

## AMHS v5.2 GUI Explanation

This explanation has been created for people that are still using GUI and are finding it difficult to get JassNewGen configured and working as well installing the actual AMHS itself

### JassNewGen

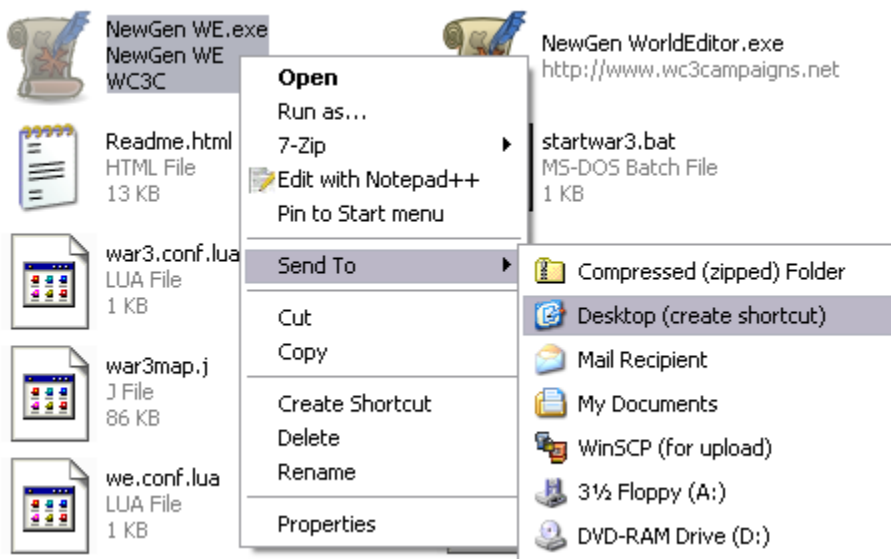
JassNewGen is a WE patcher/hacker that has been created by many people. It allows for many more options in WE, especially for JASS'ers. By far the most useful feature that JassNewGen introduces with vJASS is the OOP (Object Orientated Programming). Not only has this made creating the system far more efficient (and because of this some may consider even possible) but it allows for users that already have JassNewGen to easily install the system without any hassles.

NOTE: Some anti-virus software may detect JassNewGen as a virus. This is a false detection, and it most likely thinks that JassNewGen is a virus because it patches another process. The program itself does no intentional harm to your computer, it only patches stuff in WE (that's it)

#### Step 1

Firstly download JassNewGen from

<http://www.wc3campaigns.net/showthread.php?t=90999>. When you download it, extract its contents to any folder (it can be anywhere) using [7zip](#) or [WinRAR](#). Once extracted inside the folder you will notice NewGen WE.exe file, from now on whenever you want to launch World Editor you must launch it through this file. To make things easier you can create a shortcut to your desktop by right-clicking -> Send To -> Desktop (create shortcut)

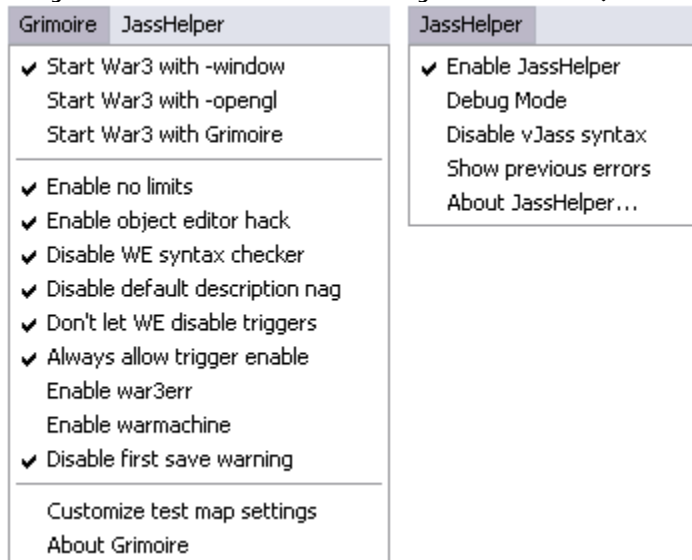


#### Step 2

Now when World Editor is opened you will notice that some extra menu items have been added which provide more functionality. We need to make sure certain options are enabled/disabled in order for WE to work



