User's Guide
CaWE

Ca3DE World Editor

Carsten Fuchs

June 24, 2005

Contents

1 Introduction

If you are interested in making new worlds for the Ca3D-Engine, then this document is for you. It is written as a guide and introduction to *CaWE*, the Ca3DE World Editor.

On problems and questions, please do not hesitate to post at the Ca3DE forums at `http://www.Ca3D-Engine.de`.

2 Installation

The CaWE editor is included with the Ca3DE Developers Kit, and as such requires no explicit installation: After unzipping the packed file, simply change into the `CaWE` sub-directory, and double-click on `CaWE.exe`[1].

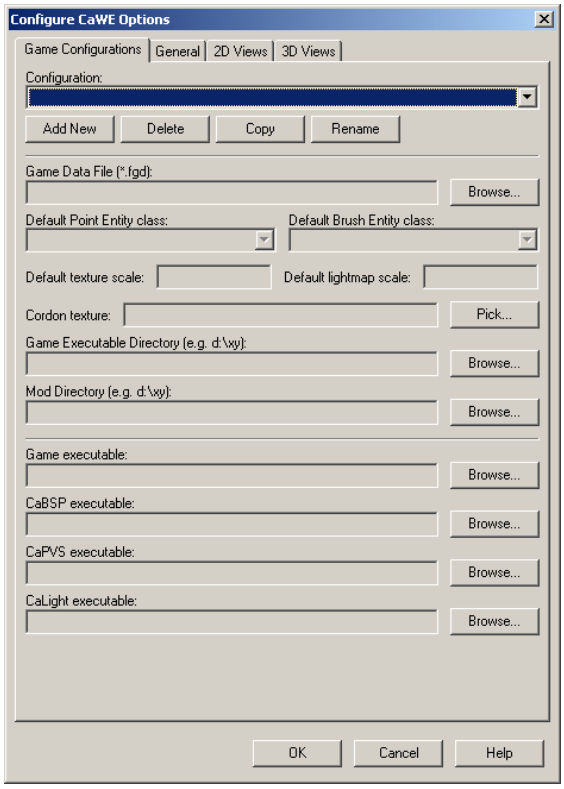
2.1 Running CaWE for the first time

CaWE requires some basic configuration for normal use, and thus asks you for providing at least one game (or "MOD") configuration when it is first run:

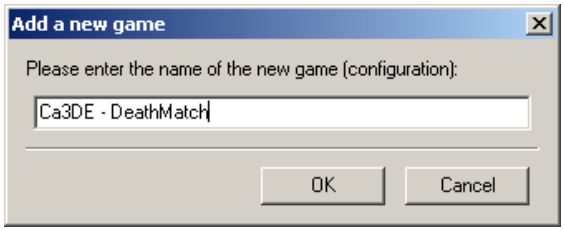
1. 

After clicking on **OK**, CaWE will automatically show you the related configuration dialog.

[1] The file is actually named `CaWE_win32_vc60_r.exe` for the Microsoft Windows release, and `CaWE_li686_g++_r.linux` for the Linux variant – don't let that fact confuse you.

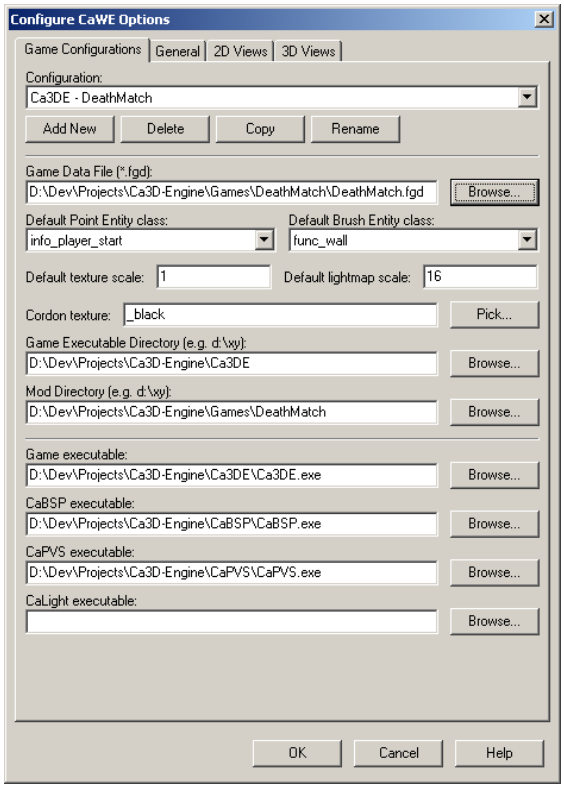
2. 

Here, click on **Add New** for adding a new game configuration. You will then be asked for the desired name of the new config.

3. 

You can enter anything here, i.e. normally something that reminds you what MOD this config is related to, like "Ca3DE DeathMatch".

4. Next you have to fill-in where the Game Data File is located. This is a file with `.fgd` suffix. One such file is already provided with the rest of the Ca3DE Developer Kit: the `DeathMatch.fgd` file in the `Ca3D-Engine/Games/DeathMatch` subdirectory. Click on the related **Browse...** button and locate the file.

5. 

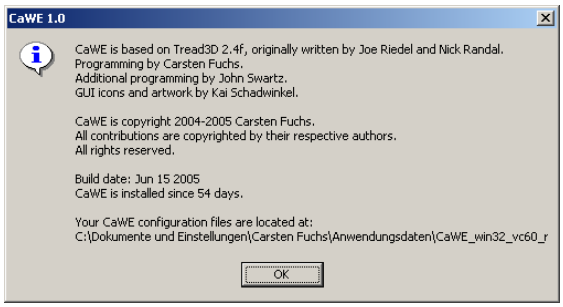
Finally, fill-in the rest of the dialog appropriately such that it eventually looks like this. The paths will of course be different for you, depending on where you installed the development kit and CaWE. Also, you do not have to provide all the settings for now, you can leave all fields with "executable" in their name empty (that is, the bottom four and the Game Executable Directory).

