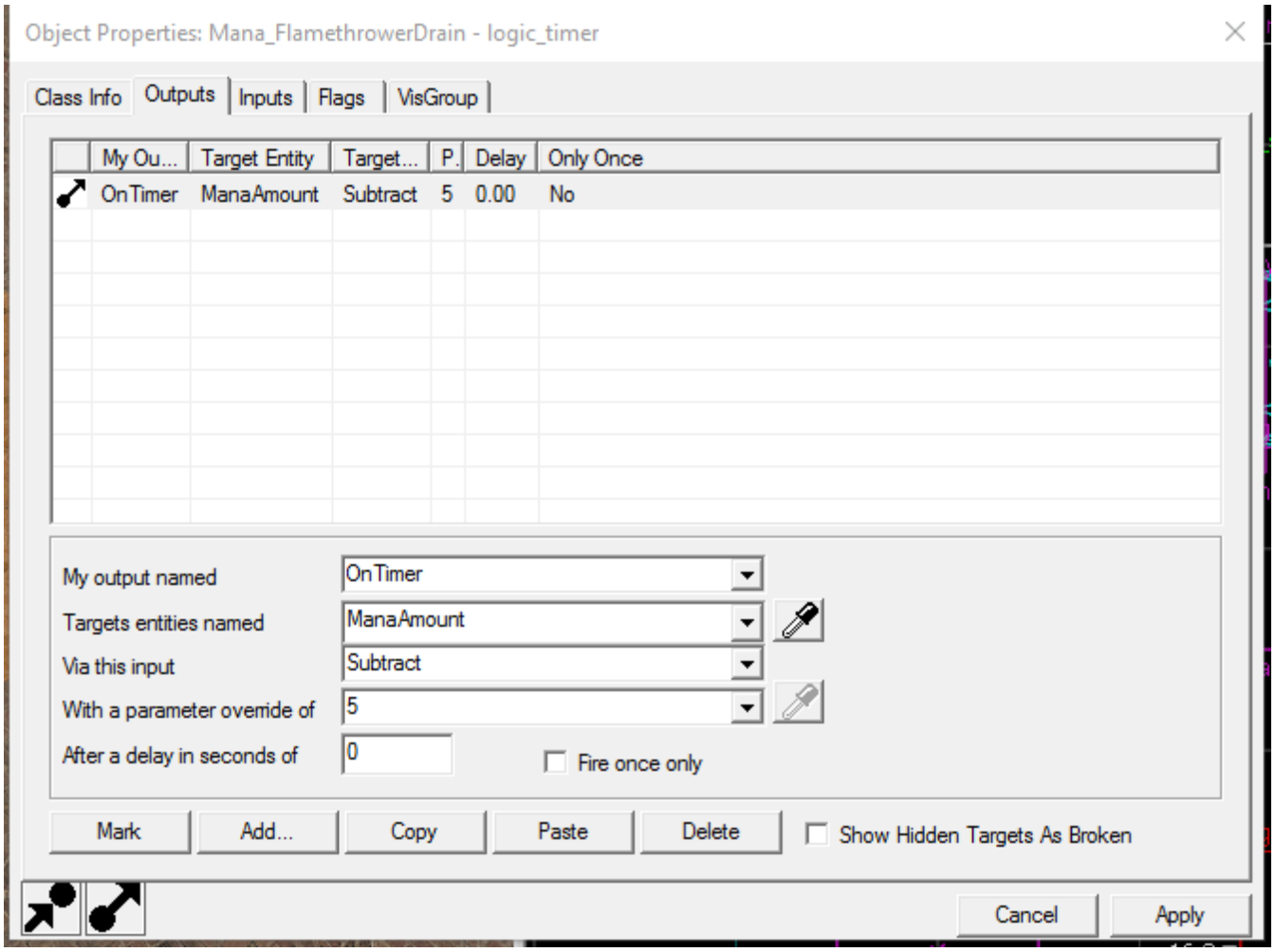
As soon as the counter hits its minimum value, it tells the flamethrowerOff_relay to turn off the flamethrower, and the ability to shoot again is disabled for 5 seconds using the delay parameter of the input/outputs. The mana drain and regeneration is going to be created with timers. As the timer hits its limit, it will subtract or add to the counter and only the add or subtract will be active at any time depending on if the player is using the ability.