Now make sure the settings in your WE matches the ones showed here, note that they don't have to be exactly the same (the settings will be explained)



Start War3 with -window – This starts Warcraft 3 with a windows, this basically makes launching wc3 much faster as well as easier

Start War3 with -opengl – OpenGL is another graphics API like DirectX. There is no reason to enable this unless you are emulating JassNewGen on Linux/Mac or you are have problems with DirectX

Start War3 with Grimoire – This option is only used if you are also patching Wc3 (e.g. custom native injection etc). In most cases you will not need to Start War3 with Grimoire

Enable no limits – Many terrain'ers may already know that normally in WE there is a limit to how many doodads (or destructables) can be placed on the map. This option removes the limit

Enable object editor hack – If you enable this option whenever you past Object Editor data in your map with CTRL + V (i.e. you copy a unit from another map and paste it in your own) it allows you to specify the rawcode for the unit

Disable WE syntax checker - **REQUIRED** In order for the vJASS syntax checker to work you must disabled the original WE syntax checker

Disable default description nag – This will disable the message that comes up when you save your map with without changing the description in Scenario – Map Description (i.e. your map description is still just "Another Warcraft III map"

Don't let WE disable triggers – This will stop the Trigger Editor from disabling triggers if there is any issue with code in one of them (i.e. the name of the global trigger doesn't match the name of the trigger in the code)

Always allow trigger enable – Will always let you enable a trigger regardless of what code is inside it

Enable war3err – **NOTE:** The latest release of JassNewGen (4.d) has a bugged version of war3err and using it will cause AMHS v5.x to crash. This means either disable it OR get the fixed wc3err from here <http://www.wc3campaigns.net/showpost.php?p=996174&postcount=248>. wc3err is a dll that gets injected into Wc3 which detects handle leaks, prevents Wc3 from crashing in most cases (infinite loops, ExecuteFunc on invalid trigger name etc) and many other options **NOTE:** In certain cases it has been noticed that wc3err does slow down the map considerably especially with very fast loops, so if your map is lagging very badly then disable wc3err to see if it fixes the problem. Also in multiplayer wc3err will cause desyncs

Enable warmachine – If you wanna use Grimoire's faster Jass VM, this is the option to enable it. When enabled this will disable war3err and vice versa because they do not work together. *Quote from JassNewGen readme v4.d (I haven't used warmachine so I can't comment on it)*

Disable first save warning –With the older versions of JassNewGen there was an annoying prompt whenever you tried to save a map for the very first time, this disables it

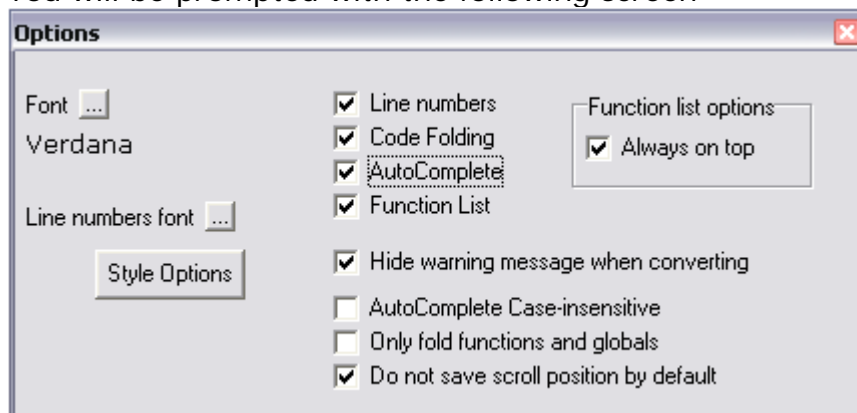
Enable JassHelper – **REQUIRED** JassHelper is the improved syntax checker created for World Editor that checks for vJASS and allows AMHS to be used in your map. The syntax checker featured in JassHelper is also much more accurate then the WE one.