All settings in this dialog can be revised later at any time via the **File → Configure CaWE...** menu. Adding new or additional game configurations at that time may however require to restart CaWE for the settings to take effect.

2.2 De-Installation

CaWE, as all other Ca3DE related software, was designed to interfere with your computer and operating system as little as possible. In general, it is enough to delete the directory that the packed file was unzipped to during installation, and all traces of CaWE are gone.

There is one exception though: When run, CaWE stores your configuration settings in a newly created directory. This directory is located in the "home" directory of the current user. The exact place on your harddisk depends on your OS and even on its version, but CaWE states the full path to it both in the **Configuration** dialog (**General** tab), and in the **About** dialog box that is accessible via the **Help → About** menu.



3 Getting Started

3.1 Introduction to Editing

Brushes

Do you know how a Lego model or a real house is built from bricks? Invidividual, small bricks are combined to form a complex architectural structure. CaWE works principally in the same way, and thus if you can visualize how individual bricks form a model or a big building, you will find it easy to grasp the concepts behind CaWE.

In CaWE, the bricks are called *brushes*, and they come in several shapes: blocks, wedges, cylinders, pyramids etc. Moreover, you can modify the default brushes in many ways. They can be scaled, sheared, mirrored, cut in pieces, carved and so on. You can create an arbitrary big number of these brushes in any shape you like, and use and combine them for building the matter that forms your world: walls, floors, roofs, furniture, rocks, ... Brushes eventually get *materials* assigned that represent the surface properties of that brush, as for example rock, glass, concrete, sand or water. The creation of brushes is detailed in section 4.3, The New Brush Tool.

Bezier Patches and Terrains

Modelling really complex surfaces can be difficult with brushes alone, especially if the surfaces should be curved, organic, or very big. Ca3DE and CaWE therefore provide two additional basic elements that complement brushes: *Bezier patches* are curved surfaces that you can imagine like bent or stamped metal plates. They can be used to model pipes, curves, smooth archs, and many other organic objects. The creation of bezier patches is detailed in section 4.5, The New Bezier Patch Tool. *Terrains* are similar, but as their name suggests, aim at surfaces that are much bigger and more irregular. They're also treated specially by the engine and have very high performance. The creation of entities is detailed in section 4.6, The New Terrain Tool.

Entities

Entities define and represent all other interesting things in a world, and they come in two flavours:

Point-based entities exist at a specific coordinate in space and be can be thought of as a reference point. Although these entities may refer to objects that actually have a volume, for example a monster, they are only treated as points in CaWE, and the Ca3D-Engine provides the volume later in the game. An example is a monster entity: In CaWE, it just represents the point where the monster will be, in Ca3DE, it will acutally be there.

Brush-based entities are related to at least one brush or bezier patch and thus have a volume. Examples for brush based entities include doors and platforms, bodies of water, area triggers, etc.

The creation of entities is detailed in section 4.4, The New Entity Tool.

Static Detail Models

While *static detail models* are really only a specific type of entity, they provide another powerful means to augment the detail in your world in a way that is not easy or even impossible to achieve with brushes, bezier patches and terrains alone. Such models can be imported from your favourite 3D modelling program, they can be arbitrarily complex, and they can for example be under the control of a level-of-detail system. Examples include models for furniture, statues, household inventory, vehicles, etc. A description of all available entity types is available in section 5.1, the Entity Guide.

Putting it All Together

By combining these simple components, you can create stunningly virtual worlds whose number of variants is virtually unlimited. Starting with a rough outline of brushes and possibly terrains, you can fill in architectural detail with other brushes, patches and entities. Eventually you will populate your world with more entities that bring your creating to life.

The final step in making a world is compiling it so that it can be run with the Ca3D-Engine. This is a quite complex process that precomputes visibility, lighting and many other details. Future versions of CaWE will come with a great tool to make compiling much easier, but until then a little typing at the command-line must be done to run the compilers. It's still easy to manage, and discussed in detail in section 6.1.

3.2 Tutorial: My first Room

Coming soon...

6.1 Compiling Maps

When you have finished editing a map, you'll want to run it with Ca3DE. The engine however requires that the map is in a precomputed form, and this section describes how you turn your map into a fully precomputed world file.

6.1.1 Compiling step 1: CaBSP

This program takes a map file in `cmap` file format and creates a Ca3DE world file from it. Example:

```
C:\Ca3DE-MDK\Ca3D-Engine> CaBSP Games/DeathMatch/Maps/MyMap.cmap
Games/DeathMatch/Worlds/MyMap.cw
```

WARNING: The specified destination file (`Games/DeathMatch/Worlds/MyMap.cw` in the above example) gets overwritten without prior notice! This is also true for mis-spelt file names. For example, if you type `Games/DeathMatch/Maps/MyMap.map` instead of `Games/DeathMatch/Worlds/MyMap.cw` – catastrophe! As with most other computer software, in order to prevent a disaster, please *backup your files* before *you begin*!

For quick tests during world development, you might want to skip the next two compiling steps (CaPVS and CaLight). You can run the file obtained from the CaBSP tool directly in the Ca3D-Engine. It will be unlit and without PVS information (which results in a lower frame rate), but is usually sufficient for early tests.

