An “Open Robot Battle Near Earth”

Remake of “Nether Earth” by team 2:

Brief Specification, Design, and User Manual

COMP371 Winter 2012

Team Members:

Jonathan Bergeron jonathan.bergeron@hotmail.com

Jeff How jeffrey.f.how@gmail.com

Robert Jakubowicz robert@robert-jakubowicz.com

Stéfanie Lavoie lavoie.stefanie@hotmail.com

Addison Rodomista addisonrodomista@gmail.com

**Table of Contents**

1. Introduction 2
2. Techniques 3
   1. Modeling Methodology 3
   2. Animation 3
   3. Texturing 3
3. Software Architecture 4
   1. Description 4
      1. Logic 4
      2. Model 4
      3. Helper 5
   2. Code Originality 5
4. Challenges 6
   1. Frame rate 6
   2. Shadows 6
   3. Music 6
   4. Portability 6
5. Bonus Features 6
   1. Music 7
   2. DeathMatch 7
   3. Joystick control 7
   4. Portability 7
   5. Menu 7
6. User Manual 8
   1. Classic Nether Earth mode 8
      1. Camera Control 8
      2. Light Control 9
      3. Player Control 9
   2. DeathMatch mode 10
      1. Camera Control 10
      2. Player Control 10
7. References 11
8. Appendix 12

# 1. Introduction

The purpose of this document is to provide the reader with an in-depth understanding of our version of the game Nether Earth that was done for the COMP 371 class.

The game was design according the teacher’s requirements during the whole semester. Some extra contented was added to give the game a better finishing touch.

This document will explain how we created this project, what techniques we used and the challenges we faced while creating this project.

# 2. Techniques

### 2.1 Modeling Methodology

In this project, all our models were made using only primitive objects. To create those primitive objects, we decided to only user the glu functions from OpenGL or we created our own using GL\_QUADS or GL\_TRIANGLES. We created a GeoHelper class which has functions to create cubes, cylinders and triangular prism. With these useful helper functions, we could all use the same functions and we only needed to scale the primitives to fit our models. We never used the GLUT functions because we were told that they weren’t good for the framerate and they don’t generate texture maps.

### 2.2 Animation

For the basic animation that we have in this project, we are only using glTranslation to move the objects around.

For the nuclear explosion, we have a timer that scales the mushroom explosion after a certain amount of time. Once the animation is over, we make the mushroom disappear.

### 2.3 Texturing

All our textures are loaded using the “imageloader” and “textureManager” classes. The textures are set using the OpenGL function glTexCoord2f. All the primitive objects we created are all using the same glTexCoord2f values, but we are setting the glBindTexture before calling the helper function.

# 3. Software Architecture

### 3.1 Description

The architecture of this project uses object-oriented programming with C++. The project is split in three different sections: Logic, Model and Helper. You can see the appendix for screen capture of our class diagrams.

#### 3.1.1 Logic

The Logic section contains as the name suggests, the logic of the game. This is where we find the main classes for the cameras, game, player, light and robot.

First, we have a general Camera class that is an abstract class used for camera orientation and movement. Every other camera classes extend the main Camera class and modify the movement functions as needed. We have a circling camera that has a bird eye view of the scene. This camera can rotate around the scene and has a spotlight that can be toggled on and off. There is also the commander camera that is a strategy game type camera. It can zoom in and out and rotate around a look-at point. We also have light cameras that are located on top of each spotlight. Finally, we have a free look camera where the camera is control with the mouse.

The LevelRenderer class is where the maps are being drawn. We are using a 2D array that contains the number of the model that should be drawn on the map. The LevelRenderer class reads the data of a map from a text file. The data is then saved into the 2D array and then drawn for the game.

The game class is where the logic of the game is computed. This class is the class that calls the render function of the level renderer and the players.

The player and playerInput classes contain everything related to the player. The player class holds all the information about the player while the playerInput class parse and execute all the player commands.

Finally, the spotlight class sets the standard spotlight components used for all instances of spotlights on the field.

#### 3.1.2 Model

The model section contains all the models used in the game, the texture manager and the material logic.

Every graphic element in the game extends the Model class. This class contains the render functions which every element need to redefine in their own class. Since every element is based on the Model class, this allows us to create a model array in the level renderer to draw the elements on the map. All the elements were created using the basic primitive objects.