### **Step 3**

Now we only need to configure one more thing (TESH) and then we will be ready to install AMHS. TESH is a incredibly handy syntax highlighter for WE and adds many other options. Open up the Trigger Editor and you will notice another menu item, TESH. Click on it and select Options



You will be prompted with the following screen



You may want to mimic the settings shown here. The one that needs special attention is Do not save scroll position by default. TESH by default will save scroll position of each trigger done in JASS, unfortunately this is done incredibly stupidly

(it just saves the line number inside the actual trigger). This means that if you have Save Scroll position enabled for a trigger, whenever you scroll in that trigger you are forced to save your map again. Enabling Do not save scroll position by default means that any new triggers that are pasted/created in Trigger Editor will have this Save scroll position disabled



Unfortunately if you have any other JASS triggers in your map before you enable the Do not save scroll position by default you need to go through them all and disable the Save scroll position trigger (however any new triggers that get placed in will already have this option disabled)

Now that all the settings are done, let's actually get into installing AMHS into your map

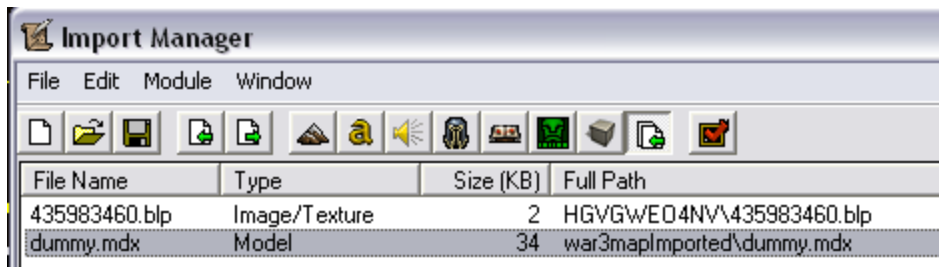
## Installing AMHS

Now we can install AMHS since everything is set up. The first most important part to realise is that the system cannot read data from the Object Editor (this currently isn't possible yet) and because of this you need to feed the relevant information from the Object Data manually into the system.

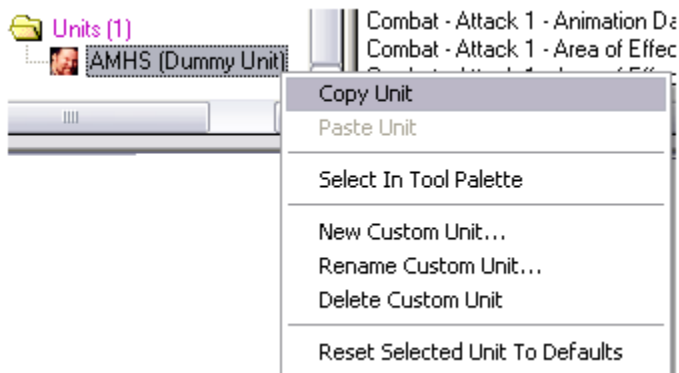
Now we need to copy the triggers over. You need to copy the following Triggers over into your map

- Firstly copy the **Replay Detect Engine** trigger
- Now you need to pick a Unit Attachment System. The recommended one is Game Cache (which is used by default) however if you want something that is faster than you can use HAIL. The other attachment systems aren't recommended for GUI users, since they can only be used in JASS. If you pick Game Cache then you just need to copy the **AMHS Game Cache** otherwise if you want to use HAIL then you need to copy the **AMHS HAIL** as well as the **HAIL** trigger located underneath the **Handle Attaching Libraries** trigger comment.
- Now copy the rest of the triggers, they are as follows
  - **Fog Protect and Shadow Engine**
  - **Shadow and UnitGraphic Configuration**
  - **Buff Configuration**
  - **MiniMap Protect Engine**
  - **Preload String**
  - **AddSpecialEffect Emulator**
  - **Initiate Replay Detect**

We now need to copy the files that are used by the AMHS system. Go into your import manager and you should see this