- Create a logic_timer
  - Name: Mana_FlamethrowerDrain
  - Start Disabled: Yes
  - Refire Interval: .1



It begins disabled because we only want it draining your mana while the flamethrower is active. The refire interval is how often it will fire an output. Every 0.1 second, the timer will trigger an output and subtract from our counter.

- [ ] Go To the Mana_FlamethrowerDrain's Output Tab
  - [ ] Output: OnTimer
  - [ ] Target: ManaAmount
  - [ ] Input: Subtract
  - [ ] Parameter: 5

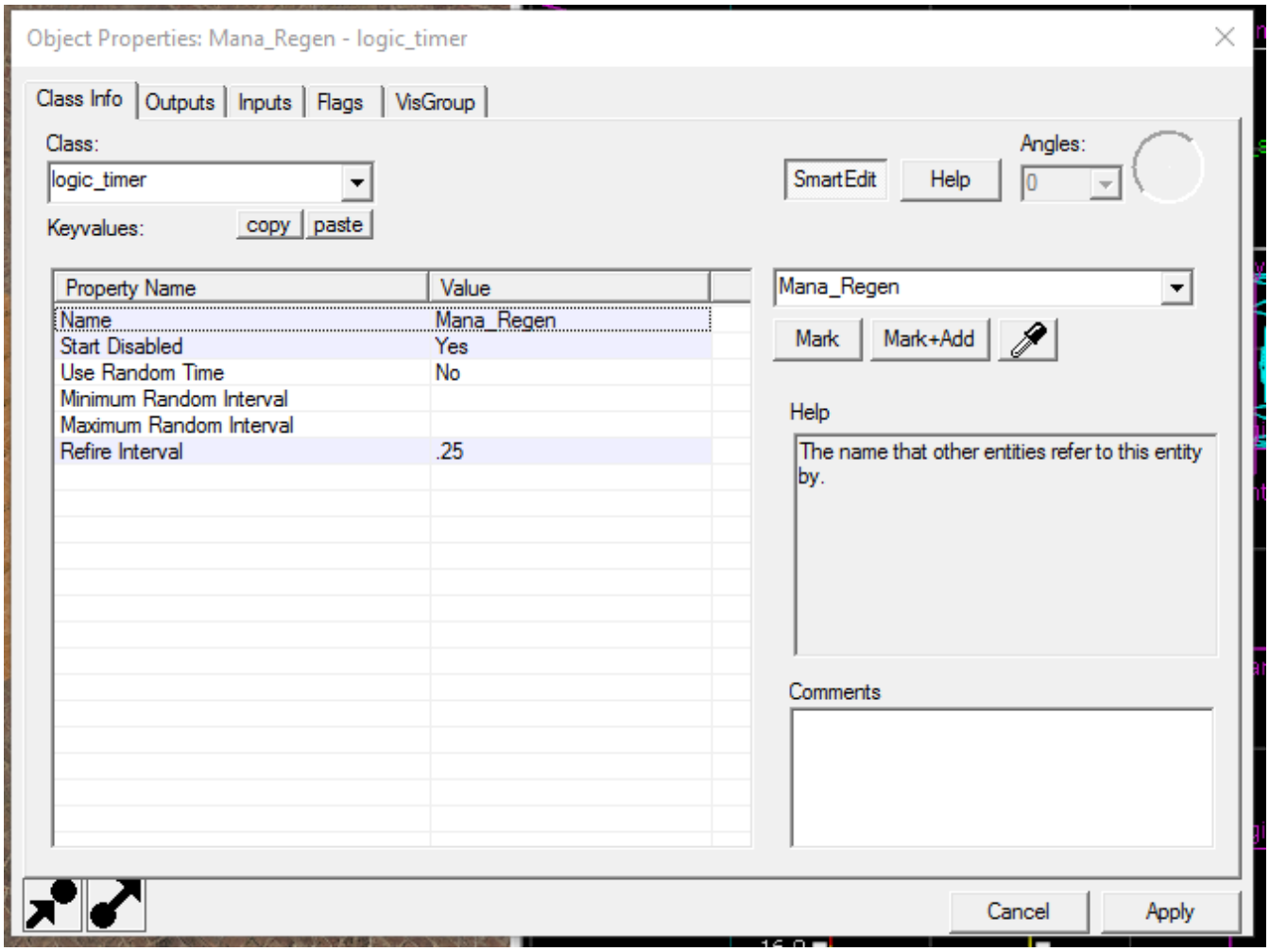