The texture manager is a singleton class called by all the models to find textures. It uses a map to make textures easily retrievable. It also uses the imageloader class, which is used to convert .bmp files into Image objects.

The materials of the game are all based on the Material class. This class controls the reflection, shininess and the diffuse properties. In the game, we have 4 different materials: default, metal, organic and rock. The default material is a normal material with no reflection and no shininess. The metal material gives a metallic look to the robots and is very shinny. The organic material is applied to the grass and gives a higher tint of green. The rock material is applied to the rocks and the walls and gives them a higher tint of brown.

#### 3.1.3 Helper

The helper section contains the AntTweakHelper, the DirectoryManipHelper and the sounds library. The AntTweakHelper class was used to setup AntTweak and bind the variables from the game to the debug menu.

The DirectoryManipHelper is used by the TextureManager to retrieve the file names for the texture files.

The sounds library are used to load the background music and sound effects.

### 3.2 Code Originality

All the code used in this project was made from scratch from the members of Team 2. The only code that we took from the web is:

* The image loader class that loads bitmap images to the project. This class was given to us during a lab session. We did not modify this code.
* The SDL libraries for loading the music and the Dirent API for file management are also not ours. Nothing was modified for this.
* The AntTweakBar is used for the debugging window. The output in the makefile has been modified to accommodate our project structure
* The shadow matrix code was modified to work with our project. A lot of the code was changed.

# 4. Challenges

### 4.1 Frame rate

During the production of the project, we had a lot of problems with the frame rate. At one point, we had a frame rate of 10 per seconds. We made a lot of changes to fix this issue. One of those changes was to put every model inside a display list. This was very helpful considering that we are re-rendering the game a lot. By activating the display lists, we ensure that the vertexes and the pixel data are evaluated only once and not over and over again. We can also toggle on/off our team number that was on the grass title.

### 4.2 Shadows

### 4.3 Music

### 4.4 Portability

Since we were developing our project on Mac and Windows at the same time, we often encountered problems where the code would work on one platform but not on the other. For example, the original help menu was using a GLUT subwindow and this was not working at all on Mac. This had to be fixed by completely removing the subwindow code and creating a new orthogonal view separated from the game’s logic. We also had problem with the directories path since Windows and Mac didn’t read it the same way. We had to implement a helper that checks which platform the project is running on change the directory file according to that.

# 5. Bonus Features

### 5.1 Music

We decided to add music and sound effects to the project, because we felt it was an important aspect of a game. The music was added with the help of the SDL libraries.

### 5.2 Deathmatch

We added a second player mode to the game. By selecting the Deathmatch option at the beginning of the game, the screen is split into two and two players can play the game. Each of the players have to destroy the robots of the other player. You can also activate split-screen view by pressing F8 in the classic Nether Earth mode.

### 5.3 Joystick control

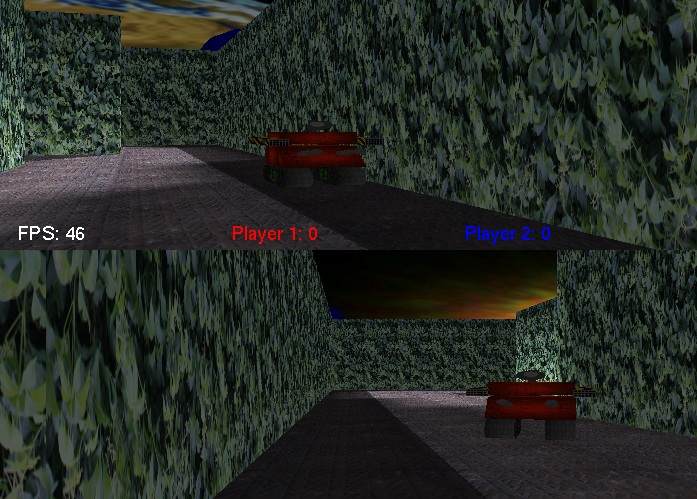
When in split-screen mode, the second player can be controlled with the use of a joystick controller.

### 5.4 Portability

The project can be run perfectly on both Windows and MacOS. This should also work on Linux.

### 5.5 Menu

A menu was added at the beginning of the game to make the selection between the different gameplay modes and the maps easier.