6.1.2 Compiling step 2: CaPVS

This program creates the *Potentially Visibility Set* for your worlds. Example:

```
C:\Ca3DE-MDK\Ca3D-Engine> CaPVS Games/DeathMatch/Worlds/MyMap.cw
```

The CaPVS compile switches

Even though the latest versions of CaPVS have become fast enough to deal with even the most complex worlds within reasonable time bounds, sometimes it may still be preferable to obtain faster results (at the cost of accuracy).

Before you read on, though, my advice is to simply skip the rest of this section, and continue with the next. Why? Well, the compile switches introduced below were made at a time when CaPVS was inherently slower. Since then, it has undergone many algorithmic improvements, and the switches were only left for safety and as a last resort. I hardly ever use them myself (although their implementation is interesting), and nowadays never use them for release compiles (the worst world I have takes 27 hours on a 866 MHz Pentium III this way). During world development, you should skip CaPVS entirely rather than trying to tune it for faster compilation. Also keep in mind that the proper use of `func_detail_object` entities can have a big impact on performance in general, and on CaPVS compile time in particular.

Anyway, the latest version of this tool has two new command line switches that were made to lower its compile times by effectively trading quality for speed. Both switches control the creation of *SuperLeaves*. To understand what SuperLeaves are, you must first have a rough idea what *leaves* are: Leaves are spatial convex cells into which CaBSP subdivides your world and in which the action of the game takes place later on. Simplified spoken, you can imagine that each room of your world corresponds to a leaf. CaPVS then determines the PVS by calculating if any unobstructed lines of sight exist between any pair of two leaves. It is not only the sheer number of leaves that makes the computation so slow, but also their spatial relations among each other. In some cases, there are unnecessarily too many leaves for the purposes of CaPVS, and it is preferable to merge some of them. Thus, a *SuperLeaf* is a set or a group of simple leaves. One could say that SuperLeaves subdivide the world in cells as leaves do, but the subdivision is "coarser" than with the simple leaves. When you run CaPVS on a world, it tells you how many SuperLeaves it constructed from the simple leaves. By default (without command line switches), the SuperLeaves are identical to the simple leaves, as is their number.

Now about the actual switches: Running CaPVS without any parameters (not even a world name) will print out a self-explaining usage information. I'll not repeat it here, but rather give you some tips on the usage of the two mentioned switches. Note that you

always trade quality for speed, but the compromise is usually a very good one (quality remains high even for quite fast compiles).

In general, try to reduce the compile time with care. If the situation was actually hopeless before, it does not make sense to use the switches to bring the compile time down to a few minutes or seconds. (That would be more like not running CaPVS at all.) Rather, a good idea would be to send me an email with a brief description of the problem, along with the world you are trying to compile (I'll only believe that you created a world that's too hard for CaPVS after I've seen it myself ☺).

Then, consider two strategies: You can either choose to work "top-down" by choosing the parameters such that the number of SuperLeaves is gradually reduced in small steps. The quality remains high, but the downside is that you possibly have to wait another long time until you know if the reduction was enough.

Thus, I'd recommend a "bottom-up" approach: Initially reduce the number of SuperLeaves a lot, start with a quite small number. If CaPVS then takes only a few minutes, but you want it rather to spend some hours, adjust the parameter values carefully so that the number of SuperLeaves increases slowly and gradually. You'll probably have to re-try and experiment a few times until you get a feel of it.

Finally, you should prefer the `-minAreaSL` switch over the `-maxRecDepthSL` switch, because the former is more sensitive to the actual world geometry. Do not worry about high numbers on `-minAreaSL`. When I last used it, I usually had values ranging from 1,000,000 to 5,000,000.

6.1.3 Compiling step 3: CaLight

This program calculates the lighting for a map. The `CaLight.cfg` file contains the radiant exitance definitions for textures that are supposed to be light emitters. This program can take very long too, but you have better control over it by specifying command line switches. Examples:

```
C:\Ca3DE-MDK\Ca3D-Engine> CaLight
C:\Ca3DE-MDK\Ca3D-Engine> CaLight Games/DeathMatch/Worlds/MyMap.cw
C:\Ca3DE-MDK\Ca3D-Engine> CaLight Games/DeathMatch/Worlds/MyMap.cw
-BlockSize 8 -StopUE 2
C:\Ca3DE-MDK\Ca3D-Engine> CaLight Games/DeathMatch/Worlds/MyMap.cw
-fast
C:\Ca3DE-MDK\Ca3D-Engine> CaLight Games/DeathMatch/Worlds/MyMap.cw
-SkipDialog
```

The first line prints out detailed usage information for the CaLight tool, the second line runs default lighting, which is recommended for most cases. The third and fourth lines are for very fast lighting, which is useful for quick tests during map development, and the fifth runs default lighting without invoking the dialog, which is good for batch processing. If you don't use the `-SkipDialog` switch, CaLight will present you a dialog that allows you to override the sunlight settings, that (should) have been set earlier

in CaWE. Please note that it is almost never reasonable to set BlockSize below 3 and StopUE below 0.1 for highest quality lighting, even though that is possible (some of the worlds of the Ca3DE demo releases were compiled with a StopUE of 0.05).

During the second phase of the computations ("bounce lighting"), note that you can interrupt the program by pressing the `space` button. That's sometimes useful for quick tests during map development.

Finally, a word about memory consumption: `CaLight` requires 64 MB of RAM for processing even the most simple worlds. Thus, make sure that you have at least 128 MB, better 256 MB (or more) physical RAM available! Note that it is normal that your computer swaps a lot of virtual memory right after program start. If however the swapping (extensive disk activity) does not even stop during the direct or bounce lighting phases of the program, better abort it. Under such circumstances, it will be proceeding *very* slow anyway, and the lengthy activity is probably not healthy for your hard-disk. Only plugging-in more RAM will help then, but be assured that until now, I have never seen a world that required more than about 192 MB.