This means that every time the timer fires (set by the refire interval of 0.1 seconds) it will tell our counter (ManaAmount) to subtract 5 from its total value. Now we have to set it up to turn on and off when the ability is being turned on and off. Add the following into the logic_relays for turning on and off the flamethrower.

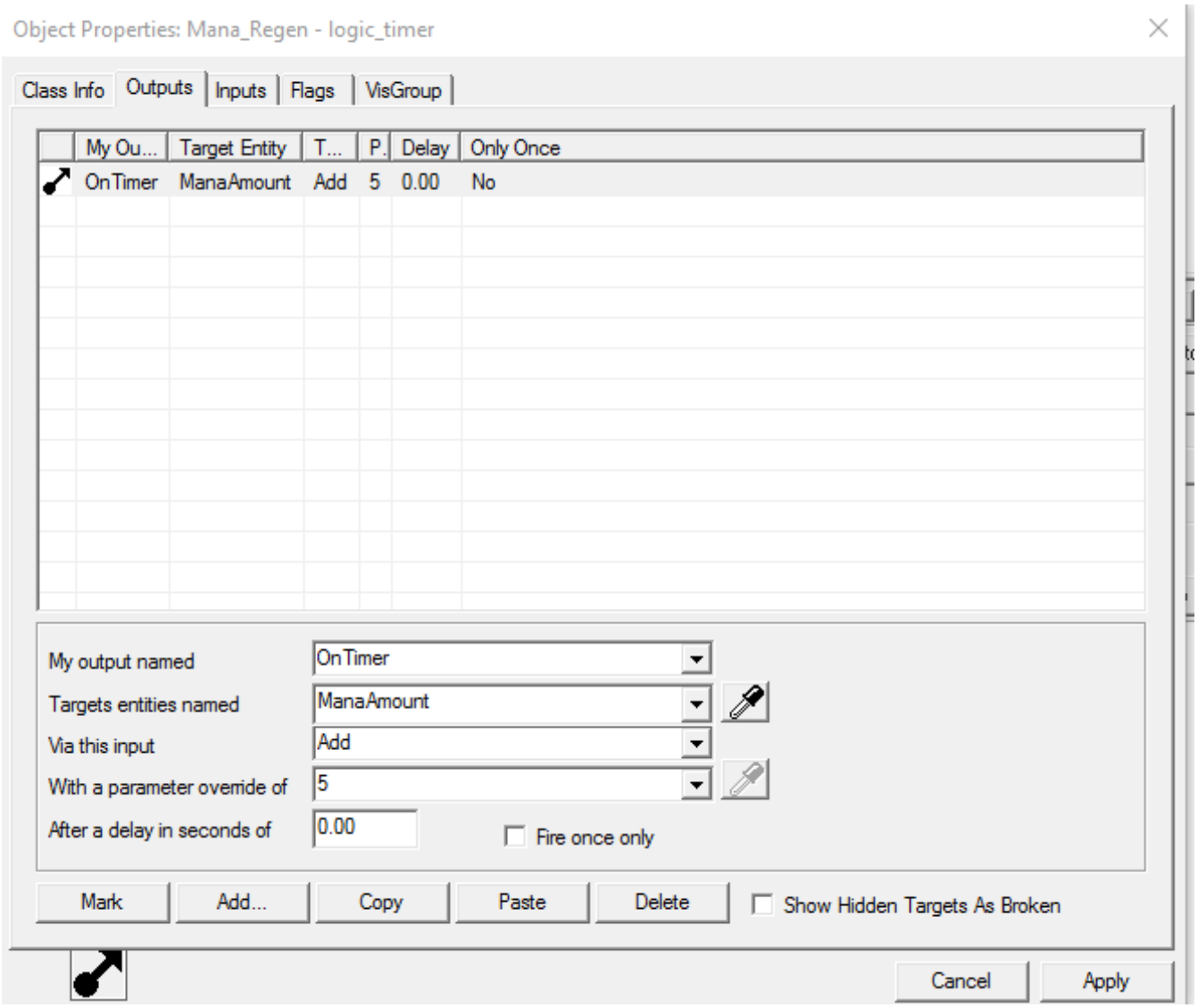
- [ ] New Output for Flamethrower_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Mana_FlamethrowerDrain
  - [ ] Input: Enable
- [ ] New Output for FlamethrowerOff_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Mana_FlamethrowerDrain
  - [ ] Input: Disable

