# GHOST RECON: ADVANCED WARFIGHTER 2

## - THE EDITOR -

## - v1.04 -

By:
**Grin_Wolfsong**

Assisted by:
**Grin_Ichabod**

## *Document Contents*

# *Chapter 1: Getting Started*

This tutorial will cover the basics of the editor. Not all the small details, but the basic stuff you need to know to create a new level from existing units.

The editor provided to the public is the exact same version that has been used by Grin during the development of GRAW2. It's not designed especially for public use, which many other editors released to the public are, and as such it has an interface that is a bit rough around the edges and not totally logical due to constant changes and additions to the game functions during development, but I'll cover all that and try to straighten things up a bit. It's also not fully stable so play it safe and save often.

## Requirements

The only requirements when working in the editor, besides that you have the hardware to run the game of course, is that you have a 4 button mouse. The thumb button [*mouse 4*] is important for placing units, and you can't rebind keys in the editor, so you must have it.

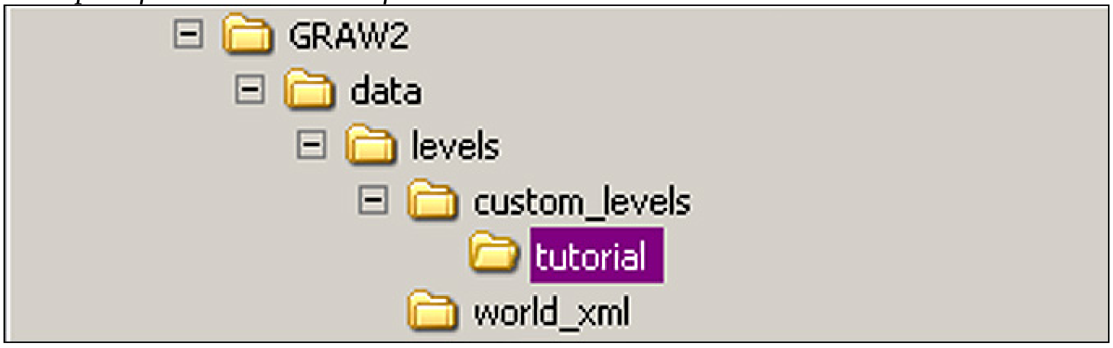
## Preparations

The simplest way to keep things clean is to start by defining a new level. If you don't do this a default level called "world_xml" will be used. In that case you'll then have to move your files into a new folder when you want to share your level, or else all levels exported with the default folder name will override each other and create problems for the players, so my suggestion is that you create a new level from the start. In this document the custom level will be called "tutorial".

Where the levels folders will be automatically created, when you first start the editor with a defined name, varies depending on if you are using XP or Vista. In XP they are created where your game is installed under GRAW2/data/levels/custom_levels. In Vista all the files are created under your profile, where a GRAW2 folder should be located with a similar underlying folder structure.

***Note:*** *Only use lower case letter for all folder and file names in GRAW2, and no special characters.*

*Example of new custom level folder called "tutorial" created in XP:*



To create this new level the best way in the long run is to edit the "`GRAW2_Editor.bat`" file to tell it what level you want to work with. If you know a little bit of BATCH programming you'll notice when you open it that you don't have to edit this file. It uses a wildcard which takes in the string written after the filename and uses it as the level to work with.

Still, I suggest that you create a copy of "`GRAW2_Editor.bat`" and then do these changes inside the copy as that will help you in the future to faster and easier start the editor with the right level or if you want to create more then one level you can have one start file for each of them. Just give the new copy a descriptive name, like "`GRAW2_Editor_Tutorial.bat`" for example as the level I'm using is called "`tutorial`", so you can easily find it. Inside the copy you should edit the last part so that it contains the name of your level, which will be the name of the level folder that will be created, after the "`-path`" tag and it should work fine.

*Contents of `GRAW2_Editor_Tutorial.bat`:*

```
graw2.exe -o context-editor.xml -path tutorial
```

With that done you are now ready to start building your level. Run the new "`GRAW2_Editor_Tutorial.bat`" start file and let's take a look at the first steps inside the editor.

# Chapter 2: Interface

Once the editor has started you are facing a blank world with a default environment and nothing else in it. In the upper left corner you can find “`layer`” and “`rendering`” options menus. On the right side you’ll find the “`unit list`” window, which shows available units to be placed in some layers and which units have been placed in others.

***Tip:** All windows inside the editor can be moved around by using the left mouse button, clicking on the top part of the window, hold down and move.*

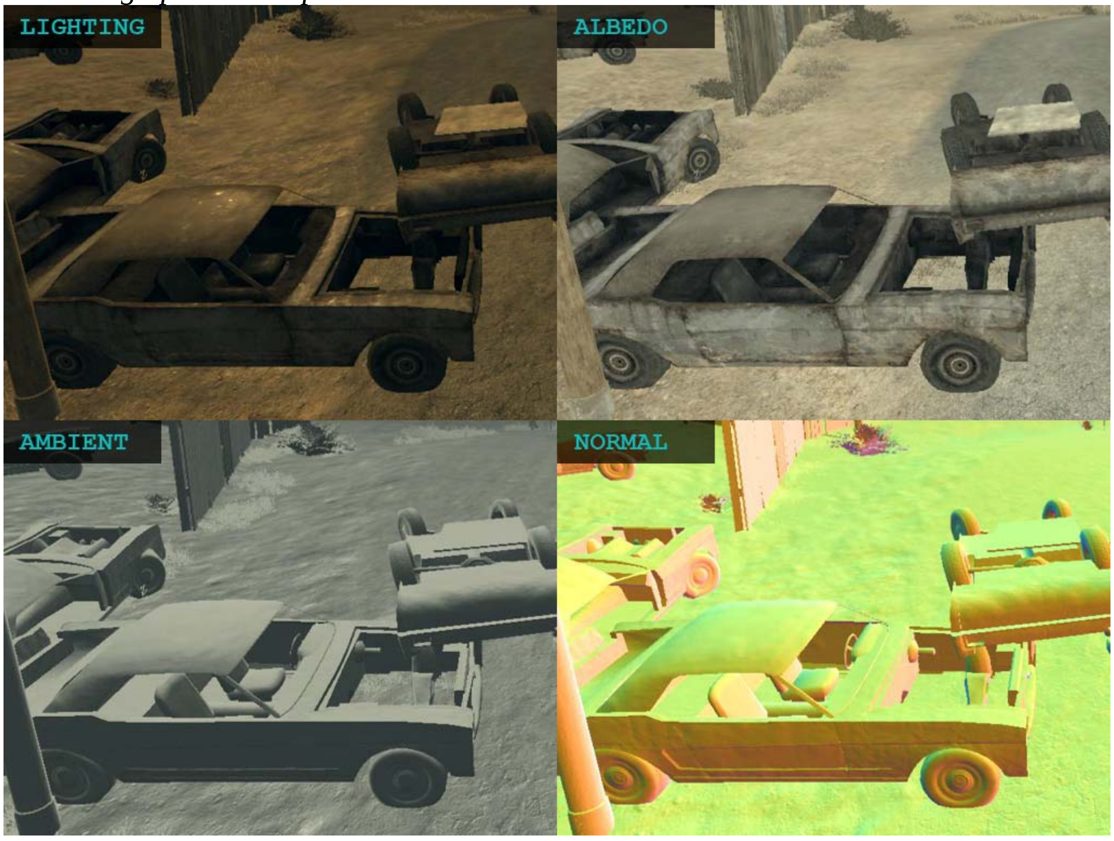
## Rendering

The “`rendering`” menu decides how you are currently watching the level inside the editor. This can be useful as you can for example turn to “`albedo`” and that way turn of the environment lighting and shadows to get a better look at what the layout of the level is even when light maps have not yet been rendered.

*Rendering options found in the editor:*

| | |
|---|---|
| `Albedo` | Shows the level in only diffuse textures. |
| `Ambient` | Shows the level in grey with only ambient shadows. |
| `Lighting` | Shows the level with the assigned environment, like it will look inside the game. |
| `Normal` | Shows the level in only normal map textures. |

*Rendering option examples:*



## Layers

The “`layer`” menu decides which units are shown in the unit list on the right side of the screen. It also automatically hides units from other layers then the one being worked on, with some exceptions for static and dynamic units, to make it easier to work. Also, only units from the current layer can be selected and edited.

*Layers found in the editor:*

| | |
|---|---|
| `Static` | Contains units like landscapes and props. Even props that can be destroyed are found under static, with the exception of destroyable vehicle props. |
| `Dynamic` | Contains working vehicles as well as prop versions of vehicles that can be destroyed, called covers. |
| `Markers` | Contains markers used for defining minimap borders, compass north direction, cinematic points and other things. |
| `Light` | Contains extra light sources. |
| `Sound` | Contains sound points used to place environment sounds. |
| `Lightmap` | |
| `Electric` | Contains all wires. |
| `Props Brush` | Contains brushes for grass, small debris, decals and similar. |
| `Human` | Contains all human AIs. |
| `AI Graph` | Contains the human and vehicle AI graphs. |
| `Locations` | Contains all zones. |

## Sub-Layers

Some layers have sub-layers as well, which allows it to only show parts of the contents of the layer in the unit list at the time. Like with layers, only units from the current sub-layer can be selected and edited.

*Keys for switching between `Static` sub-layers:*

| | |
|---|---|
| `Ctrl+1` | Static (default) |
| `Ctrl+2` | Small Static (includes destructible objects) |
| `Ctrl+3` | Landscape |

*Keys for switching between `Dynamic` sub-layers:*

| | |
|---|---|
| `Ctrl+1` | Cover (default) |
| `Ctrl+2` | Vehicles |

The `AI Graph` layer has a toggle function and will only show one graph at the time as working with two graphs showing would be very annoying.

*Key for toggling between `AI Graph` sub-layers:*

| | |
|---|---|
| `M` | Toggle between Human (default) and Vehicle graph. |

## Unit List

The contents of the “`unit list`”, and its existence, vary depending on which layer you are currently in. It has a “`mask`” input field which can be used to filter the contents of the current list. Filtering can only be made from the beginning of the unit name, and it doesn’t work with wild cards like “`*`” or “`?`”.