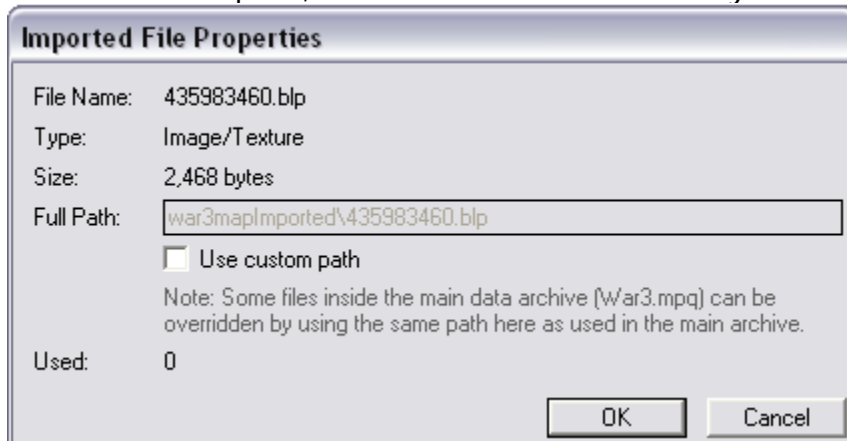


We need to export both files by right clicking them and selecting **Export File**. You then need to import the files into your map by pressing **CTRL + I**. If you want to use the AddSpecialEffectEmulator (explained later) then you also need to export the **dummy.mdx** file as well as copying the AMHS (Dummy Unit) into your map



Make sure that the AMHS (Dummy Unit) has the dummy.mdx as its model.

You will notice that when you import the file **435983460.blp** file it will have the **war3mapImported\HGVGWEO4NV\435983460.blp** path. We now need to randomize this path, double click on the file and you should see a screen like this



Select **Use custom path** so you can change the path. Now we need to change the filename and filepath of the file to anything random that you want, making sure that whatever you enter ends in **\*.blp** where the **\*** is a single character and that you enter characters, letters and **\**. The following are all valid example of a random path that can be used by the system. Later on we will use this path for the MiniMapProtect engine

- **Wihgiwhgil\w5r32twe\sgwe\t\weg\fae\r\werra\ah\gre.blp**
- **Wiwghio.blp**

- Fj2390t23Ifkwe\ag32.blp
- T234gnwelgr\agvawergaerg.blp

You can just leave `dummy.mdx` path as `war3mapImported\dummy.mdx`

## Shadows

The type of Data that needs to be fed into the system is shadows. We need to put Object Data for units into the **Shadow and UnitGraphic Configuration** trigger for **EVERY** unit that is used by your map. If you forget to input a unit will display an error message notifying. If we have a look at the **Shadow and UnitGraphic Configuration** then you will see something like this

```
//*****
//*
//*      SHADOW/UNITGRAPHIC CONFIGURATION SETTINGS START HERE
//*
//*****

// Register Shadows

call RegisterUnitShadow('Hblm',65,65,170,170,"normal")
call RegisterUnitShadow('osw1',60,60,140,140,"normal")
call RegisterUnitShadow('osw2',60,60,140,140,"normal")
call RegisterUnitShadow('osw3',60,60,140,140,"normal")
call RegisterUnitShadow('oeeye',25,25,70,70,"normal")
call RegisterUnitShadow('now1',30,30,60,60,"flyer")
call RegisterUnitShadow('now2',30,30,60,60,"flyer")
call RegisterUnitShadow('now3',30,30,60,60,"flyer")

//For the dummy unit used by the AddSpecialEffectEmulator
call RegisterUnitShadow('h000',0,0,0,0,"none")

// RegisterUnit Graphics
```

As you can see there are already entries there from the test map, you can delete these. Now we need to add the units from your map into the system, to do this we just use the `call RegisterUnitShadow(UNITID,shadowX,shadowY,shadowH,shadowW, unitShadow)` line. The fields that match this line are the same ones outlined in the Object Editor. If you go to the Object Editor, go to the units section and press **CTRL + D** to display the unit rawcodes if they aren't already displayed

Heroes (4)	scale	1.85
Udea (Death Knight)	unitShadow	Shadow
Ulic (Lich)	shadowX	75.00
Udre (Dreadlord)	shadowY	75.00
Ucrl (Crypt Lord)	shadowH	200.00
	shadowW	200.00