  
This is a picture of the DeathmMatch mode

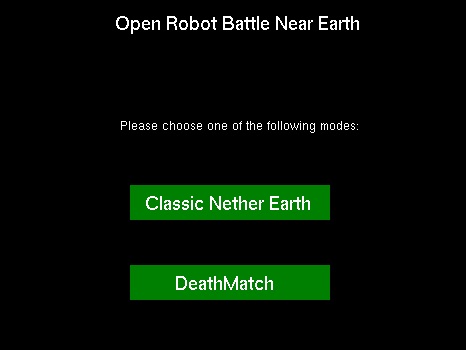
# 6. User Manual

To compile the project, please follow these instructions:

1. In Microsoft Visual Studio, click on the project.
2. Click on Properties
3. Click on Debugging
4. Change Working Directory to $(SolutionDir)

The project will work automatically in NetBeans

At the beginning of the game, you will have the choice of choosing between two different game modes. You can select one of these modes with the mouse. These games modes have different controls associated with them. Here is a list of these controls:



### 6.1 Classic Nether Earth mode

#### 6.1.1 Camera Control

To control the camera, you can press the following keys:

F2 Activate free look camera

F3 Activate normal camera

F4 Activate toggle bird view camera

[ ] Modify the “roll” of the camera

/ Reset roll/pitch

Page up/down rotate the camera

End reset rotation angle

= zoom in

- zoom out

0 reset zoom

1 camera of light 1

2 camera of light 2

3 camera of light 3

4 camera of light 4

#### 6.1.2 Light Control

To control the lights you can press the following keys:

p toggle ambient light

5 toggle light 1

6 toggle light 2

7 toggle light 3

8 toggle light 4

9 toggle all lights

#### 6.1.3 Player Control

The player can use the following keys to play the game:

F1 Cycle between wireframe/smooth shading/ flat shading

a/s/d/w Move the player controller

z toggle the skyboxes

F8 Activate DeathMatch

Space bar Make the player controller go up

### 6.2 Deathmatch mode

#### 6.2.1 Camera Control

Arrow keys move the robot camera

Moving mouse left and right move the camera left and right

#### 6.2.2 Player Control

a/s/d/w Move the robot

left mouse click Shoot missiles

# 7. References

7.1 Textures

Fence (grungemetal): <http://vortex-x.deviantart.com/art/green-white-metal-wall-118983148>

Mountain: <http://www.sharecg.com/v/16736/gallery/6/Texture/seamless-rock>

Floor (rusty\_floor): <http://www.mb3d.co.uk/mb3d/Metal_Rusty_and_Patterned_Seamless_and_Tileable_High_Res_Textures.html>

Team Number (team): <http://free4illustrator.com/2009/04/seamless-reptile-textures-and-photoshop-patterns/>

Player (gold): <http://www.psdgraphics.com/textures/brushed-gold-metal-texture/>

MetalVerticalLines: [www.webtexture.net/textures/6-high-resolution-metal-texture/](http://www.webtexture.net/textures/6-high-resolution-metal-texture/)

Brick, Concrete\_bare, Dirt, Ink, Marble, Metal1, Metal2, Metal 3, Metal4, Bipodf: [www.cgtextures.com](http://www.cgtextures.com)

Lightpost: <http://media.moddb.com/legacy/images/tutorials/30/308/gallery/t_508.jpg>

Camo: <http://farm4.static.flickr.com/3324/3628571124_51e7bbdff6.jpg>

Smooth metal: <http://hhh316.deviantart.com/art/Seamless-metal-texture-smooth-164165216>

skull: <http://bestpooltablesreview.blogspot.com/2011/04/buy-2d-glitter-skull-flights-100-micron.html>

Titanium: <http://www.jbdesign.it/idesignpro/metal.html>

mechanical: <http://www.svgopen.org/2003/papers/UsingSVGFor2DContentInMobile3DGames/index.html>

energy: <http://www.123rf.com/photo_9298276_glowing-energy-streaks-abstract-seamless-background-texture.html>

earth and moon : [http://frank.mtsu.edu/~njsmith/astr/lab02.shtml](http://frank.mtsu.edu/%7Enjsmith/astr/lab02.shtml)

sun: <http://www.dailymail.co.uk/sciencetech/article-1290919/First-photo-planet-circling-distant-sun-outside-solar-system.html>

nebula: <http://cs.astronomy.com/asycs/media/p/474777.aspx>

stars1.bmp: <http://www.khilafatworld.com/2012/01/poem-look-in-stars.html>

stars2.bmp: <http://www.therealfun.com/Stars-in-night-10535.html>

milky way: <http://www.moonphases.info/the-stars.html>

warning: <http://www.texturemate.com/content/free-texture-symbols-2010080401>

BlackDesign: [www.flickr.com/photos/torley/467773018/in/set-72157600105710970](http://www.flickr.com/photos/torley/467773018/in/set-72157600105710970)