When in “`Light`”, “`Sound`” and “`Locations`” layers, the unit list contains units that have placed on in level already, but in all other layers where it’s available it holds all the units you can select to create.

## Commands

The editor is controlled by keyboard shortcuts only, with the exception of the two menus covered earlier. There is no way around using these, so you simply have to learn to use them like Grin has.

| | |
|---|---|
| `Alt + F4` | Quit Editor |
| `Ctrl + S` | Save Level |
| `Ctrl + L` | Load Latest Save |
| `Ctrl + Backspace` | Generate Lightmaps |
| `Alt + Backspace` | Generate Silhouettes |
| `Ctrl + K` | Calculate AI Graph |
| `Ctrl + F` | Calculate Cover Points (after calculated AI graph) |
| `L` | Spawn Player (at current tool indicator location) |
| `O` | Generate Orthographic Minimap (requires markers) |
| `Shift + O` | Generate Cube Map (require marker) |
| `F6` | Toggle Editor Light - On / Off (default) |
| `F8` | Toggle Mode - Editor (default) / First Person (require spawn) |
| `F12` | Show Editor Console (provides some feedback) |
| `Numpad -` | Toggle Unit Info Mode  (uses FFM controls) |
| `Numpad /` | Toggle Render View Stats Window |

| | |
|---|---|
| `Alt + 1 (or 2)` | Static Layer |
| `Alt + 3` | Electric Layer |
| `Alt + 4` | Props Brush Layer |
| `Alt + 5` | Human Layer |
| `Alt + 6 (or 7)` | Dynamic Layer |
| `Alt + 8` | Markers Layer |
| `Alt + 9` | AI Graph Layer |

| | |
|---|---|
| `Space` | Enter / Exit “Free Flight Mode” (FFM) |
| `W` | Move Forward in FFM (camera coordinates) |
| `S` | Move Backward in FFM (camera coordinates) |
| `A` | Move Left in FFM (camera coordinates) |
| `R` | Move Right in FFM (camera coordinates) |
| `E` | Move Upward in FFM (world coordinates) |
| `Q` | Move Down in FFM (world coordinates) |
| `Mouse Movement` | Rotate Camera |
| `Scroll Forward` | Increase FFM Move Speed |
| `Scroll Backward` | Decrease FFM Move Speed |

| | |
|---|---|
| `N` | Toggle Place Alignment - Normal (default) / Grid |
| `G` | Increase Grid Size |
| `Shift + G` | Decrease Grid Size |
| `Hold TAB` | Show Grid Plane |

| | |
|---|---|
| `Mouse 1` | Select Unit |
| `Mouse 2` | Place Unit |
| `Mouse 3` | Snap Rotate Selected Unit (45 degree ccw per click) |
| `Mouse 4` | Move Selected Unit (along Normal or on Grid) |

| | |
|---|---|
| `B` | Pick Selected Unit Type (to use for “place unit”) |
| `Home` | Reset Unit X & Y Rotation |
| `Del` | Remove Selected Unit |
| `Up Arrow` | Free Rotate Selected Unit, Y Axis (object coordinates) |
| `Down Arrow` | Free Rotate Selected Unit, Y Axis (object coordinates) |
| `Left Arrow` | Free Rotate Selected Unit, Z Axis (object coordinates) |
| `Right Arrow` | Free Rotate Selected Unit, Z Axis (object coordinates) |
| `Shift + Up Arrow` | Free Rotate Selected Unit, X Axis (object coordinates) |
| `Shift + Down Arrow` | Free Rotate Selected Unit, X Axis (object coordinates) |
| `Ctrl + Up Arrow` | Move Selected Unit Upward (world coordinates) |
| `Ctrl + Down Arrow` | Move Selected Unit Downward (world coordinates) |

That’s the controls to use. With that I think it’s time to move on to start creating a level by taking a closer look at the `Static` layer.

# *Chapter 3: Static Layer*

The main parts of any level are static units. Landscapes, buildings, plants and props are all included in this category. Some have destructible, or dynamics versions which are more expensive to use when it comes to performance, so try to use their static counterparts as often as possible without sacrificing game play to gain better FPS.

When combining static units there come always the problem that you can't load all textures into memory at the same time. Because of this GRAW2 uses texture scopes in GRAW2, which are defined per level and decided which texture atlases to use when playing each specific level. Unlike in GRAW1 these will not be predefined for you. The editor has access to all textures, so you won't get any notice about this problem in there, but once you play test the level inside the game you'll get a blue and yellow checker pattern where textures are missing. If this occurs you have the option to either remove those props and maybe replace them with something else or expand the texture scope. I'll cover how to edit the texture scope later in chapter 13.

## Landscape

The first thing you'll have to add to any level is a landscape static unit, which will act like the ground for the players to walk on. The available landscape units are found in the `Static` layer [*`Alt+1`*] under the `Landscapes` sub-layer [*`Ctrl+3`*].

Many of the original SP/Campaign Coop landscapes have a built-in low-poly backdrop which can't be removed. Those low-poly houses also have low texture quality and lack collision, so avoid designing a level among them. ;) The landscapes also have roads and other ground deformations built in especially for the original game design, which can't be removed either obviously, but besides that they can be used to build entirely new levels on. Some landscapes, like for mission01 and mission07, have objects that are built to fit into the landscape very exactly, but are not spawned with it when you place the landscape in the editor. If you want to use such a landscape I suggest copying the files from the level using that landscape and cleaning it of the units you don't want which is fastest done inside the `world.xml` with an xml editor, as for example trying to align the bridge in the mission01 landscape inside the editor is virtually impossible.

Modders will be able to create their own landscape units. They need to be modelled and UV-Mapped inside 3DS Max, then exported and setup with the required XML files, upon which they will appear in the list among the other landscapes. That process will not be covered in this tutorial.

Once you have selected which landscape you want to use, let's move on to other sub-layers.

## Static

Back in the `Static` sub-layer [*`Ctrl+1`*] you'll find all large static units like for example houses and huge billboards. These are the main building blocks used to form the level and create paths that the player can use. There is nothing really special about these, so it's just to pick what you want from the list and start building.

## Small Static

In the `Small Static` sub-level [*`Ctrl+2`*] you'll find all the smaller props that can be used to add detail to your level. In here you can also find placeable effects (use mask with "`efx`"), vegetation (use mask with "`vgt`"), and some usefully level design tools like mover collisions to prevent players from going somewhere (use mask with "`mov`"), the cover dummies for extra the AI cover points (use mask with "`cov`") and the ambient dummy for extra ambient sample points around props that appear black (use mask with "`amb`").

There are also static versions of some vehicles here. The difference between these and those found in the `dynamic` layer is that these can't be destroyed or moved in game, and as such can be used to block paths you don't want the player to go to or to add more atmosphere where the player can look out in the near surrounding outside the level and it would be unnecessary to have an performance expensive dynamic vehicle.

Although there are no dynamic vehicles in this layer that doesn't apply to other props. Many small props come in two versions but when that's the case, the naming convention varies some. One of them always has its type defined if there are two versions though. If there is one version called "`_dynamic`", that is obviously dynamic and the other is static, and where there is one called "`_static`", that one is obviously static and the other dynamic. Don't use dynamic props where the player can't get to or where you want to block the player from going. Dynamic props are much more expensive to use and should be avoided on MP levels other then Campaign Coop where there are few players as they have to be synced between the players and many dynamic units will take a lot of bandwidth.

Besides destructible dynamic units, there are also non-destructible units that are affected by the game physics, most notably the wind and collision with characters. Where there are optional versions these have "`_cloth_high`", "`_cloth_low`" or "`_wind`" in their names. Like other dynamics objects, try to avoid these in MP levels other then Campaign Coop.

Lastly there are AGEIA hardware props (use mask with "`xag`"). These are very CPU expensive to use and should be avoided unless the map is made for AGEIA hardware users. People with top of the line CPU can probably handle these anyhow, but make sure to test the level on such a computer if you want to try and use them without an AGEIA card.

That should cover the entire `Static` layer. Let's move on to `Dynamic`.

# *Chapter 4: Dynamic Layer*

In the `Dynamic` layer [*`Alt+6`*] you'll find all the vehicles.

When placing dynamic units you'll notice that they interact with the environment. Hitting static units will make them rotate and they will also push other dynamic unit around. From my experience the best way to place a dynamic object is to use FFM and move as close to the position you want to place the unit at as possible, leave FFM, select the unit in the list and place it carefully a little bit above the ground. It will fall down and adjust itself to the shapes it now stands on. Avoid selecting the unit in the list first and try to fly it in with FFM.

## Cover

In the `Cover` sub-layer [*`Ctrl+1`*] all the dynamic vehicles that don't have any AI or animated sequences can be found. Most are just used as normal props, but there are some special versions found here as well.

Some of these have built it fire effects, which are of course more expensive to use then the versions without fire. These have "`burning`" in there names. Don't use too many of those on a level.

There are also heavy physics versions of the M1078 truck, whose covers are affected by the wind. These have "`_high`" and "`_low`" in their name and should be used with care like all other units for heavier physics settings.

Other then that, simply use these dynamic units like static units, but make sure it's ok that the player can move them from the position you have placed them.

## Vehicle

In the `Vehicle` sub-layer [*`Ctrl+2`*] we find all the vehicles that have AI or are pre-animated sequences. To make use of the AI vehicles you must have a Vehicle AI Graph on the map.

Vehicles also need room if they are to move, so think about that when placing you props and designing your level. They require clearance of about their double width to be quite sure that they can pass narrow passages. They also require clearance upwards which is calculated by the biggest vehicle in the game, so don't place to much overhanging details where you want the AI vehicles to be able to move.

Important to notice on vehicles is that they have a few options that affect how they are used in the game.

First they need a unique name so you can call it from within your mission script to spawn them if they aren't spawned from start, remove them from the game world, destroy them or use them inside conditions for triggers.

Second they need to be in the correct team slot, “`friendly`”, “`hostile`” or “`neutral`” so they interact correctly with the player and hostiles.

Lastly they need to be prepared for how they are going to be used, a bit more specifically how they are going to start the mission and if the player is going to be able to interact with it. “`Sequence spawn`” starts the vehicle hidden from the world and it has to be activated through the mission script. “`Sequence death`” starts the vehicle as destroyed when spawned. “`Enter_exit`” sets if the player will be able to enter or exit the vehicle and it will show in the game by displaying the “`press x to enter`” or “`press x to exit`” messages to the player once close to an entry or exit point.