The last step is to set up our mana recharge timer. It works in the same exact way as the drain, but it adds instead of subtracts.

- [ ] Create a logic_timer
  - [ ] Name: Mana_Regen
  - [ ] Start Disabled: Yes
  - [ ] Refire Interval: .25



- [ ] Go To the Mana_Regen's Output Tab
  - [ ] Output: OnTimer
  - [ ] Target: ManaAmount
  - [ ] Input: Add
  - [ ] Parameter: 5



As with the previous timer we have to set it up to turn on and off when the ability is being turned on and off. Add the following into the logic_relays for turning on and off the flamethrower.

- [ ] New Output for Flamethrower_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Mana_FlamethrowerDrain
  - [ ] Input: Disable
- [ ] New Output for FlamethrowerOff_Relay
  - [ ] Output: OnTrigger
  - [ ] Target: Mana_FlamethrowerDrain
  - [ ] Input: Enable

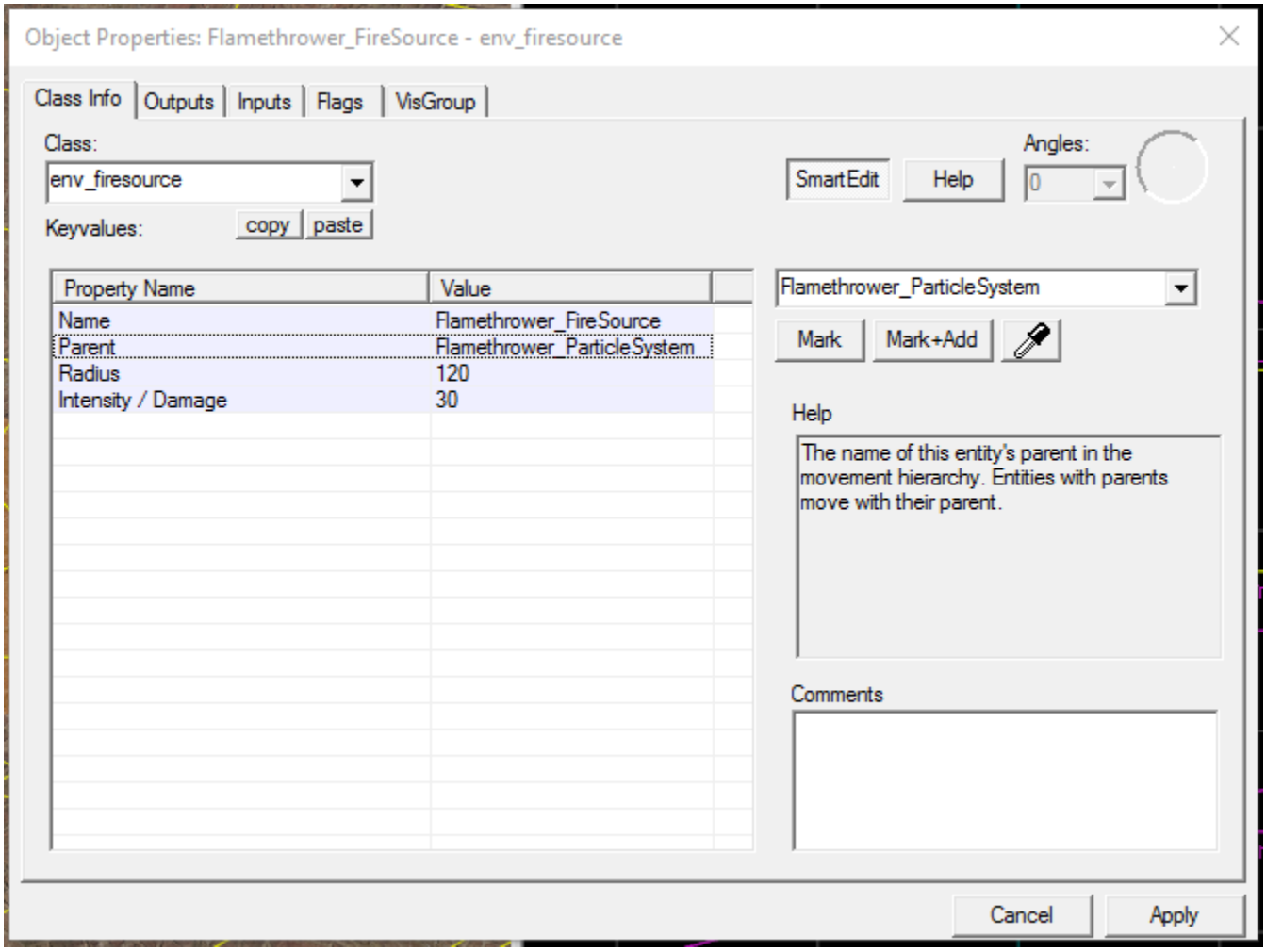
You are now limited in the amount of time you can use your ability.

Start Fires in the World

The final addition is the ability to start fires in the world. The burn trigger will not ignite fires on its own. To do this we need to add an env_firesource to our entity maker so it spawns when the player is shooting fire. And to test it out, we're going to need an env_fire in the world to ignite.

- [ ] Create an env_firesource
- [ ]