7.2 Code

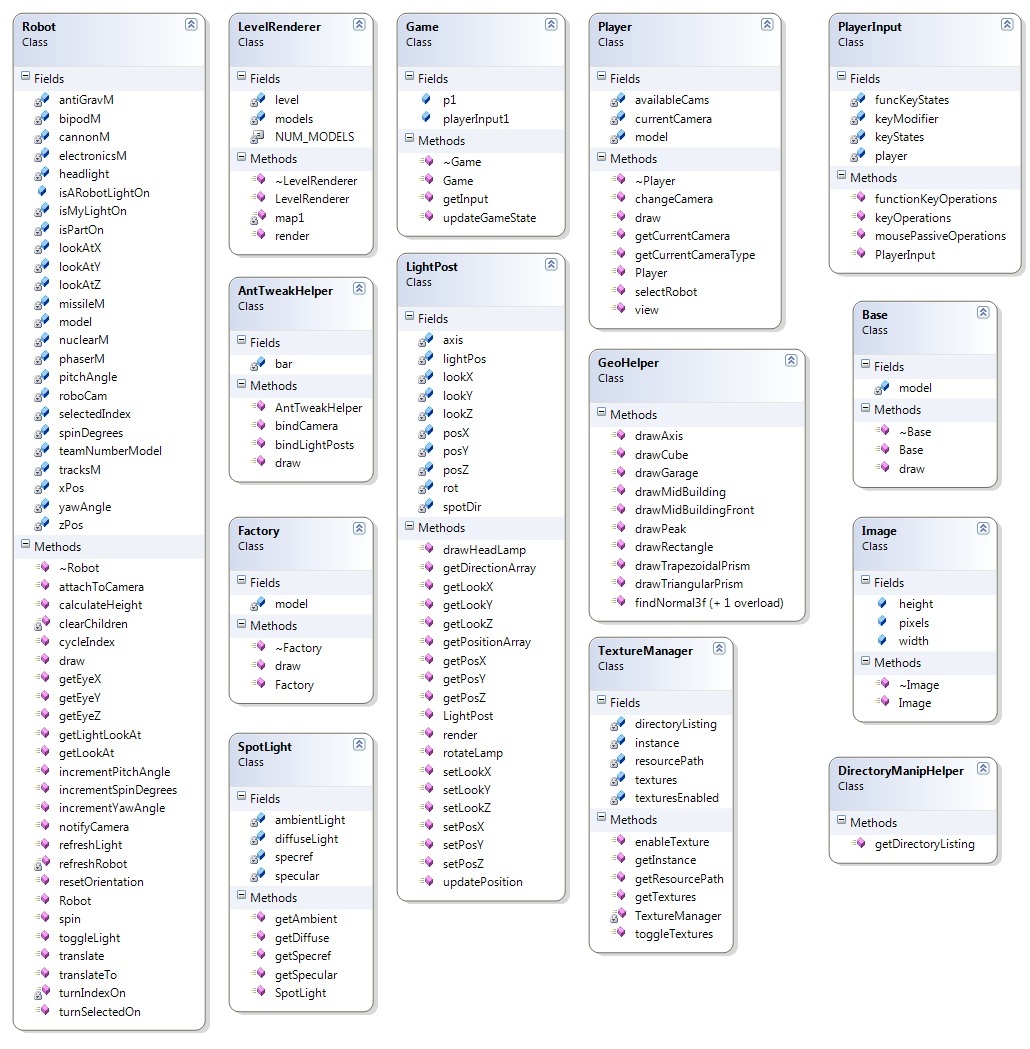
The image loader class file was taken from: [www.videotutorialsrock.com](http://www.videotutorialsrock.com). It was given to us during a lab session.