6.1.4 Running the world with Ca3DE

If not already there, copy your world into the `Games/DeathMatch/Worlds` directory if you made the world for the DeathMatch MOD, or into the `Games/OtherMOD/Worlds` directory of the MOD you made the world for. Then simply run `Ca3DE.exe` as stated in the user manual of the current demo release. Your world will be listed among the other worlds in the dialogs world list ("server options").

6.2 Dealing with Leaks

CaBSP has become very good in reliably detecting leaks, and when CaBSP aborts compiling and complains about leaks, it makes pretty clear in its error message what's going on.

However, if this is your first leak, I'd much recommend to read the great explanation in the Hammer online help in the Troubleshooting section: "How do I fix leaks in my level?". It also teaches you what leaks are and how you can prevent them. Here are a few additional remarks that might be helpful:

- Run "Check for Problems" from the CaWE "Map" menu.
- Do never assume that translucent or detail entities seal your world. Sky brushes do.
- Carefully look for misaligned brushes. In my experience, the "Carve" tool, rotation and certain other operations easily cause rounding errors which in turn cause leaks.
- The best you can do is to *prevent* leaks from the very beginning by using the grid and avoiding all tools that potentially introduce rounding errors. As a gross guideline, all operations that operate on the grid and that are guaranteed to produce *vertices* on the grid are usually safe, such as resizing rectangular blocks, shearing,

vertex manipulation (although that can introduce other errors, which can be identified with the "Check for Problems" item from the "Map" menu), mirroring, and clipping and carving, when the resulting vertices are on-grid. The following operations are usually less safe: Arbitrary rotation, and clipping and carving, when the result has vertices that are off-grid.

For less-safe operations it is usually better to try to mimic the same effect with the safer operations. For example, when you want an arched door in a wall, creating and placing the wall brushes manually is much preferred over carving an irregular hole into the wall!

Finally, let me mention a *very* rare situation in which CaBSP reports no leak, but a leak is present: The symptoms of such unreported leaks are a blank screen when you load the world in Ca3DE, only environmental background is displayed, or you are falling through the floor and everything disappears after a few seconds. Such leaks occur sometimes when some old BSP decompile tools that are still available on the internet, produce world geometry that is so degenerate that not even CaBSPs rounding error control and validity tests can help it. It is possible to remedy such cases automatically, but I will write a description for the process only should somebody actually ever experience this problem.

6.3 Light Sources and Skies

- About lights: Nearly every texture in `Games/DeathMatch/Textures` that looks like a light source, is a light source. That is, such textures are defined to actually emit light. Thus, there is hardly ever a need to use "Isotropic Point Light" entities at all. Please have a look into the `CaLight.cfg` file for a list of textures that emit light, but for future compatibility you should not make changes in this file (except for your own new textures).

- You can add sky domes to your world by applying the "sky" texture to the "ceiling" surfaces. You also have to set the sky name, sunlight direction, and sunlight irradiance (intensity and color) in the "Map properties" dialog. Table 1 on page 13 lists a few examples in order to give you a rough idea about what to enter. (Don't worry if you have initially forgotten to enter these settings in the "Map properties" dialog. The CaLight tool that you will use later gives you a second chance to override them (see section 6.1.3).)

6.4 Adding Terrains to your worlds (The Old Way)

PLEASE NOTE: The contents of this section is correct, but difficult to comprehend and thus not for the faint of heart. I'm currently working on future versions of CaWE that will make terrain handling much easier!

Ca3DE features a very powerful terrain renderer that is capable to display *very* large out-door scenes. This section will teach you how terrains are created and combined with normal worlds for use with Ca3DE.

AEonsCube (cold moonlight):	
skyname	Snow2
sun_direction_x	2
sun_direction_y	-5
sun_direction_z	-9
sun_irradiance_r	2
sun_irradiance_g	4
sun_irradiance_b	6

CastleMF (warm light of a sunrise):	
skyname	Nottingham
sun_direction_x	3
sun_direction_y	-6
sun_direction_z	-3
sun_irradiance_r	100
sun_irradiance_g	63
sun_irradiance_b	30

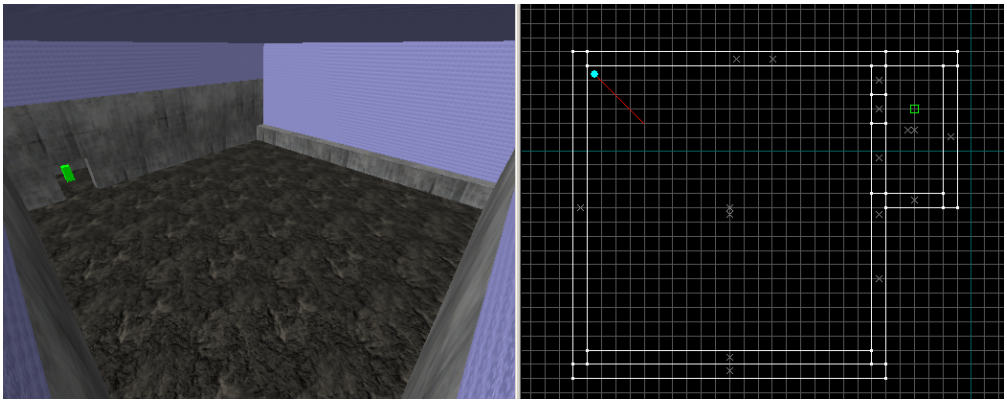
JrBase1 (warm daylight):	
skyname	T_red5
sun_direction_x	-6
sun_direction_y	5
sun_direction_z	-12
sun_irradiance_r	100
sun_irradiance_g	90
sun_irradiance_b	80

Table 1: Example values for sky dome and sunlight settings (to be set in the CaWE "Map properties" dialog).

