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
Jeff How
Robert Jakubowicz
Stéfanie Lavoie
Addison Rodomista

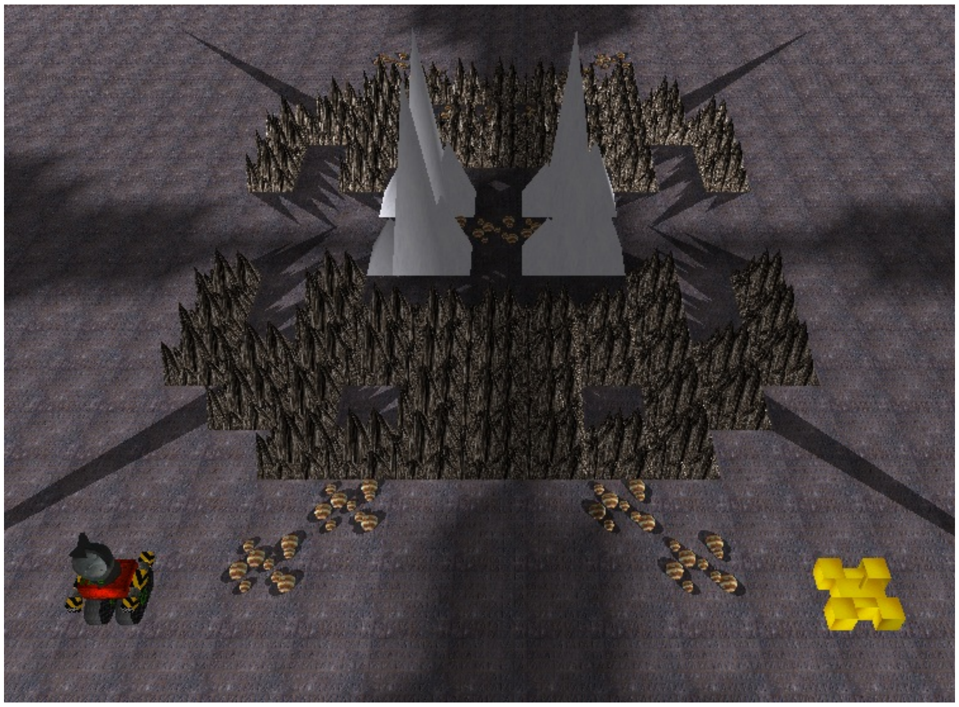
Table of Contents

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

Our models were primarily made using primitive objects. We accomplished this by using glu functions and by creating polygons using GL_QUADS or GL_TRIANGLES. We created a GeoHelper class, which we used to outsource tasks such as creating cubes, cylinders and triangular prisms. Afterwards, we could rotate, scale and translate our primitives appropriately in order to fit our models. We chose not to use GLUT functions due to their potential impact on frame-rate. Additionally, they do not provide additional features for texture mapping. Please refer to the Appendix for screenshots of our models.

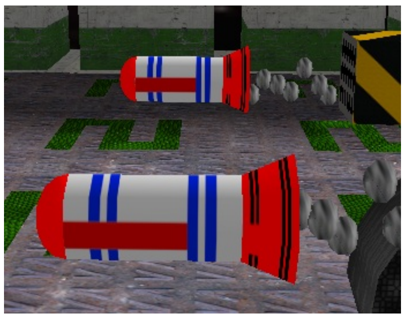
2.2 Animation

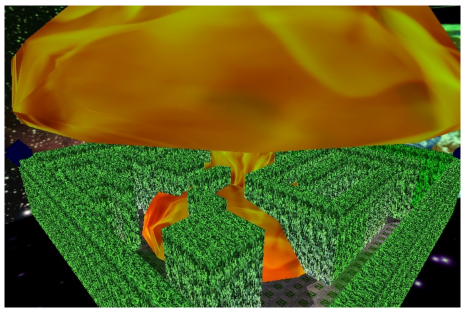
Most of our animation consists of basic translations (using glTranslate) to move objects. Additionally, we have a few other animated features.