## Orders

You’ll notice that you can’t place paths for the vehicles to move along inside the editor. All orders for vehicles have to be added in the mission script. Some of them will require coordinates, which can be gotten in a few different ways.

The first is to use the `Stats Window` [*`Numpad /`*], which among other things show the coordinates that the camera is currently at. This can be used to get a rough idea of the coordinates you want.

The second is to place units on the map and giving them a specific name like “`vehcile1_waypoint_a`” and so on and then saving the map. Exit the editor and open the `world.xml` in your XML editor and to a search for the name you gave the unit. In its data you’ll find the exact coordinates it was placed on, which you can then use inside the mission script. Remember to remove those dummy units later.

It’s up to how exact you want to the coordinates to be, as well as if it’s a flying vehicle or a ground based one of course. You’ll always have to test the paths in-game to make the final adjustments in the end. When doing that, remember that vehicle movements are FPS dependant. They operate like they should at about 35 FPS and up. Under that they will start to deviate from the set path as they won’t reach the given waypoints in time.

That’s all the info I can give you here. Let’s check out the `Electric` layer.

# ***Chapter 5: Electric Layer***

When you get to the `Electric` layer [*`Alt+3`*] you'll see… nothing! No "`unit list`" or other window that gives us any clue on what to do. Well, don't worry because it's really simple. The only thing that can be done in this layer is placing wires to make the level look more inhabited, to simulate electric wires or telephone wires for instance.

This is done by right clicking [*`Mouse 2`*] where you want the wire to start, then right clicking [*`Mouse 2`*] where you want the wire to end. While the wire is still selected you can adjust the slack on the wire by using [*`Up Arrow`*] to decrease slack and [*`Down Arrow`*] to increase slack. If you want to adjust the slack later on a wire, just select the wire with a left click [*`Mouse 1`*] and use the arrow controls. You can't move the start and end points once they are placed though so if you want to move those simply select and delete the old wire [*`Delete`*] and create a new one.

With nothing more to cover here, let's move on to the `Sound` layer.

# *Chapter 6: Sound Layer*

When you get to the `Sound` layer (which doesn't have any keyboard shortcut), we find an empty "`unit list`". This is because there is only one type of unit to place in this layer and the "`unit list`" is instead used to list all the placed sound units currently in the level.

To place a new sound unit, simply right click [*`Mouse 2`*] and it will appear at the location where the camera is currently at. To remove a sound unit, select it in the "*`unit list`*" by left clicking on it [*`Mouse 1`*] and hitting [*`Delete`*].

For each sound you place you are given the option to give it a "`name`", which is only used if you want to be able to reach it inside the mission script so it's not commonly used, and a "`cue`". The "`cue`" should be given the name of the sound you want to play. For a full list of the available sound cues in the original game, check out appendix 1 at the end of this tutorial, besides all the sound cues it also lists which sound banks each sound belongs to.

Now that the level has been given some more environment feelings, let's move on to the `Props Brush` layer.

# Chapter 7: Props Brush Layer

The `Props Brush` layer contains grass; small debris units and decals used to add the final touch to the levels. These are useful to remove repetition in larger units in the level, and also add a more used feeling in the level.

These units are "painted" onto the level and are not found inside the "`world.xml`" like the others. They are stored inside an encrypted "`massunit.bin`", so if you want to remove all units placed with this tool, simply remove that file before starting the editor (or to be safe always save a copy somewhere in case you change your mind), and they will all be gone once the editor has been started.

As this layer has a special brush tool, it also has special tool settings to control it.

*Brush Controls:*

| | |
|---|---|
| `Mouse 1` | Paint units. |
| `Mouse 2` | Erase units. |

*Tool settings:*

| | |
|---|---|
| `Random Roll` | Allowed random rotation of painted units. |
| `Radius` | Size of brush to paint or erase units with. |
| `Density` | Amount of units to paint per square meter. |
| `Pressure` | Brush pressure. Max pressure paints entire density directly. Lower pressure paints gradually up to density setting. |
| `Use Pressure When Erasing` | Toggle to use pressure value when erasing units. |
| `Height` | Sets height of brush to create a volume when erasing. |
| `Angle Override` | Locks angle to camera rotation. |

In this layer the `unit list` contains all the units that can be painted onto the map with the brush tool. The mask input field works exactly the same as in the other layers to allow you the limit the units displayed, for instance input "`vgt`" to get small bushes and grass. You can select as many units as you want at the same time by holding down [`Ctrl`] while selecting. When you have more then one unit selected the brush will place them randomly within the given brush size and each of them will be given the random rotation set in the tool settings.

One important thing to notice is that there are three versions of each grass type. These clip away at different distance from the player. It's best to combine these where you want grass so that it gets thicker the closer you get to the grass patch. Still you have to be careful not to overuse the long range grass as it will reduce the frame rate a few steps. Try not to paint grass to think in general, balance it.

Now the level should look detailed enough so let's take a look at the `AI Graph` layer in case the level is going to be used in a game mode with AI soldiers or AI vehicles, otherwise you don't need to place those.

# *Chapter 8: AI Graph Layer*

The `AI Graph` layer [*`Alt+9`*] consists of two sub-layers. The default is the `Human AI Graph` sub-layer which is most common to use, but it also contains the `Vehicle AI Graph` sub-layer which you can toggle too by pressing [*`M`*].

***Note:*** *AI Graphs are only needed on maps that use human or ground vehicle AI units, so for normal MP game modes no AI graph needs to be placed.*

An AI Graph is a network of nodes, or navigation points for the AI, connected by paths that the AI can use to travel between the nodes. Only one such network of each type is allowed on each map. In other words all nodes in the `Human AI Graph` must connect to each other, and all nodes in the `Vehicle AI Graph` need to connect to each other. The connecting paths have a limited range, which is indicated by the circle surrounding the mouse pointer while in this layer. This is so that they don't try to connect to nodes to far apart. When placing each node, take notice on how it automatically connects the surrounding nodes and make sure that each of the new paths don't go through any object on the map or else the AI will think it can go there even though it can't and will end up walking towards the object and getting stuck.

*AI Graph Controls:*

| | |
|---|---|
| `Mouse 1` | Select node. |
| `Mouse 2` | Create node. |
| `Mouse 4` | Move node. |
| `Delete` | Delete selected node. |

The `Vehicle AI Graph` allows for longer distance between each node as vehicles need more space to move. When placing their AI Graph you should make sure that there is enough space surrounding all the paths between the nodes for at least 2 vehicles in width. Also make sure that there is nothing hanging low over the path as the vehicles require good clearance to come through.

The required density of the `Human AI Graph` depends on the surrounding area. The AI will use the node points as primary stopping points when executing given orders, so in open areas the distance between nodes should be longer to allow the AI to move over that area faster. While close to objects the density should be higher, especially close to corners when there should be a navigation node on each side because of the cover system. The cover system is build up by cover points placed in the units before they are exported from Max. Each cover points that doesn't have a navigation node within an engine defined radius, will be removed from use, all others are automatically connected to the navigation point within their radius at level start so the AI can use them.

The `Vehicle AI Graph` should never have high density, as mentioned before the vehicles need more space the turn and move, and they also don't use cover points.

Make sure that both the `Human AI Graph` and the `Vehicle AI Graph` covers all areas of the map where the friendly or hostile AI of either type can move. The AI needs to find nodes close to them so they know where they are allowed to go, or else they won't move. Flying AI doesn't use the AI graphs.

## AI Graph Generation

Once you have placed your graph it has to be calculated and encrypted into the `ai.gph` and `ai_course.gph` files that the game will use. This is done by pressing [*`Ctrl + K`*]. Now you'll think that your computer has locked up, but it hasn't. Calculating the AI graphs takes a little while and there is currently no feedback on how far it has come, not even inside the editor console [*`F12`*]. But once you can move the mouse cursor again, it's done and ready to be used in-game.

Make sure that the files created by calculating the AI graphs are connected to the level inside the `world_info.xml` file, which is covered in chapter 13, or else they will not be used.

## Cover Point Generation

Once the AI Graph has been calculated you can also generate extra cover points [*`Ctrl + F`*] that are based on the landscape itself, which will end up in a file called `coverpoints.xml` in your level folder. This is not needed on city like maps that are quite flat and have many buildings, because buildings already have built-in cover points on all corners and other good cover positions, which will be used automatically as long as there are AI graph nodes relatively close to them so they AI can find them.

This function is created to generate cover points on more open landscapes like rural settings, and must have a calculated AI graph to work and as such is not needed on levels that are not built to be played without AI units.

***Note:*** *What the editor does is look for variations in the level geometry around the AI graph nodes and send out rays from such positions to check for good cover points that the AI may be able to make use of.*

Just like with the AI graphs you have to make sure that the extra cover points are connected to the level inside the `world_info.xml` file, or they will not be used.

With that I think we should move on to the `Human` layer.

# Chapter 9: Human Layer

The `Human` layer [*`Alt+5`*] contains all the human AI, hostile and friendly depending on which unit you use. Each human node placed represents a group of AI units, whose amount of soldiers varies depending on the selected group and is indicated by the small red globes above their node. What units and what equipment they have, as well as if the group is friendly or hostile, is defined in a file called “`group_manager.xml`”, and can not be changes inside the editor. Once a human node has been placed or selected you get an option box for it, in which you select what characteristics the group has.

This layer reuses the same controls for different things depending on which order type the group has been given and which mode you’re in.