6.4.1 Tutorial 1: A simple example

This tutorial will introduce you to the basic concepts of using terrains with Ca3DE by walking you through a simple example. Please make sure to read it entirely, as it covers aspects for the more advanced sections later.

1. Start making your new test world by creating two simple rooms. Connect both rooms by a small alley. The first room should contain the `info_player_start` entity, the second room should essentially be large and square:



2. Now we will create a `HeightMap` entity in the second room: `HeightMap` entities define the spatial location and dimensions of terrains.

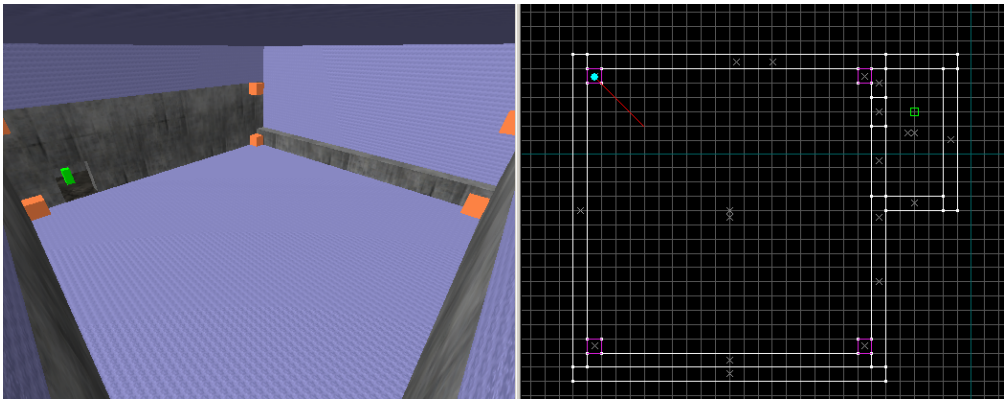
   In order to do so, we could create a single cuboid-shaped brush that entirely fills the inside of the second room (almost like flooding it with water), turn it into a `HeightMap` entity, and be done.

   While that strategy will work, it has the disadvantage that such a big brush often gets into the way when other parts of the world are to be edited. Therefore, my preferred method is to create eight smaller brushes where the corners of the single-block brush would be. Then I select all eight corner brushes simultaneously, and turn them into a `HeightMap` entity. While it is more convenient for editing, the Ca3D-Engine handles this situation exactly like the "single-block" approach otherwise.

   In fact, a `HeightMap` entity can consist of an arbitrary number and arrangement of brushes. Ca3DE (actually, CaBSP) will construct the axis-aligned bounding-box ("AABB") of these brushes later, and this AABB defines the spatial dimensions of the terrain.

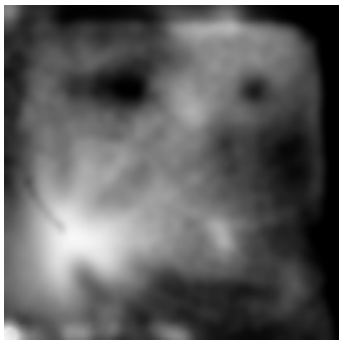
   If you have not already done so, please create the `HeightMap` entity now: Select all eight corner brushes, click the "toEntity" button, select `HeightMap` as the entity class, and close the entity property dialog (we will set the individual properties later). The next figure illustrates the result of this step.

   Tip: When you use the method with the eight cubes, please make sure that they are properly aligned to each other. Make the sides of the corner brushes precisely touch the walls, such that the terrain will cover the floor completely. It's also a good idea to make the floor of the second room invisible, such that it cannot interfere with the terrain later. The easiest way to achieve this is to assign the `sky` texture to the floor face, which also implies that this (invisible) floor face does not get a lightmap associated:



3. As far as world editing is concerned, you're done for now. Save your world in `map` file format, leave CaWE, but do not yet compile the world.

4. Next, we need to create the actual heightmap image that defines the shape of the terrain (within the AABB defined earlier). Creating really good heightmaps is a problem of its own, and thus I'd like to defer the details until later. For now, it is enough to use your favourite paint program (like PhotoShop, PaintShop, Gimp, ...) to create a gray-scale image (and thus a heightmap) by hand: The brighter the shade of gray of a pixel, the higher will this part of the terrain be – dark shades of gray correspond to low terrain heights, medium shades of gray correspond to parts of medium height, bright shades will become high parts. A fully black pixel (value 0) will correspond to the lowest point of the terrain, a fully white pixel (value 255) will correspond to the highest point of the terrain. The height of the highest and lowest points in the world were in turn defined by the AABB in the above step.

   Design tip: Make sure that the borders of your heightmap images are black where the terrain meets low walls or the adjacent surface of the floor of the alley. This ensures that there are no cracks between these parts. Proper and smooth transitions between brush based geometry and terrains are an important problem and subject to the next section.

   Your image must also

   - have a size of $129 * 129$ pixels (or $(2^n + 1) * (2^n + 1)$ in general), and
   - be saved as an 8-bit gray-scale `png` image[2] in the `Games/DeathMatch/Terrains` directory.

[2] RGB images in `bmp`, `png`, or `tga` file format will work, too. For such images, the *red* channel defines the height of the terrain, green and blue are ignored.

5. The final step is to re-open your world in CaWE, bring up the properties of the `HeightMap` entity, and set the "HeightMap File Name" to the heightmap you have just created (e.g. `Terrains/MyHeightMap.png`). You may also set the file name of the base and first detail texture. For now, it's enough if you choose one of the textures in `DeathMatch/Textures/` as the base texture (e.g. `Textures/Kai/rock2_nogoop_diff.png`), and leave the detail texture file name open.