First, we have created explosions for the robots and the end of the DeathMatch game. Both of these animations have a timer that scales a mushroom explosion after a certain period of time. Once the robot explosion animation is over, the mushroom fades away.

We have a hovering nuclear power-up, which rotates continuously in order to catch the players' eyes and missile smoke which uses rapid translations on grey spheres to imitate combustion.

Last, we have a flag that uses NURB's and rotations in order to imitate a flag waving in the wind.





2.3 Texturing

All of our textures are loaded using the 'imageloader' and TextureManager classes. Using glTexCoord2f, the textures coordinates are mapped when we define the primitive objects for our models. We choose our textures images with glBindTexture, in conjunction with a TextureManager function that calls a texture by filename. The TextureManager is also used to help toggle between different texture skins. Different skin collections are determined by their resource directories, which the TextureManager accesses.

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 primary classes that drive the high-level functionality of our game. Such classes relate to cameras, games, players, player input, lights, robots and more.

First, we have a general Camera class that is an abstract class used for camera orientation and movement. Every other camera class extends Camera and modifies the movement functions as needed. We have a circular camera that has a bird's eye view of the scene, which rotates about a fixed point on the map using a fixed radius. Additionally, it has a spotlight that can be toggled on and off. Our commander camera is a strategy game type of camera. It can zoom in and out, strafe and rotate about a look-at point. Our follow camera follows our control unit from a fixed distance. We also have light cameras that are located on top of each spotlight. Finally, we have a free look camera where the camera is controlled using 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. Game is the class that calls the render function of the level renderer and the players. It also acts as an intermediary that connects the PlayerInput class to our key press and key up states.

The Player, HumanPlayer and PlayerInput classes consist of everything that the player has ownership over. The player classes control each player's camera view, robot and control unit. The PlayerInput class parses and executes all the player commands via mouse and key states.

The BulletManager handles bullet movement and collisions.

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 needs to redefine in its own class. Since each element inherits 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 TextureManager 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 shiny metallic look to the robots. 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.

For each static model, robot, bullet and control unit, we created a bounding box used for collision detection. Our CollisionTester class was used to handle object collisions. In most cases, it was used to prevent an object from walking into another object. In other cases, such as for bullet, a collision would damage a robot's life.

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 sound libraries 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 consists of:

- 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 static model inside a display list when building the maps. This was very helpful considering the continuous re-rendering of the game. By activating the display lists, we ensured that certain vertex and the pixel data were evaluated only once and not repeatedly. Another solution we had was to toggle of certain superfluous features such as our team number on every grass title.

4.2 Shadows

One of our problems with the shadows was making them work with the display lists. Since we are using a lot of display lists, it was hard to actually figure out where and how to implement the shadows. With the help of the stencil buffer, we were able to fix this problem. We also had trouble with the dynamic shadows, because they weren't updating correctly. Now we have really good looking shadows and we even added an effect that makes the shadows disappear the further away you are from the lights.

4.3 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 problems with the directory paths since Windows and Mac read them differently. We had to implement a helper that checks which platform the project is running on and change the directory file accordingly.

4.4 Other

Other challenges included environment mapping and sound. Due to time constraints and its own complexities, environment mapping proved to be a significant challenge. Additionally, it decreases our frame-rate drastically, which is why we enabled a toggle feature for it.

Implementing sound was also time-consuming. This was because Apple deprecated a significant amount of useful functions, which were taught in most online sound tutorials. Yet we wanted to keep our program cross-platform. In the end we found alternative methods of implementing sound.