The values in the red rectangle correspond to the **shadowX,shadowY,shadowH,shadow, unitShadow** arguments and the value in blue rectangle corresponds to the **UNITID** **NOTE: UNITID's** must be put inside ' AND ' (inverted commas) and the **unitShadow** must be put inside " and ". Using the example in the picture the line

would be `call RegisterUnitShadow('Udea',75,75,200,200, "Shadow")` and our trigger will end up looking like this

```

//*****
//*
//*      SHADOW/UNITGRAPHIC CONFIGURATION SETTINGS START
//*
//*****

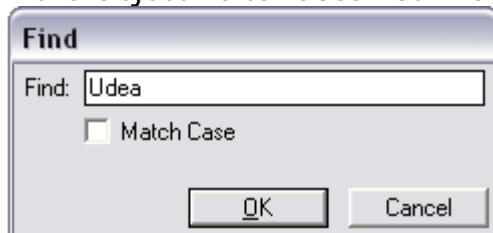
// Register Unit Shadows

call RegisterUnitShadow('Udea',75,75,200,200,"Shadow")

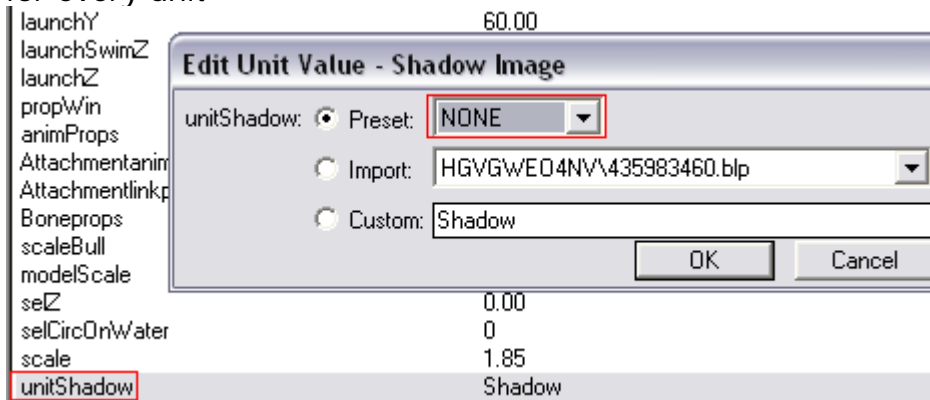
//For the dummy unit used by the AddSpecialEffectEmulator

```

As mentioned before you need to do this to every unit that is used by your map. Once you are done it is highly recommended that you check all the rawcodes, because if you get one incorrect Warcraft III will crash. You can do this by copying the rawcode between the comma's using **CTRL + C**, going to the Object Editor, pressing **CTRL F** and then pasting the rawcode in with **CTRL + V** and pressing enter, if the Object Editor does not find any values it means the code is incorrect

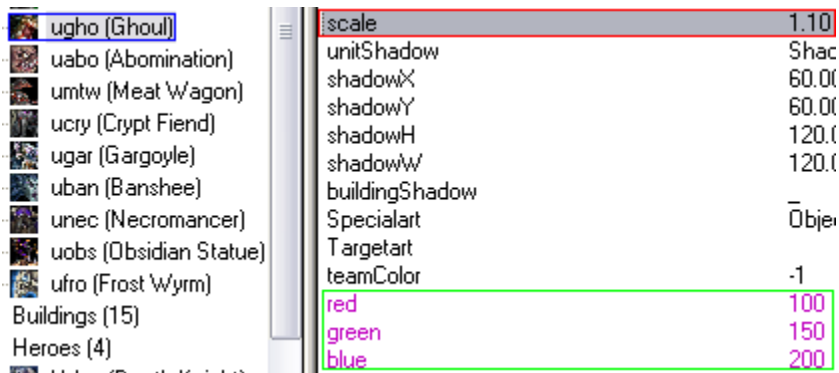


Now we need to do one last important thing, we need to make sure that we disable the shadows for every unit that is used by your map (the ones that we just inputted in the Object Editor). You simply do this by changing the unitShadow field to **NONE** for every unit