6. Save the world and compile it as described in section 6.1.

7. Congratulations – you have just completed your first world with terrain support, and you're ready to run it in Ca3DE to see the result.

6.4.2 Working up the first tutorial

Here come a few additional thoughts and considerations about the previous tutorial, in question-and-answer format:

How exactly is the heightmap image aligned with the `HeightMap` entity? For the lateral dimensions as seen in the top-down view (XY-plane), this is pretty obvious: The heightmap image is scaled such that it fits exactly into (the AABB of) the `HeightMap` entity. For the vertical dimensions (terrain height), a pixel value of 0 (black) is matched to the bottom of the AABB, and a pixel value of 255 (white) is matched to the top.

Can I have terrains that are not square, but rectangular? Yes, by creating a rectangular `HeightMap` entity. However, note that the heightmap *image* of that entity must still be square ($(2^n + 1) * (2^n + 1)$ pixels). The heightmap information will be asymmetrically scaled to match the dimensions of the `HeightMap` entity. This means that there will be some distortion, but normally that's no problem as long as you don't over exaggerate.

Can I have terrains that are not square, but diamond shaped, or round? Not directly. As explained above, Ca3DE only considers the *AABB* of the brushes of a `HeightMap` entity for determining the spatial location and dimensions of the terrain. Therefore, it makes no sense to have `HeightMap` entities have round or diamond-shaped or other irregular (non-axis-aligned) brushes. You can, however, create the heightmap *image* to represent a laterally arbitrarily shaped terrain. Set the undesired parts to a height of 0 (black), and "hide" them with (resp. in) regular brushes. This yields unused parts of the terrain, but the LoD system of the terrain renderer will take care of them.

Can I have more than one terrain in a world? Yes, of course. In some cases it is much preferable to have several smaller terrains rather than just a whole big one. Simply create additional `HeightMap` entities, and set their properties. If you want to, several heightmaps can even share common heightmap and/or texture images.

What happens when a player tries to leave a terrain at its borders? Does he fall?

Well, there are several options for dealing with the borders of terrains:

- You can use simple solid walls, as shown in the screen-shots.
- You could create the terrain to have a rising slope near the border that is steep enough such that the player cannot climb over it.
- You could create a coastline (like in a beach or harbor scene), and put some deadly fishes into the water.
- You can freely add invisible clip brushes (or of course regular solid brushes), even in the mid of the terrain, or even have them interpenetrate it (like a crashed spaceship that halfway sticks in the ground).
- In the above screen-shots, when nothing else is there, the invisible sky brushes prevent that the player falls from the surface of the earth. (All terrains must be in some kind of room. Even the terrain in the above screenshots is in a room, whose walls happen to be textured with the "sky" texture.) Thus, due to the usual requirement that maps are to be sealed such that there are no leaks, this fact also naturally makes it impossible for players to fall off the ground.

6.4.3 More advanced terrain considerations

While the previous example introduced you to the basics of using terrains with Ca3DE, in practice it is often more difficult to make brushes and heightmaps that are well aligned to each other (i.e. form a smooth transition). The key for good alignment is to adjust the height of the brushes and/or the shades of gray in the heightmap so that the desired geometric (spatial) transition is achieved.

For example, consider two separate, distant buildings that are made of world brushes. Each building has a ground floor and an entrance, and you want to connect them, entrance-to-entrance, by terrain.

- If the ground floors of both buildings happen to be at the same height level, you could make sure that the heightmap near the entrances is black (zero), and use brighter shades of gray for hills between them. This is very similar to what we did in the above example.

- You could also make the heightmap near the entrances fully white (255), and use darker shades of gray for valleys between them. (Of course this also requires appropriate adjustments to the brushes of the `HeightMap` entity.)

- If the ground floors of both buildings happen to be at different height levels, you could make the heightmap to be black near one entrance, and white near the other. Again, this requires setting the brushes of the `HeightMap` entity appropriately. The disadvantage is that you can neither have terrain that is lower than the lower floor, nor terrain that is higher than the higher floor.

The next tutorial is about the most general case, where both (or even more) floors have arbitrary height, and you want to combine them with terrain of arbitrary height.

6.4.4 Tutorial 2: Two solutions for the general case

What makes the combination of brushes and terrain difficult is simply the fact that CaWE is not yet able to handle both brushes and terrains simultaneously. So instead of editing "both-in-one", we will have to use CaWE for editing worlds and another program for making and editing terrains.

I have worked out two methods for solving the problem. With the first method, the "emphasis" is on the terrain, whereas with the second method, the emphasis is more on the buildings.

The first method works as follows: Suppose you want to make a level with a large terrain that has smaller buildings on it (the emphasis is on the terrain), like for example a beach scene with several club houses.

1. Create the `HeightMap` entity where you want to have your terrain later, but do not yet create any buildings or other brush structures on it. Also do not bother to set any entity properties at this step (i.e. the "HeightMap File Name" remains blank for now).

2. Save the world, quit CaWE, and create the heightmap for the terrain as you desire.

3. It's not strictly necessary, but at this point you should make sure that you have small plateaus (patches of a single, common shade of gray) where your buildings will be located later, because it's usually easier to build something on flat terrain. Of course, you can also still do this later, and repeat the necessary steps.

4. Use the `HmIntoMap.exe` tool. This tool takes the (unfinished) map that you previously saved to disk and inserts `IgnoreMeSolid` entity brushes where the heightmap-defined terrain(s) would be. *Their only purpose is to visualize the terrain by approximating it with brushes that resemble the heightmap.* This is for your editing convenience after you re-loaded the map into the world editor. Note that the Ca3DE engine will entirely ignore these brushes (it uses the original heightmap instead)! Therefore it makes no sense to manipulate these brushes in CaWE in any way, because they're thrown away anyway. (Of course you may delete or hide them if you don't like them, or if they get into the way.)