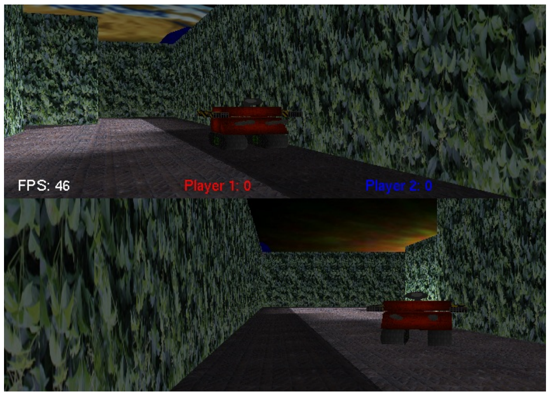
5. Bonus Features

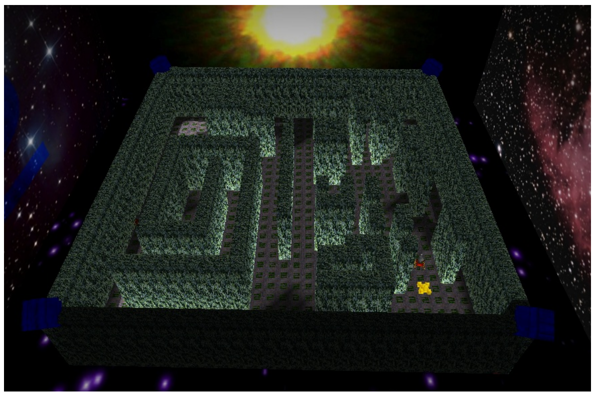
5.1 Music

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 and two players can play the game simultaneously. Each of the players has to destroy the robots of the other player. Additionally, you can also activate split-screen view by pressing F8 in the classic Nether Earth mode.





Screenshots of Deatchmatch mode and map

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

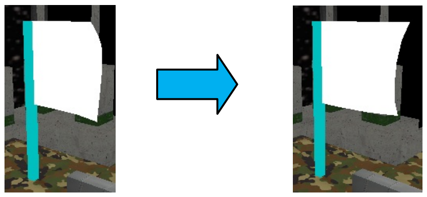
The project can be run perfectly on both Windows and MacOS. This should also work on Linux.

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.

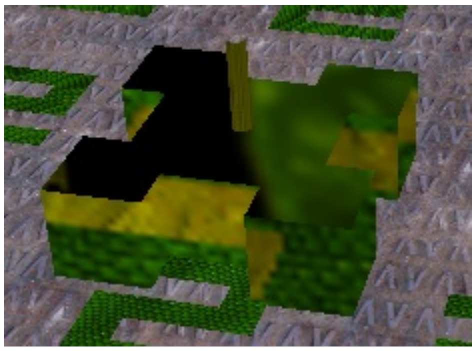
5.6 NURB's

NURB's was used to create the flags. This gives the flag a nice wind effect.



5.7 Reflection

We also used Environment Mapping to simulate the results of ray-tracing. With Environment Mapping, we get a nice reflection effect on the UFO. This can be activated by clicking “k”. In the picture below, notice the reflection of Earth and the player number.



Screenshot of environment mapping

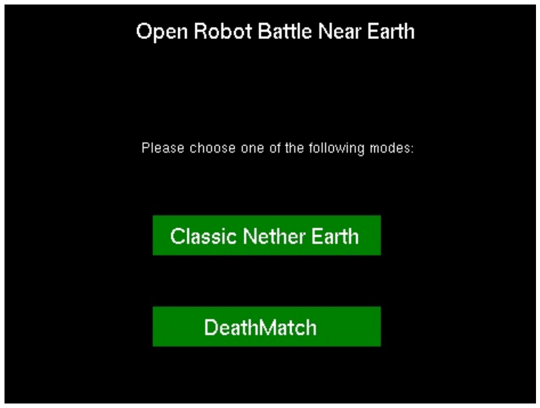
6. User Manual

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:


Open Robot Battle Near Earth

Please choose one of the following modes:

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

F5 -- Side-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
l	Toggle the light of the bird view camera and robot

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
F6	Toggle the different kind of skins
F8	Activate DeathMatch
F10	Toggle team number
a/s/d/w	Move the UFO
Space bar	Levitate UFO

z -- Toggle the skyboxes

Alt-Enter---------------------------------- Full Screen

h -- Help Menu

r --- Set robot destination with UFO

f --- Mate UFO with robot (only when sitting on top of the UFO)

