

Harrison Mercer  
Senior Project Design Document  
September 22<sup>nd</sup>, 2008

## Overview

This project is a map editor. The idea behind every feature is to allow the user to design a level for a video game visually. A GUI system provides the user with an interface to choose objects, manipulate them, and give them specific settings, such as solid form, mass, and an ability to move. The player can move around the map while it's being built with camera controls based on the keyboard. Placing objects onto the map can be done with the mouse or with the GUI entering individual numerical values. The objects themselves are called tiles loaded from a tileset file and then selected from a drop down menu. The user can name the map, choose background music for the map with a .mp3, .ogg, or .wav file, and then save the map in a .mapfile format. There are also options to load a map already made and to clear the current map of all objects. Once an object has been placed, the user can select it again by clicking on it with the mouse.

The editor will also come with a loader. The loader is bunched with code that automatically loads all data stored in the .mapfile file into classes pre-defined. From that point on, a programmer could do whatever he wants with the classes and make a game out of the loaded data.

## Requirements

The map editor will be using the following libraries:

- OGRE3D (graphics)
- CEGUI (GUI interface)
- ODE (physics)
- Audiere (for audio)

Under the licensing terms of Ogre, their libraries cannot be directly compiled into the program and then sold, so all libraries will be included with the download of the map editor and imported at runtime.

Ogre runs seamlessly with Crazy Eddy GUI, so the level itself is rendered with Ogre and loaded into Ogre with the loader and the GUI system is rendered with CEGUI.

Collision detection between objects with each other and objects with the mouse is handled with ODE. The user has the option to change the bounding boxes around each individual object to ensure better collisions.

The music is not actually loaded in the map editor, but its file name is stored in the .mapfile and the music is loaded in the map editor loader with Audiere's streaming techniques.

## Modes

There are three different modes the map editor can be in. The editor can only be in one mode at one time and can switch between them either via shortcut keys or interacting with the GUI.

- Normal Mode
  - This is the default mode of the editor. All of the GUI is functional and the player can select objects already placed on the map.
- Place Mode
  - The editor enters this mode when the Place button on the GUI is pushed. The user can place objects onto the map in this mode. During this time, the user cannot use the GUI, select objects, or enter Camera Mode.
- Camera Mode
  - In this mode, the camera looks around via mouse movement and move by using the keyboard. In this mode, the user cannot use the GUI or select objects.

## Architecture

The GUI system is separated into three tabs:

- File
- Object
- Transformation

### **File Tab**

#### 1. Tileset (edit box)

- The user enters the file name of the tile set and loads all tiles in that file by clicking a button.
- The text entered in the box must end with “.tileset”. This ensures an ability to implement different file names in the future.

#### 2. Map Name (edit box)

- The user enters the name of the map and confirms it with a button.
- This is the name of the .mapfile.
- The text entered does not require .mapfile to be at the end of the name. The program will automatically add the file extension if it not typed in by the user.

#### 3. Music File (edit box)

- The user enters the file name of the desired background music for the level and confirms it with a button
- The name must include either “.mp3”, “.ogg”, or “.wav” at the end of the file name.

#### 4. Save (button)

- Clicking this button saves the map file in the current directory of the map editor.
- The file name is determined by what is currently typed in the Map Name edit box.

5. Load (button)

- Clicking this button unloads any data loaded by the program so far and loads in a pre-made .mapfile.
- The .mapfile loaded is what is currently typed in the Map Name edit box.

6. Clear (button)

- Clicking this button wipes all objects from the map while still keeping the previously loaded tileset, map name, and music file set as to what they were before the button was clicked.

7. Quit (button)

- Clicking this button exits the map editor.
- The map is **NOT** automatically saved.

## **Object Tab**

1. Tile (drop box)

- Once a tileset is loaded from the File tab, this drop box becomes filled with all objects listed in the .tileset file.
- Selecting one of these shows the object on the map but does not place it.

2. Place (button)

- Clicking this button puts the editor into Place Mode, where mouse movement changes the position of the selected object. If no object was selected before the button was clicked, then a new temporary object of the selected tile is made.
- The user can make copies of an object already placed on the map by selecting it and going into Place Mode.

3. Static (checkbox)

- This checkbox represents whether or not the object moves.

4. Solid (checkbox)

- This checkbox represents whether the object has physical attributes or if other objects can pass through it as if it did not take up physical space.

5. Mass (edit box)

- The user enters the mass of the object and confirms it with a button.

6. Bounding Box (drop box)

- The user selects one of three bounding box options

1. Rectangle

- Sets the bounding box of the object to be an axis-aligned bounding box determined by Ogre and converted into ODE.

2. Sphere

- Set the bounding box of the object to a sphere with a radius that encompasses the entire object, determined by Ogre and converted into ODE.

3. Tri-mesh

- Set the bounding box of the object to be a mesh of triangles built around the polygons that make the actual mesh of the object.

7. Remove (button)

- Clicking this button removes the selected object from the map.