   For using the `HmIntoMap.exe` tool, you have to specify the map name that you want to have brushes inserted into, and the path to the game directory, such that it can find the heightmap image. Example:

   ```
   C:\Ca3DE-MDK\Ca3D-Engine> HmIntoMap
   Games/DeathMatch/Maps/MyMap.cmap Games/DeathMatch
   ```

   The `HmIntoMap.exe` tool automatically creates a backup file before it changes the actual map file. If you specify `-nB` as a third command line parameter, no backup file is created.

5. Re-load your map into CaWE and continue editing: First, bring up the `HeightMap` entities properties dialog, and set the "HeightMap File Name" to the heightmap

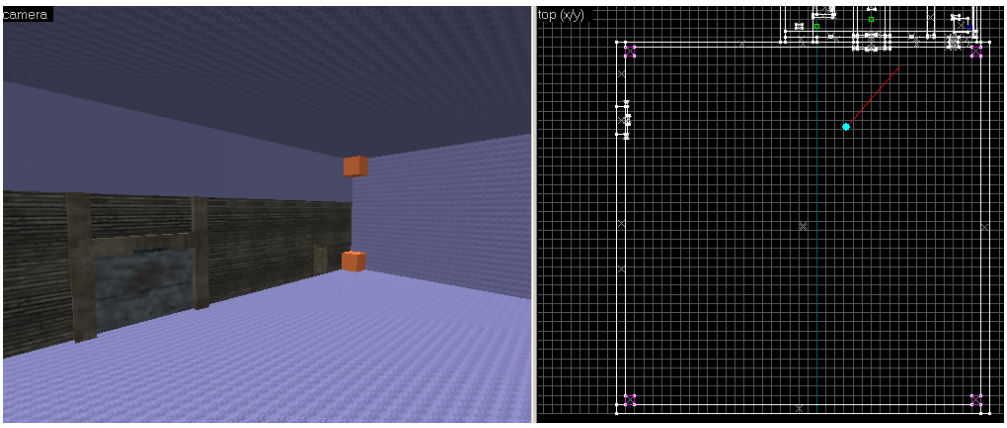
image you have created in the previous step. Then continue by creating the club houses and other brush geometry. After the above steps, it will be easy to align everything to the approximated terrain.

The second method works "the other way round": Here you have already created the essential buildings, and now for example you want to fill-in the space between the buildings with terrain.

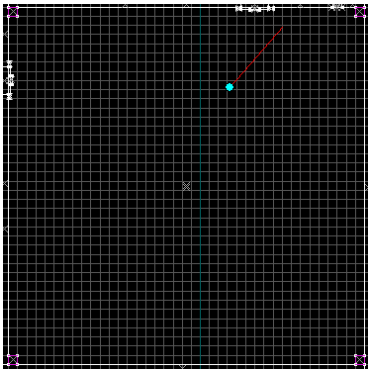
1. Start making your level by creating the buildings (at least their gross walls), the main components, and other essential brushes and entities. It is not necessary to make them very detailed at this stage, but the brushes that you place here should pretty much reflect the gross layout of your level.

2. Create the `HeightMap` entity as desired. Its "HeightMap File Name" and other properties are left blank for now.

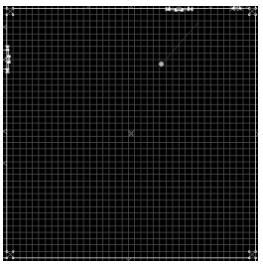
3. Before you close the world editor, take a screenshot of the top-down view (XY plane, 2D birds-eye view) of the map. Make sure you have the zoom set such that the *entire* `HeightMap` entity is visible. If you don't know how screen-shots are made: Simply press the `Print Screen` key on your keyboard. This copies the contents of the screen into the clipboard. You can then run your favourite image processing program to paste the clipboard contents into the workspace. The following figure shows how (part of) this initial screenshot might look like:



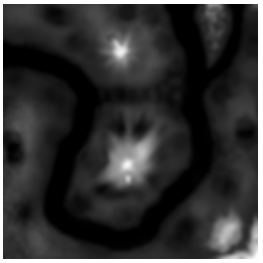
4. Use your image processing program to crop the screen-shot such that only the (quadratic) part of the `HeightMap` entity remains. It might be a good idea to keep not only the `HeightMap` entity, but also one or two "border" pixels at each side. See the figure to the right for my result from the previous step.



5. Scale the remaining image to a resolution of $(2^n + 1) * (2^n + 1)$ (e.g. $129 * 129$) pixels, and convert it into an 8 BPP gray-scale image. The figure to the right shows the result.



6. This gives you an initial heightmap that already has some lines and pixels in it, indicating the brushes on (and possibly in the close neighbourhood of) your terrain. Of course we will eventually overdraw all these lines. Their only purpose is to give you initially a precise idea about where brushes are located in your heightmap!
7. In places where brushes are indicated in our initial heightmap, you usually want to have the terrain at an exactly specified height. That is, you have to overdraw the brush lines with the right shade of gray, such that later the terrain height matches the height of the indicated brush. How is the proper shade of gray determined?? Well, I usually do this by linear interpolating in the side- or front-view in CaWE: Figure out at what height you want your terrain to be near the brush in question, and compute the relative height wrt. the `HeightMap` entity (this is a number between 0 and 1). This is at the same time the proper value for the right shade of gray.
8. For all other parts of your heightmap, simply draw what your heart desires. When you're done, save the result to disk. Finally, don't forget to re-start CaWE and set the "HeightMap File Name" in the `HeightMap` entity properties dialog to the file you have just saved. *Congratulations, you're done!* The figure to the right shows my final result.