*Human controls:*

| | |
|---|---|
| `Mouse 1` | Select node. |
| `Mouse 2` | Create node. |
| `Mouse 4` | Move node. |
| `Delete` | Delete selected node. |

*`Sniper`, `SniperKneeling` and `SniperStanding` order controls:*

| | |
|---|---|
| `Mouse 3` | Place cover cone center. |

*`Guard` order controls:*

| | |
|---|---|
| `P` | Place guard zone. |
| `Mouse 3` | Place guard zone radius (after P has been pressed) |
| `Mouse 4` | Move human node. |

*`Patrol` order controls:*

| | |
|---|---|
| `Insert` | Toggle to patrol path mode. |
| `P` | Place first patrol path node |
| `Mouse 1` | Place patrol path node |
| `Mouse 3` | Remove last patrol path node in chain. |
| `Mouse 4` | Move human node. |

*Human node options:*

| | |
|---|---|
| `Name` | Name of group (not needed). |
| `Group ID` | The ID of the group which is used to call it inside the mission scripts. This is important as you won't be able to activate the group unless you know the name and all groups with the same name will answer to the same script calls including condition checks. |
| `Order` | How the group will behave once spawned. "`Patrol`" will activate "`patrol type`" options. |
| `Crew` | Define if the group is the crew of a vehicle. If they are they will spawn with that vehicle and no additional script command will be needed. |
| `Transport ID` | If the group is defined as crew you have to give the ID of the transport they belong to. |
| `Group Type` | Defines which group template to use. The template defines number of unit in group, each unit's equipment and looks as well as the group alignment. |
| `Patrol Type` | When a group is set to "`patrol`" in "`order`", you here define which type of patrol it should be. |

*`Order` types:*

| | |
|---|---|
| `Guard` | Will move around inside given zone. |
| `Patrol` | Will use the given patrol type. |
| `Sniper` | Will cover given cone area. |
| `SniperKneeling` | Will cover given cone area always kneeling. |
| `SniperStanding` | Will cover given cone area always standing. |
| `None` | Will just stand still on the spot. |

*`Patrol` types:*

| | |
|---|---|
| `Loop_Idle` | Follow patrol nodes in order and when at the end start again with the first node to create an infinite loop. AI will act relaxed. |
| `Loop_Recon` | Follow patrol nodes in order and when at the end start again with the first node to create an infinite loop. AI will act alert and ready. |
| `Moveguard_Idle` | Follow patrol nodes in order, which end in a guard area that the AI will move around inside at random. AI will act relaxed. |
| `Moveguard_Recon` | Follow patrol nodes in order, which end in a guard area that the AI will move around inside at random. AI will act alert and ready. |
| `Pingpong_Idle` | Follow patrol nodes in order and when at the end follow them back in reversed order to create an infinite oscillating path. AI will act relaxed. |
| `Pingpong_Recon` | Follow patrol nodes in order and when at the end follow them back in reversed order to create an infinite oscillating path. AI will act alert and ready. |

The `Guard` order uses a guard zone, which defaults to the groups' position with a set radius. It can easily be moved by pressing [`P`] to define the new center of the zone, followed by using the middle mouse button [*`mouse 3`*] to set the outer limit of the zone. You can also move the group away from the zone by using the thumb mouse button [*`mouse 4`*] where you want to place the group. When the group starts outside the guard zone they will begin by running to the zone as fast as possible. If you want them to take it slow and follow a specific path you should give them a "`moveguard`" patrol order instead.

The `Sniper`, `SniperKneeling` and `SniperStanding` orders allows you to define a cone for the AI to cover. This sniper cone can be adjusted going pressing the middle mouse button [*`mouse 3`*] at the point where the center of the cone, which the sniper should concentrate at covering, should be. The difference between the different sniper orders is only that you can force the AI to fold a standing or crouched stance to prevent undesired stances in situations where they would create clipping of objects. If the normal order is given the AI will act on its own.

The `None` order will simply leave the AI standing on the spot you place them.

The Patrol order allows you to define a patrol path which the AI will always try to follow unless they have detected the player team. This is done by going into the `patrol path mode` [*`Ins`*] with the unit selected and placing patrol nodes that automatically becomes connected into a patrol path. By defining which patrol type it should be you can decide how the AI will use the patrol path. Either they will follow it until the end and then follow it back to the beginning, and continue like that forever, which is the "`pingpong`" version. You can also make them "`loop`" the path, which connects the first and last nodes into a never ending path. The last option is the have the AI travel along the path until the end and then set up a guard zone for them there, which is the "`moveguard`" version.

Make sure when making patrol paths that they don't go through objects on which the AI can get stuck. All patrol types are carried out at approximately the same speed. In the alert versions, "`recon`", they move a little faster then in relaxed, "`idle`", but as they also take cover very often which ends in a movement speed which is quite equal.

There isn't much more then that involved in placing humans, so let's move on to the `Locations` layer.

# Chapter 10: Locations Layer

The `Locations` layer is used to place zones or areas that can be used to either detects things and through that act like triggers, or for pointing out targets for some of the scripting events.

In this layer the "`unit list`" is used to list all the locations currently placed on the level for quick access to them. There is also a special settings window in this layer where you can select which location shape tool you want to use when placing the location, as well as other settings for the locations themselves.

*Location Settings:*

| | |
|---|---|
| `Shape` | Set which shape tool to use when drawing the current location. |
| `Altitude` | Used to adjust the altitude of the location once it's been placed. |
| `Name` | The name of the location which is used inside the missions scripts to make use of it. |
| `Height` | The height above the bottom plane for the location. Used for "`box`", "`cylinder`" and "`polygon`" shapes. |

*Location shapes:*

| | | |
|---|---|---|
| `box` | `sphere` | `Circle` |
| `cylinder` | `square` | `Polygon` |

"`Square`" and "`circle`" are 2D locations that are best to use on flat surfaces. "`Sphere`" is a 3D volume the same height as width. "`Box`", "`cylinder`" and "`polygon`" has a "`height`" variable that defines how high above the placed area their 3D volume should cover.

*Location Controls:*

| | |
|---|---|
| `Mouse 1` | Place location. |
| `Mouse 2` | Finalize selected location and create new location. |
| `Mouse 4` | Move location. |
| `Delete` | Delete selected location. |

Selecting already placed locations can only be done through the "`unit list`" window by using the left mouse button [*`mouse 1`*].

**Warning**

The "`polygon`" locations tool is very unstable. Avoid using it if possible. If you have to use it, save as soon as you've completed placing a location as the probabilities of the editor crashing soon after or during are very high.

That's it for locations; let's move on to the `Makers` layer.

# *Chapter 11: Markers Layer*

The `Markers` layer [*`Alt+8`*] is used to place markers to define minimap borders, compass north direction, cubemap rendering spot and cinematic camera points. Once you place a marker you'll get a settings window which contains a list of the available versions so that you easily can set the marker to the desired type.

*Marker Controls:*

| | |
|---|---|
| `Mouse 1` | Select marker. |
| `Mouse 2` | Create marker. |
| `Mouse 4` | Move marker. |
| `Delete` | Delete selected marker. |

*Markers Settings:*

| | |
|---|---|
| `Name` | Name of the marker, which is important for animation and cinematic markers so they can be referred to from the mission script. |
| `Type` | What the marker should be used for. |
| `Altitude` | The markers altitude in the world. |

*Marker types:*

| | |
|---|---|
| `Animation_Point` | Used with the "`PlayCustomAnimation`" script element. See the scripting tutorial. |
| `Camera_Cinematic` | Used to position the camera in cinematic. |
| `Camera_Team_Select_A` | |
| `Camera_Team_Select_B` | |
| `Compass_North` | Coordinate to the north heading on the map. |
| `Cubemap` | Coordinate to generate cubemap from. |
| `Minimap` | Defines borders of the map for the minimap. |
| `Target` | Used for aiming the camera in cinematic. |

The most common use on this layer is the definition of the "`minimap`" borders. These coordinates are important as they are used to calculate the icon locations on multi player minimaps. They are also used by the minimap generation as that it also corresponds to the correct coordinates. You can place any number of minimap markers around the edge or the area that the player can use, but it will only use the extremes when creating the minimap. In other words, the highest and lowest x and y coordinates among the minimap markers. Those values have to be entered into the maps `world_info.xm`l as well, which will be covered later in chapter 13, so that the character positions can be calculated.

The second most common use of the markers is to define the "`compass north`" direction. This is needed for the compass on the player HUD to work properly. Simply place it outside the map in desired direction.

The “`cubemap`” marker is used to define the position from which the levels cubemap should be generated. The levels cubemap is used by all shaders where the environment is reflected on surfaces, like windows and weapon scopes.

The “`animation point`” markers are used to define points to play custom animations at through the `mission.xml` during cinematics.

Final there are the marker types used by the cinematic scripting in the `mission.xml`. “`target`” is used to point the camera at something, while “`camera_cinematic`” is used to define camera paths and locations. For more info on where these are needed, look for the cinematic events inside the “*`GRAW2: Scripting for beginners`*” tutorial.

That covers all the layers in the editor. Remember to save often in the editor as it is a little unstable. Each time you save a new incremental “`world_X.xml`” file is created which you can always go back to by using it to replace the “`world.xml`” file in your level folder, so each save creates a new backup.

Now let’s take a look at some other functions inside the editor which you’ll most likely need to use.

# *Chapter 12: Additional Editor Functions*

There are some built in functions in the editor which aren’t layer dependant, and this chapter will cover those.

## First Person Mode

To quickly be able to test out if the placement of units is good for the game play, if it possibly blocks the player from passing through a passage that you had planed to allow the player to use or if the player can get outside the map somehow, you can spawn in a player character and go into first person mode inside the editor. This is done by first placing the mouse icon where you want to spawn the character followed by pressing [`L`]. Now you’ll se the player character standing there on the map, which can also be useful to check heights while building obstacles and such things. To get into First Person Mode you simply press [`F8`] and now the controls are just like when you control the character inside the game and you’re free to run around and shoot at the environment. You can at any time press [`F8`] again and go back into Editor Mode to continue working on the level or change anything you’ve noticed.

## Minimap Generation

Once markers have been placed around the border of the playable part of the map, generating the minimap is quick work by pressing [`O`]. You should have lightmaps and silhouettes already generated when creating the minimap, or it will turn out looking strange.

***Note:*** *When doing this you need to have the editor resolution (which is the same as the game resolution set inside the game, not inside the editor) needs to be at least* `1024` *pixels in height, or you’ll get an error message saying your resolution is too low for the generation to be successful. This makes the* `1280x1024` *resolution found under the* `5:4` *aspect ratios a good option as the lowest setting to allow minimap generation.*

The output files from the generation can be found inside the level folder, for example inside the folder called “`tutorial`” use in this tutorial. In there you’ll find two versions. The first is the map that is ready for use, which you can also spice up inside any texture editing software of your choice like the original minimaps. The second is the same map but generated with all the locations on it. This is to help when editing the map for the Siege and Hamburger Hill game modes as those need visual representations painted onto the minimap texture for their game critical locations so the players can find them.