Toggle robot components:

i --- Cycle components

j --- Turn component off

u -- Turn component on

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

6.2.3 Joystick Control

Arrow keys------------------------------- Move the robot

Bumper buttons ----------------------- Move the robot camera left and right

A -- 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/
Brick, Concrete_bare, Dirt, Ink, Marble, Metal1, Metal2, Metal 3, Metal4, Bipodf: 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
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

7.2 Code

The image loader class file was taken from: 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/

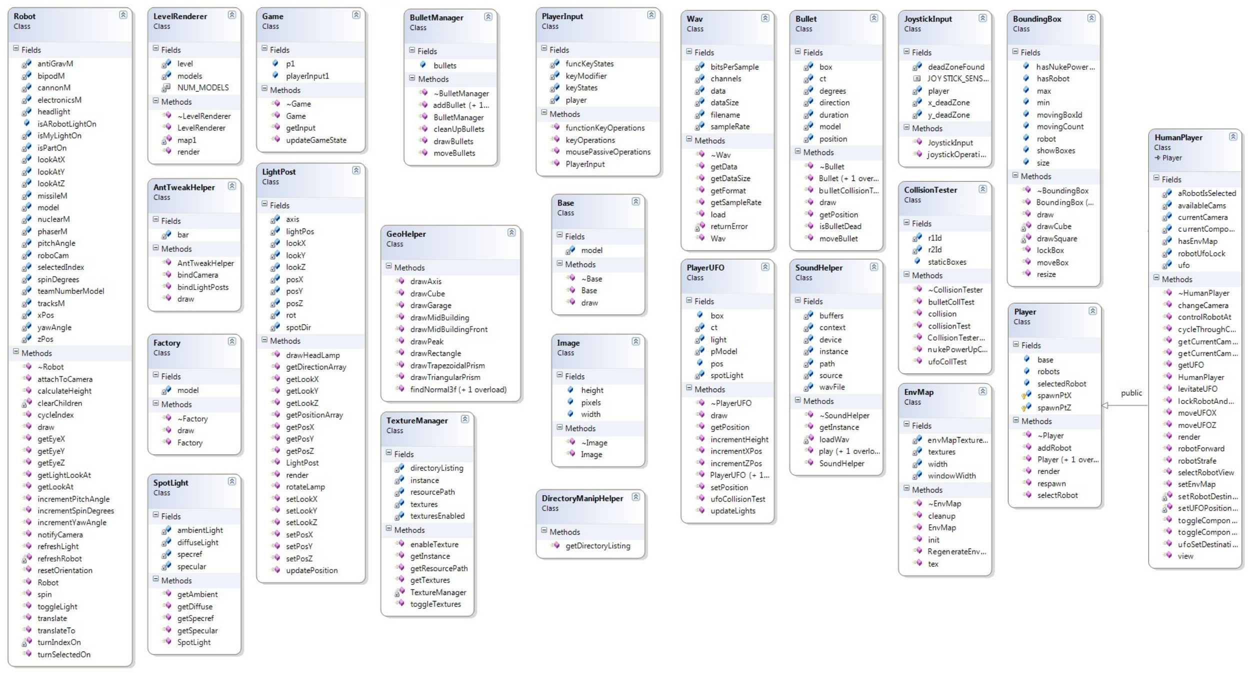
The shadow matrix code is from: http://www.opengl.org/archives/resources/features/StencilTalk/sld021.htm