Note that although is seems that both methods are mutually exclusive, they are actually independent of each other. You can use one or both for the world you're making, use them repeatedly, and use them in any order. For example, depending on whether you want to make a small garden or a large coastline, you'll find that one method works better than the other. All this is just a matter of taste and of the details of the desired result. Also note that you can modify the heightmaps long after the world was compiled.

6.4.5 Creating heightmaps

Until now, I have not yet said much about *how* heightmaps are created. During my research, I've downloaded and tested about half a dozen terrain editors for creating heightmaps. Unfortunately, I've found a *huge* amount of problems, and each of the programs suffer from at least one:

- Frequent program crashes or freezes.
- *Very* bad GUIs.
- Support only for side lengths of 2^n, but side lengths of $2^n + 1$ are required.
- Relative vs. absolute heights: Most programs work in "relative height" mode. That means that they permit to raise terrain higher than 100% (white) or lower terrain below 0% (black), by adjusting the gray-scale value of the *other, remaining* terrain accordingly. This might be desirable in some applications, but for our purposes, "absolute height" editing would be required. (100% (white) corresponds to the upper ceiling of the `HeightMap` entity, 0% (black) to the bottom. Period.) This mis-feature is hard to understand if you have not experienced it, but it is a very dangerous (and silent!) killer!
- Idiotic import/export facilities. Often only raw and native formats are supported, where at least gray-scale png image support would be highly desired.

Many of these problems are so serious that they make the affected program entirely useless, and for me, sadly, not a single one survived (not even TerraGen)!

As at the time of this writing CaWE does not support integrated terrain editing (what would be the ideal solution), my next best recommendation is to create your terrains with your favourite, general, all-purpose image processing program, like PhotoShop, PaintShop, Gimp, ... I've created the terrains that ship with Ca3DE this way. These image processing software will not give you a terrain wire-frame preview of the grayscale image that you are editing, but it usually suffers of none of the above mentioned problems.

Finally I'd like to mention that once you have created a heightmap in some way or another, *TerraGen* is an excellent tool to create *base (ground) textures* for the terrain. How this is precisely done is enough stuff for yet another tutorial. For now, I'll just give a very short overview over this sometimes not-quite-so-easy process: First, you'll have to import your heightmap into TerraGen. This might require the help of a TerraGen importer plug-in, and/or the conversion of your heightmap image into an image file format that TerraGen and/or the plug-in can understand. Then you'll have to setup the terrain dimensions in TerraGen (lateral size, altitudes), which you can derive by scaling the dimensions of the `HeightMap` entity in CaWE. Finally, you'll have to setup the TerraGen materials that determine how your terrain will look, setup the sun such that it is full-bright and doesn't cast any shadows, turn off all atmospheric effects, and setup the camera for orthogonal projection.

6.4.6 Final words about terrains

I hope I didn't scare you too much. Making terrains, heightmaps, and terrain textures *is* currently difficult, at least when you're new to it. Integrating terrains into maps and working with them in general is difficult, too.

My aim is to augment CaWE as soon as possible such that it can provide more and better help with terrain editing, and I hope that that will take out 90% of the problems.

6.5 Porting existing worlds to Ca3DE

If you have already made worlds for other games (e.g. Half-Life DeathMatch), you possibly wish to convert them for a Ca3DE MOD, like the *Ca3DE DeathMatch MOD*. You save a lot of work by doing so, and in general, porting "foreign" worlds to Ca3DE is easily done. Most likely, there are only two problems you will encounter: texture and entity mismatches. The following list summarizes what you have to know and consider about porting:

- Because the existing entities in your world were defined and made for another game, they will mismatch with the Ca3DE MOD entity definitions! Note that there are two kinds of mismatch:
  - Some entities exist both in the other game as well as Ca3DE (for example translucent brushes, ladders, ...), but their names and/or their properties differ.
  - Other entities only exist in the other game and Ca3DE has no equivalent entity at all, or vice versa! For example, Ca3DE DeathMatch has different weapons, but unfortunately lacks of many of the advanced entities of other games. I am very sorry, but at present you have to find some kind of work-around in such cases. However, I am doing my best to improve the situation in the future.
- Here are some suggestions to fix the entity mismatchings:
  - It's probably the safest (easiest, but most expensive) to simply delete *all* existing entities from your world, and only keep the geometry. Then, recreate them from the Ca3DE (DeathMatch) entities.
  - In many cases it might be preferable to simply open the *properties dialog* for each entity, and to reset their properties there. Just make sure that you actually check out each individual property tag.
  - If you are very experienced, you might solve many mismatchings directly by loading the `map` file into a text editor.

  Although Ca3DE ignores unknown entities (e.g. entities left from other games), and fills-in default values for unknown properties, it is still best to do the conversion carefully in order to avoid unexpected results. For example, be specially

careful with `func_wall` entities: you'll have to reset their *rendermode* properties (translucent, blue becomes invisible, ...) *explicitly*!

- The worlds have to be *re-textured*. Re-texturing can either be done with the "Replace" tool of CaWE, or (if you are very experienced) directly within the `map` file with a text editor, or even with a Perl script.
- Note that it is *not* advisable to use textures from WAD files other than the WAD file you have made yourself with `MakeWAD.exe`. Foreign WAD files do not contain any new-style multi-image texture information (normal-maps, specular-maps, ...), and you had to manually extract their contents somewhere into the `Games/MyMOD/Textures` directory hierarchy, or else the engine cannot find the textures when loading your world.
- Don't forget to change the name of the sky (environment) map.