The minimap textures can be located inside the level folder itself and doesn’t need to be placed in the textures folder like for the original maps. Which minimap to be used with which game mode and the paths to find each minimap, is set inside the `world_info.xml`, and we’ll take at how to set that up in chapter 13.

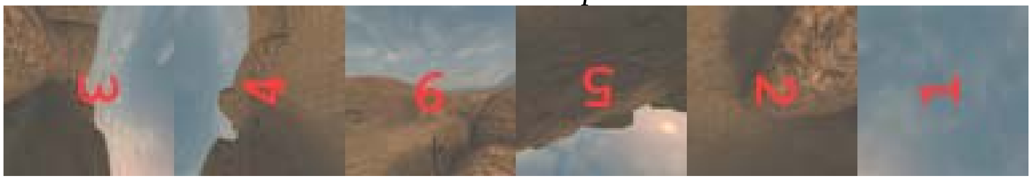
## Cubemap Generation

The cubemap is used to create reflections of the environment in surfaces that have been assigned shaders with that property, which include objects like windows and weapon scope glass. You should have the lightmaps and silhouettes generated when creating the cubemap or it will turn out looking strange.

To generate the cubemap [*Shift + O*] for a level you first need to have placed a marker for it, from which it will generate 6 pictures needed to create an environmental cube.

The generated cubemap will end up in a folder called "cubemaps" under your level folder, which would be in "data/levels/custom_levels/tutorial/cubemaps" for my example. You'll have to assemble a cube map from these 6 pictures by combining them in a picture editing software of your choice. They should be lined up on a single 384x64 pixel texture in a specific order and with specific rotations. There was never an automated script done for this so we can't provide one. The example below is of a cube map assembled for the level Nowhere and the red numbers are the names of the generated textures.

*Exact order and rotation needed on the cube map:*



*Needed rotations on generated textures and order to be placed on new texture:*

| | |
|---|---|
| Cubemap 3 | Rotate 90 degrees clockwise |
| Cubemap 4 | Rotate 90 degrees counter clockwise |
| Cubemap 6 | Shouldn't be rotated. |
| Cubemap 5 | Rotate 180 degrees |
| Cubemap 2 | Rotate 90 degrees clockwise |
| Cubemap 1 | Rotate 90 degrees counter clockwise |

You should also level the new texture to make it brighter and with lower contrast. In Photoshop for example you should use the "Brightness/Contrast" tool with brightness set to around +35 and contrast to around -20.

Once done, save the cubemap with the name "cube" as either a DDS (DXT5) or a TGA file into the cubemap texture folder dedicated to your level. It's found under "data/textures/custom_levels/" followed by your level name and then inside its "cubemaps" folder, named "data/textures/custom_levels/tutorial/cubemaps" for my example. This folder is already connected to the levels default texture scope and should work right away in the game.

***Note:*** *The reason the generated picture are not put into the correct folder is that you don't want to load over4MB of textures into memory by mistake. The way it is now that can't happen.*

## Lightmap Generation

Lightmaps are used in GRAW2 for larger units to reduce the system resources needed to generate dynamics shadows for everything.

They need to be regenerated after objects that are not using dynamic shadows, which are usually larger units, have been added, deleted or moved around. They also have to be regenerated if ambient dummies have been added to lighten up a prop.

Ambient points are included in the landscape meshes as well as those for larger units. They are used to set the light tone of props placed close to them by sampling how dark the lightmap is at the ambient point's position and then adding that to the props. For this to work there has to be at least one ambient points close to each prop or the props missing an ambient point close by will turn dark. As you can't go into 3DS Max and add ambient points exactly where you need then when these issues arise, there is an ambient dummy that can be found in the `Static` layer [*`Alt+1`*] under the `Small Static` sub-level [*`Ctrl+2`*] by using "`amb`" in the mask filter. Simply place one of these dummies close by any unit that is dark and then regenerate the lightmaps so that the dummies samples in the values to use, and that should solve the problem.

Lightmap generation starts to take very long time, so only do it when you absolutely need it. You can test the level without lightmaps, or with old lightmaps, as they don't have to be generated they you want to create a bundle like it was in GRAW1. The bundle tool is now a separate tool in GRAW2, which is not covered in this tutorial. When you are about to generate a lightmap I suggest that you first turn on the Editor Console [*`F12`*] as that is the only place you'll get feedback on the lightmap generation process. Then turn it on [*`Ctrl + Backspace`*] and go to bed or visit a friend.

***Note:*** *Lightmap generation at Grin usually took between 1 and 4 hours depending on the level size and number of units on it, with render farms using 4 or 5 computers.*

***Tip:*** *If you want to generate a faster lightmap you can edit the lightmap settings inside `sb_global.xml`. Look for the two lines called "`lightmap_render_quality`" and "`lightmap_size_quality`", and then lower them from "`medium`" to "`low`". This should cut the rendering time to about 20-25% but with low quality. It can be good to just while testing and evolving the level, and then setting it back to "`medium`" before generating the final lightmap before distribution.*

Once the lightmaps are generated they are located in a folder called "`lightmaps`" under "`data/textures/custom_levels`" and your level folder, which would be in "`data/textures/custom_levels/tutorial/lightmaps`" for my example level. This folder is already linked to the levels texture scope so that they are visible inside the game and also in the editor directly after generation.

***Note:*** *It's optional for you to create an atlas of the generated lightmap with the "`atlasgen`" tool. This will make your lightmaps smaller and also add to the game play performance of your map. Look in the atlas generator folder for more info on how to use it. Once an atlas has been generated you should remove all the old TGA files or they will still be loaded into memory. Copy them to another folder to begin with so you don't have to regenerate them in case you get problems with the atlas lightmaps.*

## Silhouette Generation

GRAW2 uses a silhouette system, which are the textures for the second LOD step with baked in lightmaps. These needs to be generated for each map or the second LOD step will look like blue and yellow checkers, which is not caused by errors in the texture scope.

The generation is easy to do, simply press [`Alt + Backspace`] and the process will begin just like with lightmaps. Once the silhouettes have been generated they will be located in a folder called "`silhouettes`" under "`data/textures/custom_levels`" and your level folder, which would for my example level be in "`data/textures/custom_levels/tutorial/silhouettes`". That folder is already linked to the levels texture scope so that they should be functional inside the game directly after generation.

***Note:*** *Just like with the lightmaps, it's optional to create an atlas with the silhouettes by using the "*`atlasgen`*" tool. This will enhance the performance of the map. Once this has been done, remember to remove the TGA files generated by the editor so they don't get loaded into memory together with your new atlas.*

## Optimization Tools

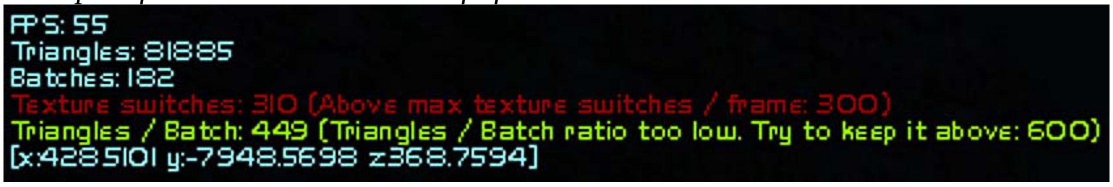
There are two tools available in the editor for checking if levels and objects are reasonably optimized.

### Render View Stats

The Render View Stats window [`Numpad /`] shows the current fps, the number of triangles currently being displayed in the camera view, the number of batches and texture switches being used to generate the current picture, what the current triangle to batch ratio is and also which position in the world that the camera is currently at.

With this tool turned on you can navigate around the map in free flight as well as spawn in [`L`] a character which you can enter [`F8`] to make sure that you are seeing the map at the correct level from the ground, and get direct feedback on the current status of the level.

*Example of* `Render View Stats` *info from* `Fort`*:*



```
FPS: 55
Triangles: 81885
Batches: 182
Texture switches: 310 (Above max texture switches / frame: 300)
Triangles / Batch: 449 (Triangles / Batch ratio too low. Try to keep it above: 600)
[x:428.5101 y:-7948.5698 z:368.7594]
```

As you can see in the picture above, the editor will color code any info that is of higher importance in the current frame. Red is bad, blue is good and green is good but inefficient.

In the example above the triangle counts is ok as well as the number of batches being sent to the processor. But the texture switch value is too high, which is caused by to many different materials being used on the bodies currently being rendered in the view. How to fix that is in the model itself is not important in this tutorial, but you should try to keep these values away from red if possible. The triangle per batch ratio is low because there are no high poly objects in the view at that time.

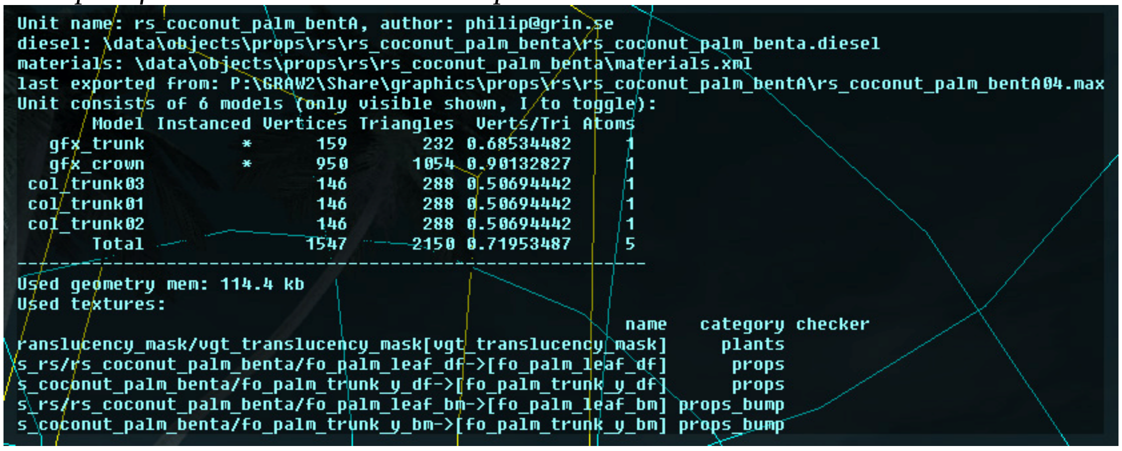
## Unit Info Mode

This mode uses its own free flight camera and with it you can fly around and look at stats and info on specific units. With this you can that way find which units are more expensive to use as it will show how many objects it contains, as well as how many polygons each of them contains, if they are instanced (which is good if you're using many of that object in the level), how many textures it uses and which textures it uses.