## UnitGraphic

Now we need to do UnitGraphics. Unlike Shadow's, UnitGraphics don't have to be done for every unit. They only need to be done with units that you have specified custom tinting/scaling to in the Object Editor



If you have a unit that has a scale value other than 1.00 or any color tints other than 255 you need to input them into the system (otherwise the unit will have default scaling/tinting). You do this the exact same way but you use the **RegisterUnitScale** and **RegisterUnitTint** functions i.e. using our example we would add the lines `call RegisterUnitScale('ugho',1.10)` and `call RegisterUnitTint('ugho',100,150,200)`. Our trigger at this point would look like this, using the ghoul and the Death Knight as an example

```
// Register Unit Shadows

call RegisterUnitShadow('Udea',75,75,200,200,"Shadow")

// Register Unit Graphics

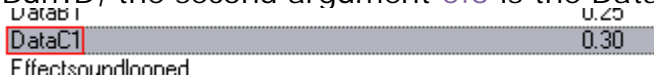
call RegisterUnitScale('ugho',1.10)
call RegisterUnitTint('ugho',100,150,200)

//For the dummy unit used by the AddSpecialEffectEmulator
```

As with the shadows, it is important to check that the UnitID's that you entered are absolutely correct otherwise the map will crash

## Bufs

Luckily this is the shortest part. You only need to specify buffs which alter a unit's size, and the only one that does this is Blizzard's bloodlust. If we have a look at the Buff Configuration trigger you will see there is already one entry which is the Bloodlust spell used by the shaman i.e. `call RegisterBuffScale('Bblo',0.3,1,60,1)`. As you will see there are more arguments that need to be passed into the function than what is seen in the Object Editor. The first argument 'Bblo' is just the the BuffID, the second argument 0.3 is the DataC1 field in the Object Editor i.e.



The third argument 1 is how long it takes the unit to increase/decrease to its size (default is 1), the fourth argument 60 is the duration of the spell and the last argument 1 is the level of the spell.

**NOTE:** If your Bloodlust based spell has more than one level, you need to call this function for EVERY level i.e.

```
call RegisterBuffScale('Bblo',0.3,1,60,1)
call RegisterBuffScale('Bblo',0.3,1,60,2)
```



If the levels have different duration then this is how you set the different options for different levels. Also if you have different versions of a spell based of Bloodlust that have different durations, they must use separate buffs

### **MiniMapProtect**

The only installation procedure we need to do for MiniMap protect is to specify the random path that we created earlier when importing the files into your map. Go to the **MiniMap Protect Engine** and you will see the `private constant string MiniMapPath =` followed by the path of the string. Earlier you would have randomized your own string, replace it with the one already in there i.e. if we used `T234gnwelgr\agvawergaerg.blp` as our random string, the line would look like this `private constant string MiniMapPath = "T234gnwelgr\agvawergaerg.blp"`. Notice how the string is placed in " and ", also the single back slash is replaced with a double black slash (\ replaced with \\)

### **AddSpecialEffect Emulator**

The AddSpecialEffect emulator is more optional then the other engines, if you don't want to use it then you can delete the **AddSpecialEffect Emulator** trigger and the `dummy.mdx` file you imported. What the AddSpecialEffect Emulator does is it hides effects that would normally be created with the **Special Effect - Create a special effect at Point** GUI function. This means if you want to use this emulator, you need to replace all instances of **Special Effect - Create a special effect at Point** with the custom script line call `AddSpecialEffectEx(string modelName, real x, real y)` – This will be explained later

Firstly to set up the AddSpecialEffect Emulator have a look at the **AddSpecialEffect Emulator** trigger in the configuration settings. You will notice `private constant integer Dummy_UnitId = 'h000'` line, you will need to change the 'h000' to the rawcode of the AMHS (Dummy Unit) in your map as it most likely would have changed.