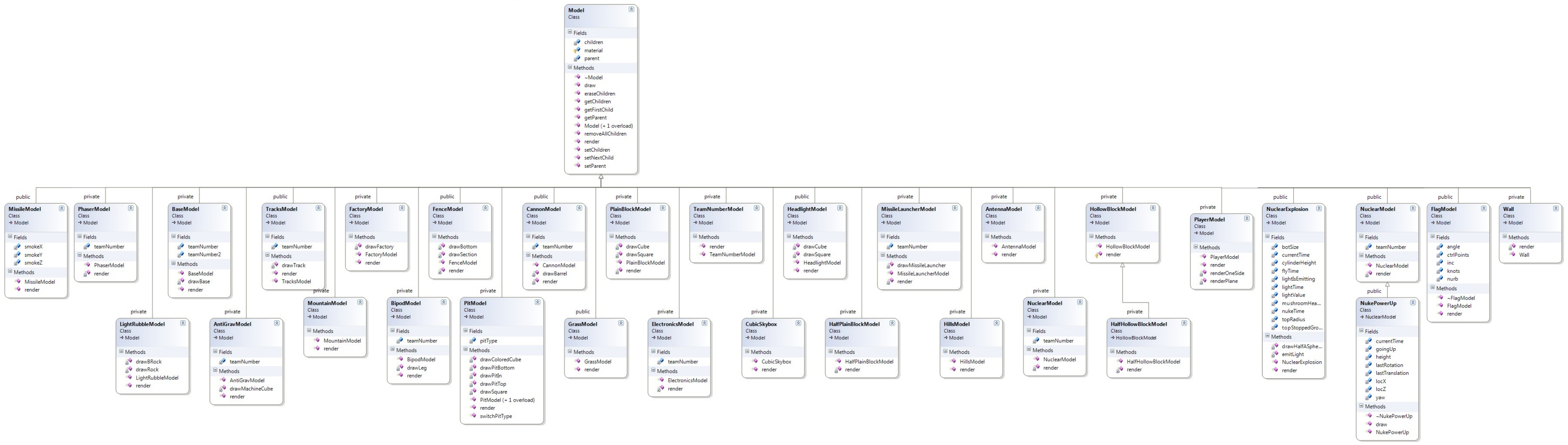
AntTweakBar for the debugging window: http://www.antisphere.com/Wiki/tools:anttweakbar

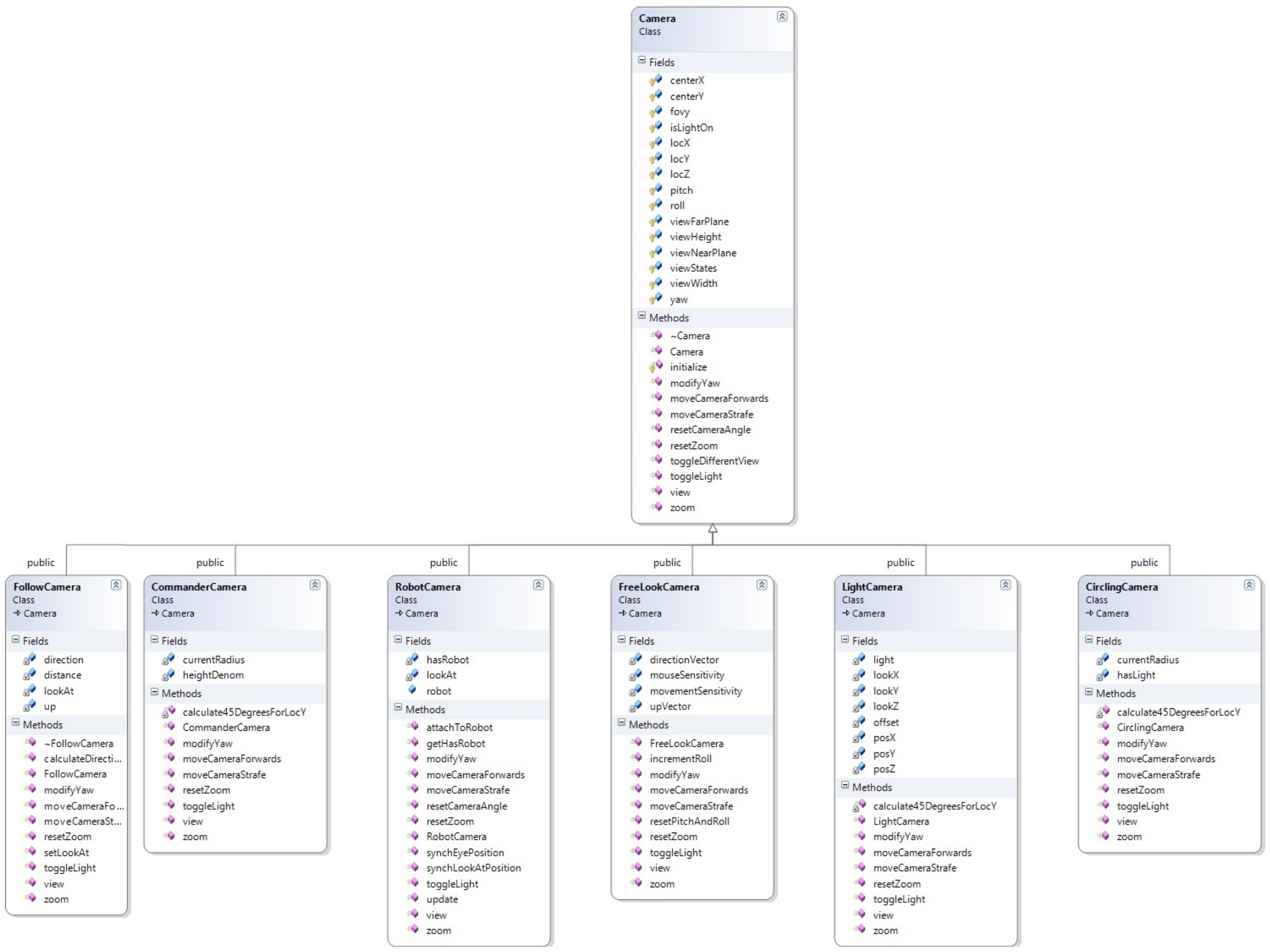
Dirent API for windows : http://www.softagalleria.net/dirent.php