**Warning:** This tool is a bit unstable and can cause editor crashes quite often.

It also shows the locations inside the hierarchy for those textures, the `materials.xml` file which defines the shaders used, the diesel file for the unit, as well as which max file it was exported from and the location of it. Besides that useful info it also shows who exported the unit last and what the unit name is.

*Example of `Unit Info Mode` used on palm tree on `Fort`:*



```
Unit name: rs_coconut_palm_bentA, author: philip@grin.se
diesel: \data\objects\props\rs\rs_coconut_palm_benta\rs_coconut_palm_benta.diesel
materials: \data\objects\props\rs\rs_coconut_palm_benta\materials.xml
last exported from: P:\GRAW2\Share\graphics\props\rs\rs_coconut_palm_bentA\rs_coconut_palm_bentA04.max
Unit consists of 6 models (only visible shown, I to toggle):
      Model Instanced Vertices Triangles  Verts/Tri Atoms
   gfx_trunk         *      159       232 0.68534482     1
   gfx_crown         *      950      1054 0.90132827     1
 col_trunk03                146       288 0.50694442     1
 col_trunk01                146       288 0.50694442     1
 col_trunk02                146       288 0.50694442     1
      Total                1547      2150 0.71953487     5
-----------------------------------------------------------
Used geometry mem: 114.4 kb
Used textures:
                                                     name   category checker
ranslucency_mask/vgt_translucency_mask[vgt_translucency_mask]     plants
s_rs/rs_coconut_palm_benta/fo_palm_leaf_df->[fo_palm_leaf_df]      props
s_coconut_palm_benta/fo_palm_trunk_y_df->[fo_palm_trunk_y_df]      props
s_rs/rs_coconut_palm_benta/fo_palm_leaf_bm->[fo_palm_leaf_bm] props_bump
s_coconut_palm_benta/fo_palm_trunk_y_bm->[fo_palm_trunk_y_bm] props_bump
```

This mode will be very useful to check out custom units once they have been implemented into the game to see if they should be optimized further inside 3DS Max, or maybe be retextured with less textures to enhance game performance.

As seen in the example above, that units has two graphic objects and three collision objects (which are used to reduce the processor usage while calculating physics), as well as five textures where two are diffuse textures, two are normal maps and one is a translucency mask for the leafs. The polygon count for the tree is 2150 triangles, which is ok for such a large unit. If the vertices per triangle value for a mesh is over 1.00, that mesh should be looked over inside 3DS Max to try and optimize it more.

After these additional editor functions have been covered, it's now time to take a look at the final fixes on the outside of the editor so the level can be used in the game.

# *Chapter 13: Final Fixes*

Once the editor work is done there are some files that need to be added for the level to work in GRAW2. As this needs to be done in a certain way, I'll start using examples here which should be easy to follow. Like in chapter 1 I'll be using a level in a folder called "`tutorial`" for all examples.

## Changing Environment

If you want another environment then the default one, which will get pretty boring after a while, the following is what you need to do. To begin with you need an environment file. The simplest way is to copy one from a map in the original game that has the kind of setting you want.

I'll use the environment file from "`mission01`", so I copy the "`environments.xml`" from that folder into my "`tutorial`" folder.

If you open the "`environments.xml`" file you'll find a tag about 15-20 lines down called "`<env name="default">`". Inside that one you can see which of the different post effects and skies in this XML that are used when the mission starts as default.

*`Mission01` default environments:*

```
<env name="default">
  <adaption_light_effect value="m01_morning"/>
  <posteffect value="m01_morning"/>
  <sky value="m01_morning"/>
</env>
```

Any of the "`post_effect`" found in this XML can be used with the "`adaption_light_effect`" and "`posteffect`" tag, as well as any "`sky`" can be used with "`sky`" tag.

Inside the mission script you can also change environments as the mission progress. When doing this it's the name inside the "`env`" tag which should be used with the appropriate script event, which you can find in the tutorial "*`GRAW2: Scripting for beginners`*". In the environment file used in this example there are three additional environments that can be used from the mission script; "`formiddag`", "`middlemorning`" and "`night`".

You can also create your own environment combinations by creating a new "`env`" tag with the combinations you want, as well as create new "`post_effect`" and "`sky`" elements to use. I'll leave that for you modders to play around with.

Lastly you need to tell the level to look for the new "`environments.xml`" file but adding "`<environments path="environments.xml"/>`" into the levels "`world_info.xml`", which we'll look at in more details next in this tutorial. But after adding that line you can start up the game and should now have another environment.

## World Info

The world info file is the centre point for any level as it defines many elements of the level to the game. It contains info on where the level can find all needed files, how it can be played and how it should appear in the lists inside the game. I'll try to explain as much as possible about this file with the help of two examples.

### Single Player and Campaign Coop Example

Let's take a look at one of the original `world_info.xml` files from a campaign level to begin with.

*Example `world_info.xml` for the "`mission01`" level:*

```
<?xml version="1.0" encoding="UTF-8"?>
<world_info path="/data/levels/common/campaign_settings.xml"
    name="mission01" mission_time="day">
  <world path="xml/world.xml"/>
  <mission_script path="mission.xml"/>
  <info_strings name_id="campaign_mission_1"/>
  <environments path="environments.xml"/>
  <massunits path="massunit.bin"/>
  <sound>
    <soundbank name="act01_memo_music_sound" type="memorable"/>
    <soundbank name="act02_memo_music_sound" type="memorable"/>
    <soundbank name="act03_memo_music_sound" type="memorable"/>
    <soundbank name="act04_memo_music_sound" type="memorable"/>
    <soundbank name="ambience_shanty_morning_sound" type="ambient"/>
    <soundbank name="mission01_sound"/>
    <soundbank name="music_act01_sound" type="mood"/>
  </sound>
  <texture_scope path="texture_scope.xml"/>
  <extra_coverpoints path="coverpoints.xml"/>
  <campaign name="graw2" act="1" order="1" coop="true">
    <candidate name="BEASLEY" kit="" def_team="true"/>
    <candidate name="BROWN" kit="" def_team="true"/>
    <candidate name="HUME" kit="" def_team="false"/>
    <candidate name="JENKINS" kit="" def_team="false"/>
    <candidate name="MITCHELL" kit="" def_team="true"/>
    <candidate name="RAMIREZ" kit="" def_team="true"/>
    <block_weapon name="barrett"/>
    <block_weapon name="hk21e"/>
    <block_weapon name="m32"/>
    <block_weapon name="predator" coop="true"/>
  </campaign>
  <texture name="loading"
    texture="data/textures/atlas_gui/mission_gfx/load_sp_m01"
    uv_rect="0,0,2048,1080" width="2048" height="2048"/>
  <texture name="minimap" texture="/data/levels/mission01/minimap"
    uv_rect="0,0,1024,951" width="1024" height="1024"/>
  <graph name="coarse" path="ai_coarse.gph"/>
  <graph name="main" path="ai.gph"/>
  <border name="minimap" min_x="-12821.985" min_y="-11086.894"
    max_x="17869.268" max_y="17421.324"/>
</world_info>
```

In the main tag called "world_info", we find attributes that defines the mission name. The name has to be UNIQUE of the mission will not show up in the lists. Here you'll also set if the mission is played during "day" or "night", which affects if lights should be turned on or not on vehicles and such.

The above example is a world_info.xml file used for a map that only has one game mode, as single player and campaign coop are setup as a combined game mode. Because of this the path to the *_settings.xml for that mode is also defined in the main tag. We'll take a look at how to implement multiple game modes into a single world_info.xml later. Also refer to the tutorial "*GRAW": MP Game Modes*" for more info on this.

Next you see a tag called "world" that defines the path to the world.xml, which is the file that contains most of the work you've done inside the editor. This tag is set in the default world_info.xml that is created once you first save your level in your folder.

*Default world tag:*

```
<world path="world.xml"/>
```

After that comes the "mission_script" tag that defines the path to the mission.xml that contains your mission script. Even levels only using the unified game modes uses the mission.xml, and its contents in those cases is covered in the "*GRAW": MP Game Modes*" tutorial.

The "info_strings" tag defines the name of the mission to be shown in the Single Player mission selection screen. It needs to be given the name of a string defined inside a strings.xml file. To include a strings.xml file for your custom mission, you'll have to add another attribute here called "path" and set it to "strings.xml". With the help of this you can now create your own mission specific strings.xml file inside your custom mission folder to handle all your info tags, briefing texts, waypoint tags, objective info and so on. We'll cover the simple syntax of the strings.xml file later in this chapter.

*Example of modified "info_strings" tag for custom missions:*

```
<info_strings name_id="campaign_mission_1" path="strings.xml"/>
```

The "environments" tag was covered briefly in the last section, and it defines which environment file to use for the map. As default this tag is given the path to the default environment file when a level is first created.

The "massunits" tag defines the path to the massunit.bin file, which is the file containing all the units placed in the props brush layer. Without this tag those units will not be used when the mission is played. This should also be set by default.

Next in line is the "sound" tag which contains info on which sound banks should be used with the mission. I'm no expert on this tag, but you can see how "memorable" music, the default "ambient" sounds and the "mood" music are defined in the example. This only defines which sound banks should be preloaded. Using sound not part of a defined sound bank inside the mission should work but it needs to load at the moment it is first played.

The “texture_scope” tag defines where the file containing info on which texture atlases should be loaded when the mission is played can be found. This is a very important file. If the content of the texture scope is wrong you’ll get blue and yellow checkers showing on objects as the textures for them are not included in the atlases you’ve defined the game to load. The texture scope also has to include the folder used for the lightmap. We’ll cover editing the texture scope later in this chapter.