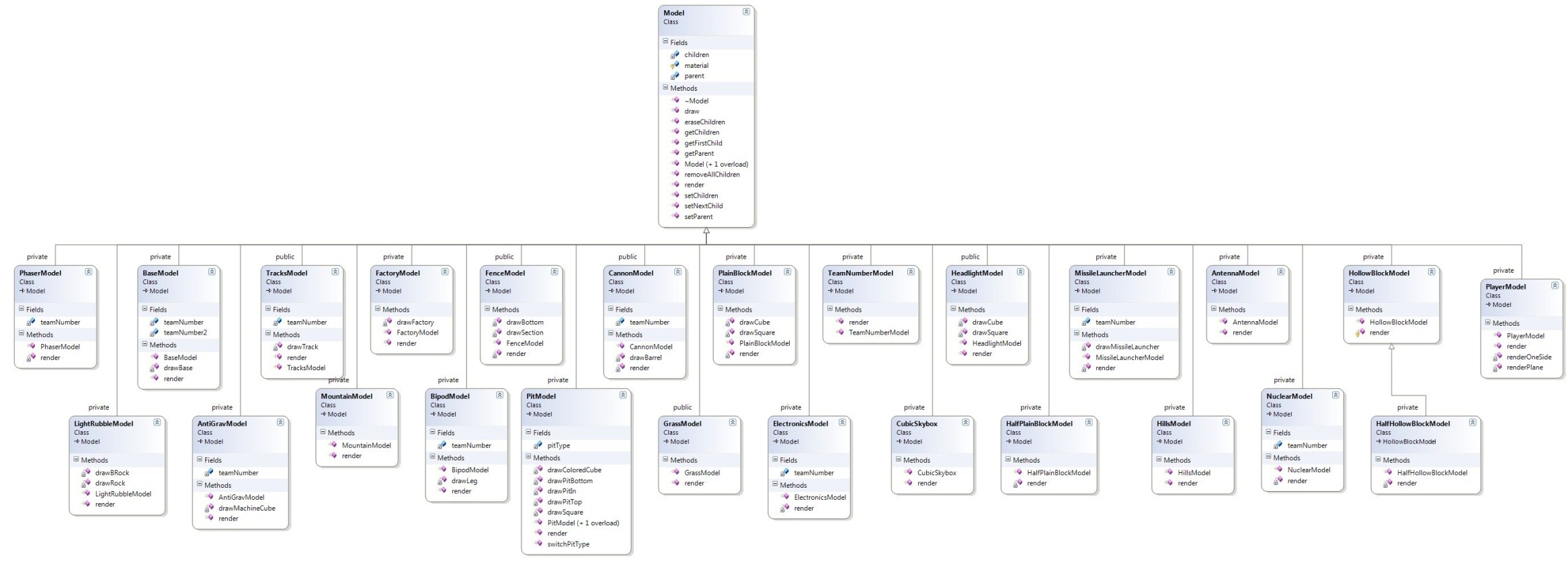
For the music, we used the sdl library and sdl mixer found at:   
http://www.libsdl.org/  
http://www.libsdl.org/projects/SDL\_mixer/  
  
The shadow matrix code is from: <http://www.opengl.org/archives/resources/features/StencilTalk/sld021.htm>  
  
AntTweakBar for the debugging window: http://www.antisphere.com/Wiki/tools:anttweakbar