Appendix

Exhibit 1 - Class Diagrams




Model Class
Fields children material parent
Methods ~Model draw eraseChildren getChildren getFirstChild getParent Model (+ 1 overload) removeAllChildren render setChildren setNextChild setParent
public MissileModel Class → Model Fields smokeX smokeY smokeZ Methods MissileModel render
private PhaserModel Class → Model Fields teamNumber Methods PhaserModel render
private BaseModel Class → Model Fields teamNumber teamNumber2 Methods BaseModel drawBase render
public TracksModel Class → Model Fields teamNumber Methods drawTrack render TracksModel
private FactoryModel Class → Model Methods drawFactory FactoryModel render
public FenceModel Class → Model Methods drawBottom drawSection FenceModel render
public CannonModel Class → Model Fields teamNumber Methods CannonModel drawBarrel render
private PlainBlockModel Class → Model Methods drawCube drawSquare PlainBlockModel render
private TeamNumberModel Class → Model Methods render TeamNumberModel
public HeadlightModel Class → Model Methods drawCube drawSquare HeadlightModel render
private MissileLauncherModel Class → Model Fields teamNumber Methods drawMissileLauncher MissileLauncherModel render
private AntennaModel Class → Model Methods AntennaModel render
private HollowBlockModel Class → Model Methods HollowBlockModel render
private PlayerModel Class → Model Methods PlayerModel render renderOneSide renderPlane
public NuclearExplosion Class → Model Fields botSize currentTime cylinderHeight flyTime lightIsEmitting lightTime lightValue mushroomHea... nukeTime topRadius topStoppedGro... Methods drawHalfASphe... emitLight NuclearExplosion render
public NuclearModel Class → Model Fields teamNumber Methods NuclearModel render
public FlagModel Class → Model Fields angle ctrlPoints inc knots nurb Methods ~FlagModel FlagModel render
private Wall Class → Model Methods render Wall
private LightRubbleModel Class → Model Methods drawBRock drawRock LightRubbleModel render
private AntiGravModel Class → Model Fields teamNumber Methods AntiGravModel drawMachineCube render
private MountainModel Class → Model Methods MountainModel render
private BipodModel Class → Model Fields teamNumber Methods BipodModel drawLeg render
private PitModel Class → Model Fields pitType Methods drawColoredCube drawPitBottom drawPitIn drawPitTop drawSquare PitModel (+ 1 overload) render switchPitType
public GrassModel Class → Model Methods GrassModel render
private ElectronicsModel Class → Model Fields teamNumber Methods ElectronicsModel render
private CubicSkybox Class → Model Methods CubicSkybox render
private HalfPlainBlockModel Class → Model Methods HalfPlainBlockModel render
private HillsModel Class → Model Methods HillsModel render
private NuclearModel Class → Model Fields teamNumber Methods NuclearModel render
private HalfHollowBlockModel Class → HollowBlockModel Methods HalfHollowBlockModel render
public NukePowerUp Class → NuclearModel Fields currentTime goingUp height lastRotation lastTranslation locX locZ locZ yaw Methods ~NukePowerUp draw NukePowerUp