The tag called “extra_coverpoints” is only used on some levels that are setup to use AI. It’s not needed on city maps as all larger static units have built in cover points that will be used close to corners and other good positions. So unless your level is rural or very open, you should comment out or remove this tag from the world_info.xml. The cover points defined inside that file are calculated automatically by the editor when generating the AI graphs.

The “campaign” tag is the main tag that decides where the mission will be listed in single player; if it can be played in campaign coop as well, which team members are in the default team and which weapons that are not shown in the inventory lists (in other words are blocked from use on the mission).

The attribute “name” defines which campaign it should be listed in. That’s right; there is support for multiple campaigns in GRAW2. If it’s set to “graw2” the mission will show up in the default mission list. If it’s set to something else, an additional control will show up in the upper right corner of the mission selection screen, which lets the player switch between campaigns. “Act” decides under which act it should be listed and “order” is which mission inside that act it should be. “Act” set to “0” will list the mission above the acts like “Training Grounds”, which is where missions that don’t need to be unlocked must be placed. Lastly there is the attribute “coop”, which decides if the mission can be seen in the campaign coop create server menu.

***Note:*** *There is a possible conflict situation here as two missions can’t have the same act and order settings. If they do only the mission with the highest priority will be seen.*

By setting the “def_team” attribute to true on any team member, they will default into the mission team when entering the team selection in single player. Which kit they should use can be set in the “kit” attribute, where “_m02” would for example give the kit designed for mission02. The kits themselves are defined inside the data/lib/managers/xml/ghost_templetes.xml file.

The “block_weapon” tag removes the weapon from the inventory lists in single player, but if it’s given the “coop” attribute set to “true” it will apply to campaign coop as well. To find the internal name used for each weapon in GRAW2, take a look in appendix 2 in “*GRAW”: MP Game Modes*” where they are all listed.

The "texture" tags works just like explained in the tutorial "*GRAW": MP Game Modes*". When giving the "name" attribute the value "loading", you define which texture to use for the loading screen, and when giving it the value "minimap" you define which texture to use for the minimap. Minimap is not required for single player or campaign coop as it's not used. The "uv_rect" attribute needs the coordinates for where the upper left corner and the lower right corner of the area to be displayed from the texture is. It should be given in pixels. Lastly you should define the "width" and "height" of the entire texture you want to use.

The "minimap" texture should, for my level that's called "tutorial" be set to the look inside the level folder itself for a file called "minimap", which is the name and the location of the minimap generated by the editor. It also uses a full 1024x1024 pixel texture, so the rest of the attributes are easy to get correct.

*Default "minimap" tag for "tutorial" level:*

```
<texture name="minimap"
  texture="/data/levels/custom_levels/tutorial/minimap"
  uv_rect="0,0,1024,1024" width="1024" height="1024"/>
```

There are two different uses for the "graph" tag. The first is to set it to "main", which should define which AI graph to use. The second is to set it to "course", which also points to an AI graph, but this one has a lower density and is used when the AI needs to navigate over longer distances and will then use fewer stop locations to move faster. Both files are generated in the levels folder when calculating the AI graph as described in chapter 8.

Lastly in the example above we find a "border" tag. This defines the borders defined for the minimap inside the editor. It requires values found inside the world.xml file in the form of coordinates to the minimap markers to create a rectangle around the playable area to use when calculating the position of the icons on it in multiplayer.

To get these values you need to open the world.xml in a text browser or XML editor. Do a search for "minimap" and you'll quickly find the minimap markers inside that file. Copy those into a new document so we can clean out the parts we need.

*Example of minimap markers found inside mission01 world.xml:*

```
<marker name="" type="minimap">
  <position pos_x="-12821.985" pos_y="9525.5225" pos_z="9659.4834"/>
  <rotation yaw="-78.921143" pitch="0.0010194057" roll="-140.63948"/>
</marker>
<marker name="" type="minimap">
  <position pos_x="8064.0854" pos_y="17421.324" pos_z="9035.6865"/>
  <rotation yaw="-40.824047" pitch="-0.000204905" roll="120.44201"/>
</marker>
<marker name="" type="minimap">
  <position pos_x="17869.268" pos_y="-2202.4155" pos_z="7623.9951"/>
  <rotation yaw="-34.744892" pitch="-0.000800839" roll="122.86446"/>
</marker>
<marker name="" type="minimap">
  <position pos_x="-10609.501" pos_y="-11086.894" pos_z="6056.4604"/>
  <rotation yaw="-33.764946" pitch="-0.00140531" roll="-66.985222"/>
</marker>
```

There can be any number of minimap markers used, but we are only interested in the extreme positions. So after you have copied the entries, delete all lines except those holding position coordinates. In the position tags you can remove everything besides the “pos_x” and “pos_y” and their given values.

*Cleaned minimap marker list:*

```
pos_x="-12821.985" pos_y="9525.5225"
pos_x="8064.0854" pos_y="17421.324"
pos_x="17869.268" pos_y="-2202.4155"
pos_x="-10609.501" pos_y="-11086.894"
```

Now we can easily get the highest and lowest “pos_x” values and enter them as “max_x” (*17869.268*) and “min_x” (*-12821.985*), as well as the highest and lowest “pos_y” value and enter them into “max_y” (*17421.324*) and “min_y” (*-11086.894*).

That is all the entries inside that world_info.xml. Now we’ll take a look at the differences inside one from a multiplayer level.

## Multiplayer Game Mode Example

As all the different tag types where covered in the last section, this section will only cover what is special about setting up the world_info.xml for a level that supports multiple game modes.

*Example world_info.xml from the “nowhere” level:*

```
<?xml version="1.0" encoding="UTF-8"?>
<world_info name="nowhere" mission_time="day">
  <world path="xml/world.xml"/>
  <mission_script path="mission.xml"/>
  <environments path="environments.xml"/>
  <massunits path="massunit.bin"/>
  <sound>
    <soundbank name="ambience_park_night" type="ambient"/>
    <soundbank name="avr_nowhere_sound"/>
    <soundbank name="music_act01_sound"/>
  </sound>
  <texture_scope path="texture_scope.xml"/>
  <texture name="minimap"
    texture="/data/textures/gui/nowhere_minimap"
    uv_rect="0,0,1024,792" width="1024" height="1024"/>
  <border name="minimap" min_x="-10676.768" min_y="-6114.8867"
    max_x="18646.32" max_y="16576.049"/>
  <world_info path="/data/levels/common/tdm_settings.xml" type="tdm">
    <texture name="loading"
      texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
      uv_rect="0,0,2048,1080" width="2048" height="2048"/>
  </world_info>
  <world_info path="/data/levels/common/dm_settings.xml" type="dm">
    <texture name="loading"
      texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
      uv_rect="0,0,2048,1080" width="2048" height="2048"/>
  </world_info>
</world_info>
```

The level in this example is only setup for two game modes, but you can of course just repeat the game mode specific parts infinite number of times to accommodate any number of game modes you want setup on your map. More info on setting up everything for each specific game mode can be found in the companion tutorial "*GRAW": MP Game Modes*".

The first difference you can see in this world_info.xml is that it's much smaller. That's because many of the items in the previous example are only needed when in single player or campaign coop, or when using AI in the game mode like done in coop. These nodes include the "info_strings", "graph" and "campaign" tags.

When you look closer at the tags you'll also notice a difference in the main "world_info" tag, which doesn't have an attribute that defines which *_settings.xml to be used. This is because each game mode defined on a level MUST have a separate *_settings.xml file. So this will be defined later in the game mode specific sections.

Besides the differences mentioned above, the file looks basically the same besides having another "world_info" tags inside the main one. These are the game mode specific areas. This map only has two game modes, TDM and DM. Each has its own "world_info" tag inside the main "world_info" tag. Let's take a look at one of them, the TDM tag.

*Masked out TDM game mode specific parts:*

```
<world_info path="/data/levels/common/tdm_settings.xml" type="tdm">
  <texture name="loading"
    texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
    uv_rect="0,0,2048,1080" width="2048" height="2048"/>
</world_info>
```

As you can see it is here defined which *_settings.xml to be used for this game mode by assigning it to the "path" attribute in this internal tag. It also specifies which tag name the game mode has by assigning it to the "type" attribute, which should be the same tag name as defined inside the *_settings.xml used.

Everything that is included inside the "world_info.xml" will only be used once the TDM game mode is being played. In this case the only difference is which loading screen to be used, but any of the other tags could be included here as long as make sure that every game mode gets there needed tags defined. If you want to use a game mode that has AI graph you only need to include the "graph" nodes inside that game modes specific area. But if you want to assign a different environment for a game mode, each game mode must have an environment defined inside them as you always have to have and environment to play a level in any mode.

*Example of different environment implementation:*

```
<world_info path="/data/levels/common/tdm_settings.xml" type="tdm">
  <environments path="environments_tdm.xml"/>
  <texture name="loading"
    texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
    uv_rect="0,0,2048,1080" width="2048" height="2048"/>
</world_info>
<world_info path="/data/levels/common/dm_settings.xml" type="dm">
  <environments path="environments_dm.xml"/>
  <texture name="loading"
    texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
    uv_rect="0,0,2048,1080" width="2048" height="2048"/>
</world_info>
```

There isn't much more to say about the `world_info.xml` actually. Just that if there are files you don't have, like `coverpoints.xml`, make sure to remove those tags inside the `world_info.xml`. Now let's move on to the texture scope.

## Texture Scope

The next we need to look at is setting up the texture scope for the level to remove all possible checkers. Remember that if it's only a few objects that obtain blue and yellow checkers you should maybe remove those instead of changing the texture scope to gain better performance on the map, but it's up to you of course.

In your new levels folder, "`tutorial`" in my case, you should have a basic `texture_scope.xml` file. Inside the file you'll find a list of all the atlases in the game, but many of them are commented out and as such are not used right now.

To remove the possible checkers you'll need to do some testing at this stage, which can't be done inside the editor as it has access to all textures in the game. You'll have to try and uncomment an extra atlas and then testing in-game if it solved the problem on any of the checker units. If it did, they keep it, if it didn't then comment it back out and try another one. Repeat these steps until you have all the textures you need visible inside the game. Once again, if there is an atlas that only helps with very few units, you should think it over if those are really needed of they should be removed instead of their atlas added to the texture scope, which would give better performance.