### **Replay Detect**

You will notice in the **Initiate Replay Detect** trigger this line call `TriggerRegisterTimerEventSingle( gg_trg_Initiate_Replay_Detect, 0.00 )`. This initiates the replay detect function, which detects if the game is being played or if it is being viewed as a replay. Unfortunately when this function is called It creates a noticeable pause which can't be hidden. By default the function is called at 0.00 seconds, i.e. as soon as the map loads but if there is a better time to call the function you can replace `0.00` with another time in seconds

## **Replacing Instances of GUI Functions**

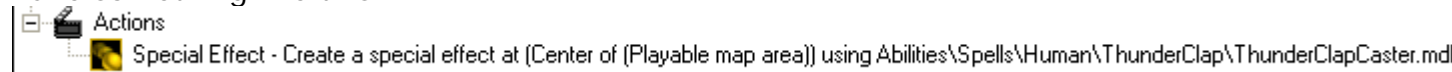
Unfortunately due to the nature of the AMHS system, you can no longer use some of the functions that are used by GUI. These functions are listed below. Fortunately there are replacements for most of these functions

- **Animation – Change Unit Size**
- **Animation – Change Unit Vertex Coloring**

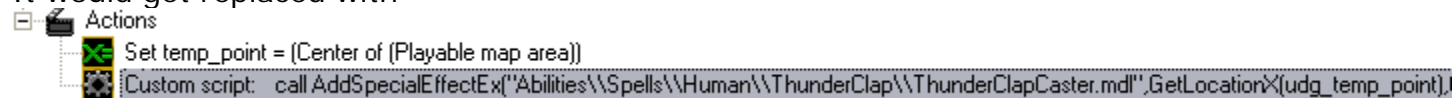
- **Special Effect - Create a special effect at Point** – Only if you are using AddSpecialEffectEmulator
- **Neutral Building – Change Special Minimap Icon** – There is no replacement for this function unless you don't want to use MiniMap protect
- **Neutral Building – Turn Special Minimap Icon On/Off** – There is no replacement for this function unless you don't want to use MiniMap protect

You **MUST** replace all instances of **Animation – Change Unit Size/Animation – Change Unit Vertex Coloring** and you **CANNOT** use **Neutral Building – Change Special Minimap Icon/ Neutral Building – Turn Special Minimap Icon On** unless you want to disable MiniMap protect.

In order to replace the instances you must use the custom script line, so if you have something like this



It would get replaced with



Where the line Custom Script Line reads Custom script: `call AddSpecialEffectEx("\\Abilities\\Spells\\Human\\ThunderClap\\ThunderClapCaster.mdl\\,GetLocationX(udg_temp_point),GetLocationY(udg_temp_point))`

Here is a guide made by Piggy that explains how to do this in GUI

-=INGAME UNIT SIZES=-

For changing the unit's size in game using the trigger editor's : **Animation - Change Unit Size**, the GUI line must be replaced using the **Action : Custom Script** with code :

`call SetUnitScalePercentAMHS(udg_UNIT_VARIABLE,100,100,100)`

Where **udg\_UNIT\_VARIABLE** is the unit whose size is being replaced. To see the exact coding, make a copy of the trigger, use **Edit -> Convert to Custom Script** and take note of the unit variable. In most triggers using triggering unit, the line will be : `call SetUnitScalePercentAMHS(GetTriggerUnit(),100,100,100)`

Once again, this is used to change the unit's size ingame in replacement of the GUI line : **Animation - Change Unit Size**.

-=INGAME UNIT TINITING=-

To change a unit's ingame vertex coloring, i.e : **Animation - Change Unit Vertex Coloring**, the GUI line must be replaced using the **Action : Custom Script** with code :

`call SetUnitVertexColorBJAMHS(udg_UNIT_VARIABLE,100,100,100,100)`

where it's : `call SetUnitVertexColorAMHS(udg_UNIT_VARIABLE,TINT RED,TINT GREEN,TINT BLUE, TRANSPERANCY)` where 100 alpha = fully opaque and 0 alpha = fully transparent.

This is to change a unit's INGAME vertex coloring.

## Future Updates to Your Map

Any future updates made by your map now need to follow the template shown above. Any time you modify/create a new unit you must change its shadow/unitgraphic data in the **Shadow and UnitGraphic Configuration** and not the Object Editor. The same goes for any modifications to Abilities based off Bloodlust. Also in the future you cannot use the functions mentioned **Replacing Instances of GUI Functions**, instead you need to use the mentioned replacements