Camera
Class
Fields
centerX
centerY
fovy
isLightOn
locX
locY
locZ
pitch
roll
viewFarPlane
viewHeight
viewNearPlane
viewStates
viewWidth
yaw
Methods
~Camera
Camera
initialize
modifyYaw
moveCameraForwards
moveCameraStrafe
resetCameraAngle
resetZoom
toggleDifferentView
toggleLight
view
zoom
public
FollowCamera
Class
→ Camera
Fields
direction
distance
lookAt
up
Methods
~FollowCamera
calculateDirecti...
FollowCamera
modifyYaw
moveCameraFo...
moveCameraSt...
resetZoom
setLookAt
toggleLight
view
zoom
public
CommanderCamera
Class
→ Camera
Fields
currentRadius
heightDenom
Methods
calculate45DegreesForLocY
CommanderCamera
modifyYaw
moveCameraForwards
moveCameraStrafe
resetZoom
toggleLight
view
zoom
public
RobotCamera
Class
→ Camera
Fields
hasRobot
lookAt
robot
Methods
attachToRobot
getHasRobot
modifyYaw
moveCameraForwards
moveCameraStrafe
resetCameraAngle
resetZoom
RobotCamera
synchEyePosition
synchLookAtPosition
toggleLight
update
view
zoom
public
FreeLookCamera
Class
→ Camera
Fields
directionVector
mouseSensitivity
movementSensitivity
upVector
Methods
FreeLookCamera
incrementRoll
modifyYaw
moveCameraForwards
moveCameraStrafe
resetPitchAndRoll
resetZoom
toggleLight
view
zoom
public
LightCamera
Class
→ Camera
Fields
light
lookX
lookY
lookZ
offset
posX
posY
posZ
Methods
calculate45DegreesForLocY
LightCamera
modifyYaw
moveCameraForwards
moveCameraStrafe
resetZoom
toggleLight
view
zoom
public
CirclingCamera
Class
→ Camera
Fields
currentRadius
hasLight
Methods
calculate45DegreesForLocY
CirclingCamera
modifyYaw
moveCameraForwards
moveCameraStrafe
resetZoom
toggleLight
view
zoom