This is a bit different from what we did at Grin as we defined many atlases after the levels, not the other way around. But it's the only way to do it when mixing units from different levels. But you can get a hint in some unit names which atlas they belongs to. Especially buildings and larger units usually have a prefix, like "`city_`", that besides being good for sorting in the lists also refers to which atlas it uses.

In the texture scope we also have to define the path to where the levels lightmaps are located and the same for the silhouettes and the cubemap, but those parts are already defined and should work with their default values for your level.

## Strings

The syntax of the `strings.xml` file is very simple. It contains a main tag called "`stringset`" inside which all the strings are defined with the help of an element called "`string`".

You give each "`string`" element the attribute "`id`" which you set to the name of the string to be used when calling for it, so this has to be unique or you'll get problems. A good rule is that always start the id with the level name followed by "_" and then a short description of either what it is to be used for or what it contains. After that you add an attribute called "`value`" which you give the string text you want to have. It's as simple as that.

*Example of `strings.xml` syntax:*

```
<stringset>
  <string id="title_m01" value="Search and Destroy"/>
</stringset>
```

There are a few special characters that can be used inside the string values to create line breaks "`\n`" and other formats. Check inside the original strings files found under "`data/strings`" if you want to use them.

## Outro

*That covers the tools included in the editor used to develop GRAW2 and some addition aspects that needs to be worked on to make a level playable inside the game. I hope this document answers a few questions at least, even though I guess it will also raise a few. The provided tutorials in combination should provide a good starting point for modders to evolve their own game play experiences.*

*Good Luck.*

*Grin_Wolfsong, out.*

## *Appendix 1: Environmental Sound Cues*

This appendix contains a list of some of the sounds found inside GRAW2
.

### Sound Cues

**A**

| | |
|---|---|
| aa_1shot | 1 missile fired from distant aa cannon |
| aa_40shot | 40 shot burst from distant aa heavy machinegun |
| aa_bh_shake | played in BlackHawk when hit by aa fire |
| aa_explosion | aa mid-air explosion |
| ac_hum | humming of external wall mounted ac |
| ad_pillar_collapse | ad pillar destroyed (by explosion) |

**B**

| | |
|---|---|
| baby_cry | one non-looped baby cry |
| barrel_pristine_impact | impact sound of pristine barrel |
| barrel_rusty_impact | impact sound of rusty shanty town barrel |
| bird_a | one non-looped bird call from bird "a" |
| bird_b | one non-looped bird call from bird "b" |
| bird_c | one non-looped bird call from bird "c" |
| bird_d | one non-looped bird call from bird "d" |
| bird_e | one non-looped bird call from bird "e" |
| bird_f | one non-looped bird call from bird "f" |
| bird_g | one non-looped bird call from bird "g" |
| birds | randomly selects a bird (a-g) and loops it eternally |
| blast | electricity spark from broken circuit box |
| branch_small_rustle | rustle of small branch (e.g. player brushes against bush) |
| brick_impact | non-looping sound of brick falling from ruin (to be triggered with visual effect) |
| brick_kick | sound of a brick being kicked across the ground |
| buzz | looped buzz from circuit box |

**C**

| | |
|---|---|
| cafe_chair_impact | iron chair bump |
| cafe_table_impact | iron table bump |
| cardboard_box_impact | cardboards box bump (from falling, e.g.) |
| child_cry | one non-looped child cry |
| cistern_loop | looped dripping sound from metal cistern |

| | |
|---|---|
| coin_spit | non-looping sound of vending machine or similar spewing out coins for app. two seconds |
| coin_stone | non-looping sound of a coin dropping to the ground |
| container_shake | sound of steel container set vibrating by a bullet hit |
| crickets | randomly picks one of three cricket sounds and loops eternally |
| crows | randomly picks one of five crow sounds and loops eternally |

**D**

| | |
|---|---|
| debris_large_impact | generic sound of large debris to ground |
| debris_medium_impact | generic sound of medium debris to ground |
| debris_small_impact | generic sound of small debris to ground |
| drainage_loop | looping sound of a slurping water drainage |

**F**

| | |
|---|---|
| factory | randomly picks one of five muffled factory noises and loops eternally |
| fire_large_10 | sound of large fire playing for app. ten seconds |
| fire_large_120 | sound of large fire playing for app. 120 seconds |
| fire_large_15 | sound of large fire playing for app. 15 seconds |
| fire_large_20 | sound of large fire playing for app. 20 seconds |
| fire_large_30 | sound of large fire playing for app. 30 seconds |
| fire_large_5 | sound of large fire playing for app. five seconds |
| fire_large_60 | sound of large fire playing for app. 60 seconds |
| fire_large_90 | sound of large fire playing for app. 90 seconds |
| fire_large_loop | sound of large fire looping eternally |
| fire_small_10 | sound of small fire playing for app. ten seconds |
| fire_small_120 | sound of small fire playing for app. 120 seconds |
| fire_small_15 | sound of small fire playing for app. 15 seconds |
| fire_small_20 | sound of small fire playing for app. 20 seconds |
| fire_small_30 | sound of small fire playing for app. 30 seconds |
| fire_small_5 | sound of small fire playing for app. five seconds |
| fire_small_60 | sound of small fire playing for app. 60 seconds |
| fire_small_90 | sound of small fire playing for app. 90 seconds |
| fire_small_loop | sound of small fire looping eternally |
| flag_loop | sound of flag flapping and wire hitting flag pole, looping eternally |
| flies_loop | sound of flies, heard up to 20 meters, looping eternally |
| flies_loop_small | sound of flies, heard up to 10 meters, looping |

| | |
|---|---|
| | eternally |
| foliage | 2d sound of wind in a grove of trees, heard up to 30 meters, looping eternally |
| foliage_rustle_occasional | non-looping 3d sound of wind in foliage, heard up to 30 meters |
| fountain_large_loop | water flowing in large fountain, looping eternally |
| frogs_loop | sound of frogs, looping eternally |

**G**

| | |
|---|---|
| glass_break_big | window blown out by explosion |
| glass_break_small | smaller window blown out by explosion |
| gravel | eternally looping sound of occasional gravel falling in quarry |
| grenade_frag_explode | frag grenade explosion |

**H**

| | |
|---|---|
| hangar_door_open | hangar door opening, 6 seconds from start to stop |

**I**

| | |
|---|---|
| intruder_alarm | alarm woop (one single non-looping sound) |
| intruder_alarm_break | sound of electronics breaking and distorted speaker feedback |
| intruder_alarm_broken | broken alarm woop (one single non-looping sound) |
| intruder_alarm_button | push of alarm button |

**L**

| | |
|---|---|
| large_rock_impact | sound of a large concrete slab or rock hitting the ground hard |
| lightbulb_break | lightbulb breaking (from damage (explosion, bullet) or falling to ground) |

**M**

| | |
|---|---|
| magslide_out | weapon reload |
| man_scream | one non-looped man screaming in pain |
| man_shout | one non-looped man pleading in desperation |
| metal_creak | metal creak sound (e.g. marquee or parasol folding) |
| mortar_1shot | firing of mortar shell far, far away (low, rumbling sound) |
| mortar_drop | three second "screaming" sound of mortar falling (to precede the explosion sound of the mortar shell) |

| | |
|---|---|
| mortar_explode | explosion sound of a mortar shell |

**O**

| | |
|---|---|
| ocean_loop | randomly selects one of two sounds of open water and loops it eternally |

**P**

| | |
|---|---|
| paper_cup_impact | paper cup kicked across ground |
| pigeons_loop | eternally looping sound of pigeons, to be played by pigeons on ground and stopped when they take off |
| pigeons_takeoff | non-looping sound to be played from the "effect" of pigeons taking off |
| plank_impact | plank bump (piece of collapsed wooden crate) |
| pot_large_shatter | large clay pot shattering (shot to pieces) |
| pot_shard_impact | piece of destroyed clay pot hitting the ground |
| pot_small_shatter | small clay pot shattering (shot to pieces) |

**R**

| | |
|---|---|
| rattlesnake | eternally looping sound of occasional rattlesnake noise |
| roadsign_impact | metal road sign falling to ground |
| rooftop | eternally looping sound of occasional noise from a rooftop (bottles falling over, gusts of wind, hinges creaking |
| rooftop_randomizer | randomly plays one non-looping sound from the "rooftop" sound (above) |
| rpg7_explode | enemy RPG missile exploding (hitting target) |
| ruins | non-looping sound of dust falling from ruins (to be triggered with visual dust effect) |

**S**

| | |
|---|---|
| scrambler_explode | scrambler being shot to pieces |
| scrambler_loop | eternally looping sound of scrambler affecting the HUD (effective up to 100 meters) |
| seagulls | eternally looping sound of occasional seagull bird calls (can be used to "diffuse" the above sound, or all by itself) |
| seagulls_loop | eternally looping sound of constant seagull noise (as no seagulls are visible, preferably place this sound behind rocks) |
| shatter_glass | sound of glass shards falling to the ground (beneath broken window) |

| | |
|---|---|
| shatter_glass_big | larger glass object shot to pieces (shards sent flying) |
| shatter_glass_small | small glass object shot to pieces (no shards sent flying) |
| shovel_impact | shovel falling to ground |
| soda_can_impact | soda can kicked across ground |
| squeaky_door_close | non-looping sound of squeaky "shanty town style" door closing |
| squeaky_door_open | non-looping sound of squeaky "shanty town style" door opening |
| stones | eternally looping sound of occasional stones falling in quarry |

**T**

| | |
|---|---|
| togo_box_impact | takeaway food carton kicked across ground |
| tunnel | eternally looping 2d sound of tunnel ambience audible up to 60 meters - useful also to create indoors ambience in e.g. a hangar |

**W**

| | |
|---|---|
| wave_splash | non-looping sound of wave spashing against rocks (to be triggered by visual water effect) |
| whistle | one non-looped mouth whistle |
| woman_cry | one non-looped woman cry |
| wooden_crate_collapse | wooden crate blown apart |
| wooden_crate_impact | wooden crate bump (from falling, e.g.) |