**Transformation Tab**

1. Selected Object (edit box)

- The information displayed here represents the name of the currently selected object

- The user can type in a new name for the object and confirm it with a button

- The name cannot be a name currently given to an object that already is placed on the map

2. Type (drop box)

- The user selects one of three transformation options.

### 1. Positon

- The numerical values entered in the transformation options effect the position of the selected object.
- This option is a translation and moves the object from its current position the entered amount.

### 2. Rotation

- The numerical values entered in the transformation options effect the rotation of the selected object
- All angles are measured in degrees

### 3. Scale

- The numerical values entered in the transformation options effect the scale of the selected object.
- This option is a resizing and scales the object from its current size by the entered amount.

### 3. X (edit box and button)

- Based on the type selected from the drop box, this effects the x-axis by the entered amount and confirmed with the button.

### 4. Y (edit box and button)

- Based on the type selected from the drop box, this effects the y-axis b the entered amount and confirmed with the button.

### 5. Z (edit box and button)

- Based on the type selected from the drop box, this effects the z-axis by the entered amount and confirmed with the button.

6. All (edit box and button)

- Based on the type selected from the drop box, this effects all axes by the entered amount and confirmed with the button.

## Shortcut Keys

The Map Editor has several different shortcut keys that are required to move around the map efficiently while editing it.

### **Normal Mode**

This is the default mode of the editor where the user is simply selecting objects or interacting with the GUI.

- CTRL + C
  - This switches the map editor from Normal Mode to Camera Mode
  - The user can switch back to Normal Mode by hitting this key combination again
- CTRL + U
  - This is the “undo” feature. The last action taken will be reversed and the map will be restored to the point before the action took place
  - Action that can be undone:
    - Applying a transformation to the object with translation, scaling, rotation, or setting it's position on any or all axes.
    - Changing an object's name.
    - Removing an object.
    - Changing an objects bounding box type.
    - Placing an object in a different location using the place button.

### **Place Mode**

This is the mode the editor goes into if the Place button is pushed. While in this mode, the object selected before the Place button was pushed can be moved around the map with the mouse along the X- and Z-axes or the X- and Y-axes.

- SPACEBAR
  - Creates a new instance of an the selected object at it's given position, orientation, and scale.
    - The object created will be named “ 'object's name' + \_ + n” where “n” is the number of objects with that name placed on the map already. So hitting SPACEBAR three times with a selected object “Barrel” will create three new instances on the map named “Barrel\_1”, “Barrel\_2”, and “Barrel\_3”.
- LEFT SHIFT
  - Switches the axes the object is currently moving based on mouse movement. If the object is currently moving along the X- and Z- axes, then it will switch to the X- and Y- axes and visa versa.
- ESCAPE
  - This key brings the editor out of Place Mode and puts it into Normal Mode.

### **Camera Mode**

The camera goes into a first-person “god mode” and looks around via mouse movement.

- A or LEFT
  - The camera strafes to the left of its current position until the key is let go.
- D or RIGHT
  - The camera strafes to the right of its current position until the key is let go.
- W
  - The camera moves forward in the direction it's looking until the key is let go.
- S
  - The camera moves backward from the direction it's looking until the key is let go.
- UP
  - The camera flies upward from it's current position until the key is let go.

- DOWN
  - The camera drops downward from it's current location until the key is let go.
- Right Mouse Button Click
  - The camera's position resets to it's initial position when the editor was opened.
- CTRL + C
  - This key combination brings the editor out of Camera Mode and puts it into Normal Mode.

**\*NOTE\*** The camera's movement is not axis-aligned but based on the orientation of the camera at that specific time.

## MapLoader.h File

The MapLoader.h file contains a function and a structure. Including it in your code requires the Ogre.lib file to be properly linked into your project.

### **MapObject Structure**

- name
  - An Ogre String class that holds the name of the object.
- mesh\_name
  - An Ogre String class that holds the name of the Ogre Mesh associated with the object.
- is\_solid
  - A boolean value storing whether or not the object has physical mass.
- is\_static
  - A boolean value storing whether or not the object moves.
- mass
  - A floating point value storing the mass of the object.
- bounding\_box\_type
  - An integer value storing the bounding box type of the object and can be one of only three values.
    - BBOX\_RECT (1)
      - An axis-aligned bounding box type.
    - BBOX\_SPHERE (2)
      - A sphere bounding volume type.
    - BBOX\_TRI (4)
      - A tri-mesh bounding volume type.
- position
  - An Ogre Vector3 class storing the x, y, and z coordinates of the object.

- scale
  - An Ogre Vector3 class storing the x, y, and z scaling values of the object.
- orientation
  - An Ogre Quaternion class storing the w, x, y, and z orientation values of the object.

### **loadMap Function**

- First parameter (String fname)
  - An Ogre String class representing the name of the .mapfile.
- Second parameter (std::vector<MapObject> &vref)
  - The user passes an std vector of MapObject structures by reference and it is filled with all the MapObjects of the .mapfile.
- Third parameter (String &musicref)
  - The user passes an Ogre String by reference and it is set equal to to the name of the music file representing the background music of the map.

The programmer can use this header file to load the map data then parse the MapObjects returned into their own game object class.