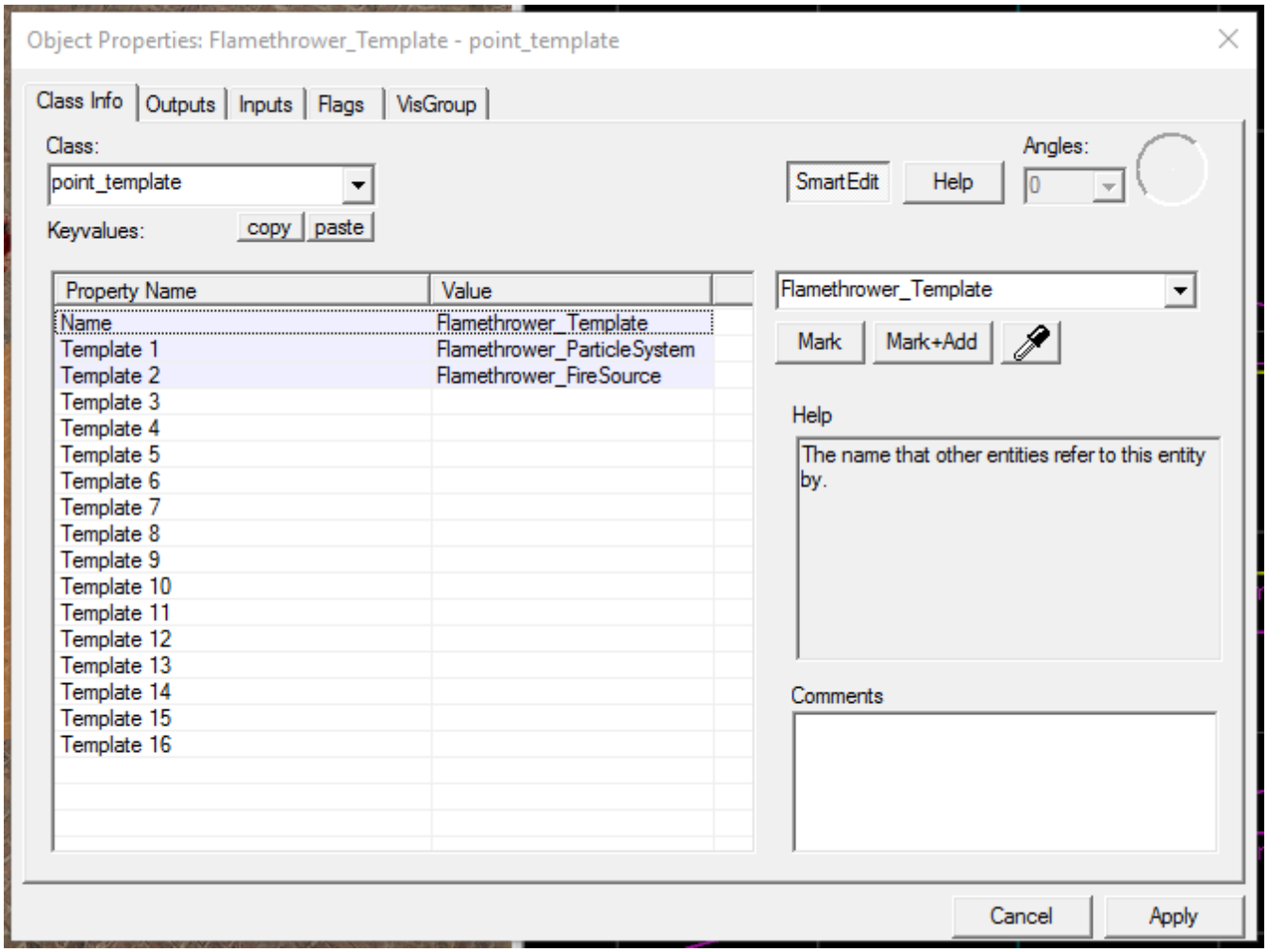
- [ ] Place it 96 units in front of the point_template that contains the particle system
- [ ] Set up the class info:
  - [ ] Name: Flamethrower_FireSource
  - [ ] Parent: Flamethrower_Particle_System (This way it is destroyed with the particle system and already parented to it when it spawns, saving us having to parent it to the player and killing it separately)
  - [ ] Radius: 120
  - [ ] Intensity: 30
- [ ] In the Flags tab, make sure StartsOn **IS** checked



This entity will provide heat inside its radius with the intensity chosen. If there is an env_fire close to it, that env_fire will ignite without having to be specifically scripted to.

Now finally, make sure the firesource is attached to our point_template so that it spawns when the particle system spawns.

- [ ] In the point_template (Flamethrower_Template)
  - [ ] Add our new firesource (Flamethrower_FireSource into the Template 2 slot



Place an env_fire somewhere in the world and shoot your flamethrower at it and it should ignite.