Dirent API for windows : http://www.softagalleria.net/dirent.php

# Appendix

Exhibit 1: Class Diagrams





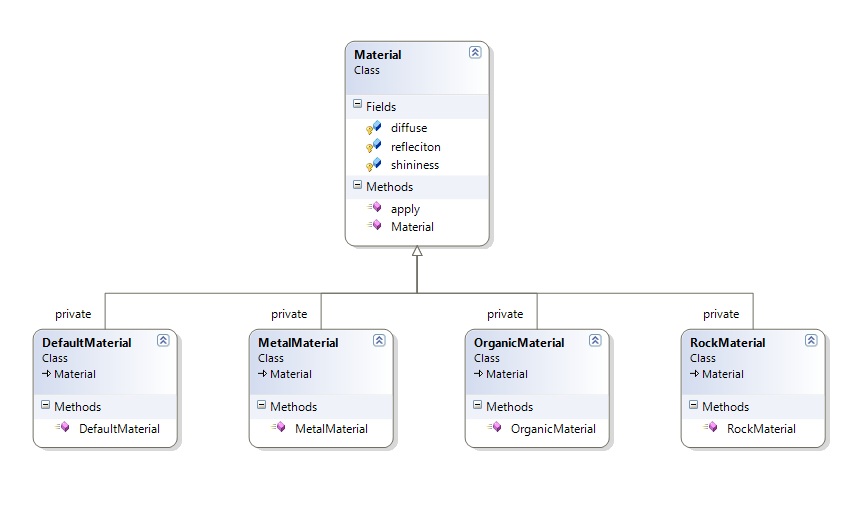
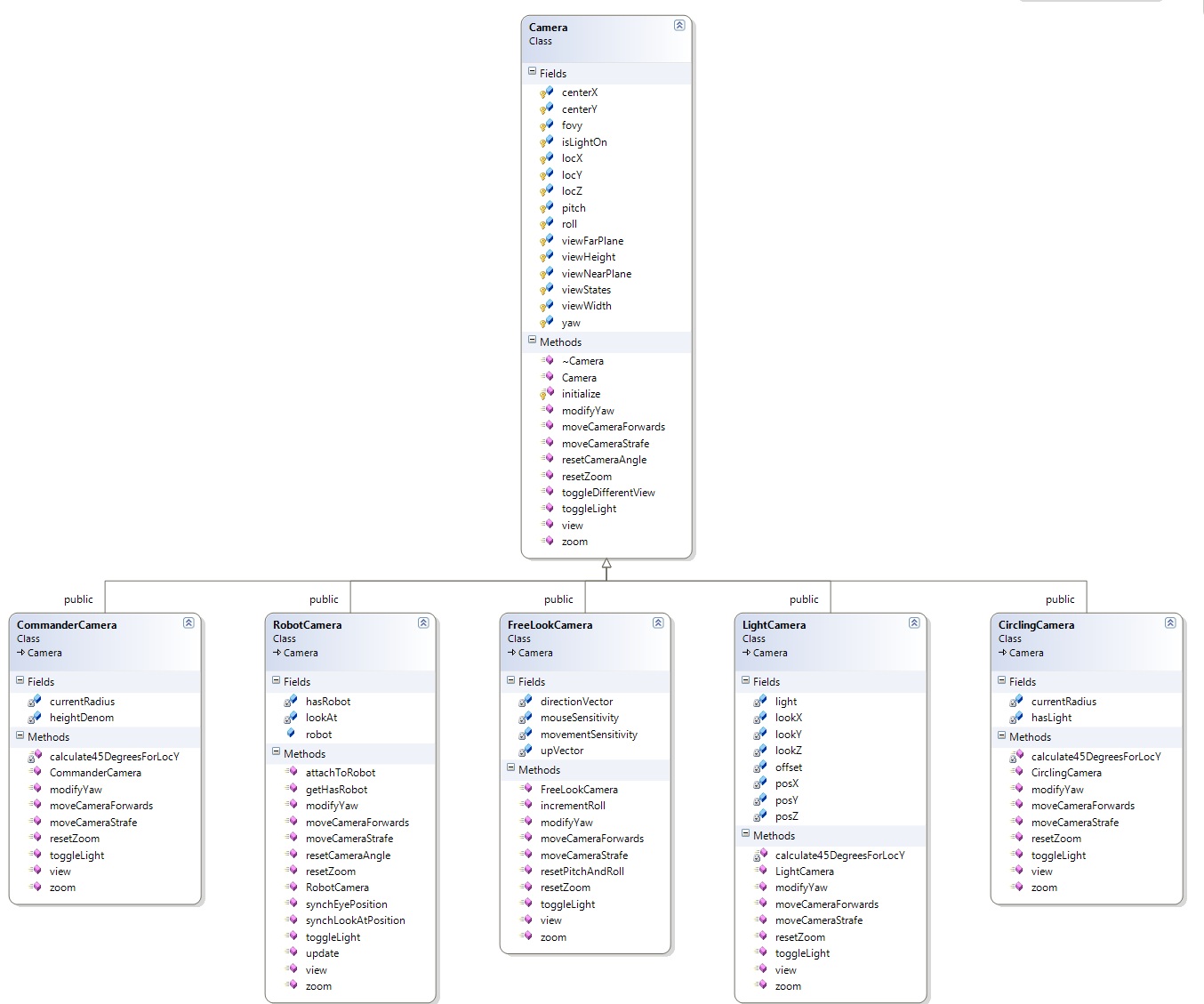


Exhibit 2: Screenshots