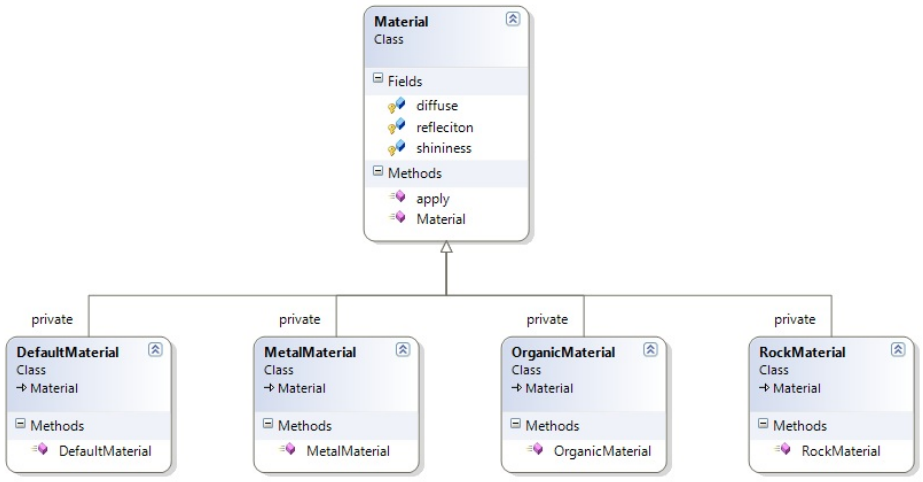
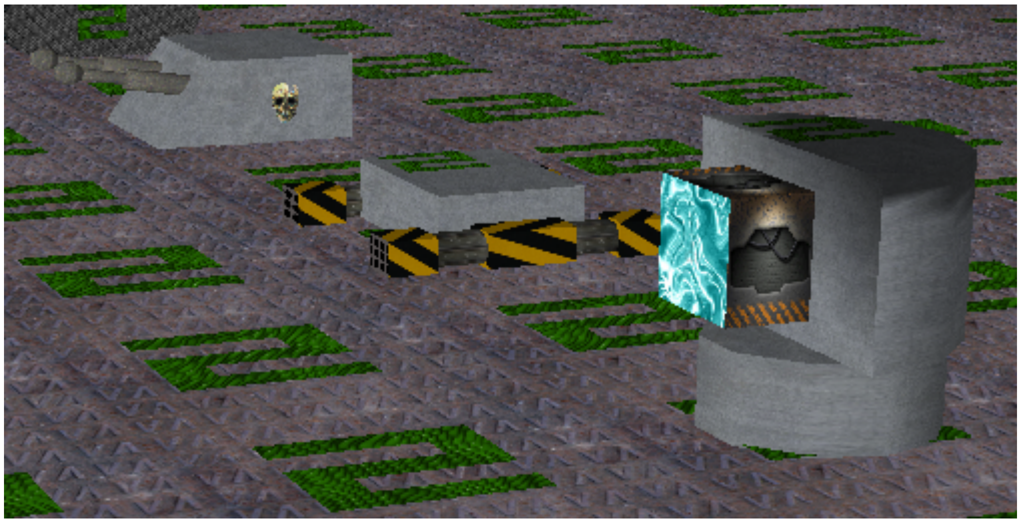
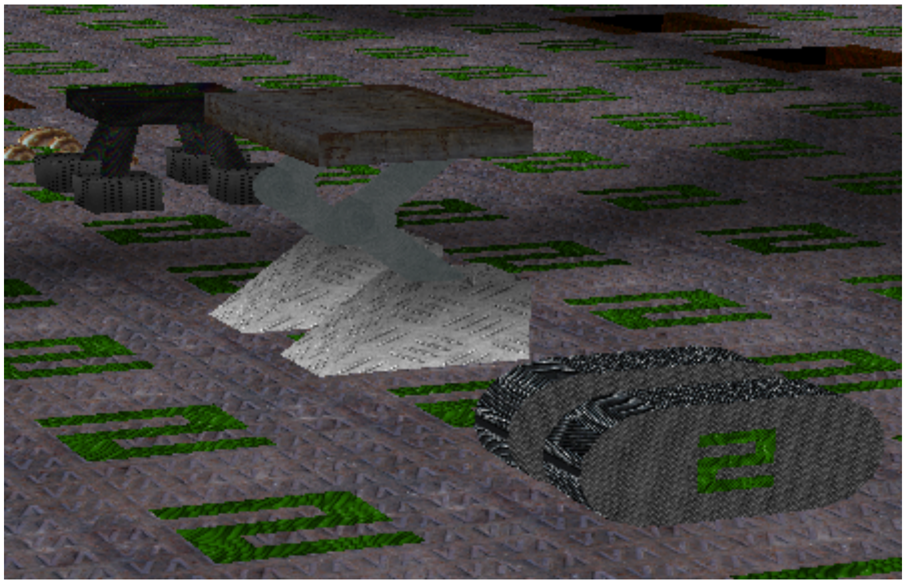
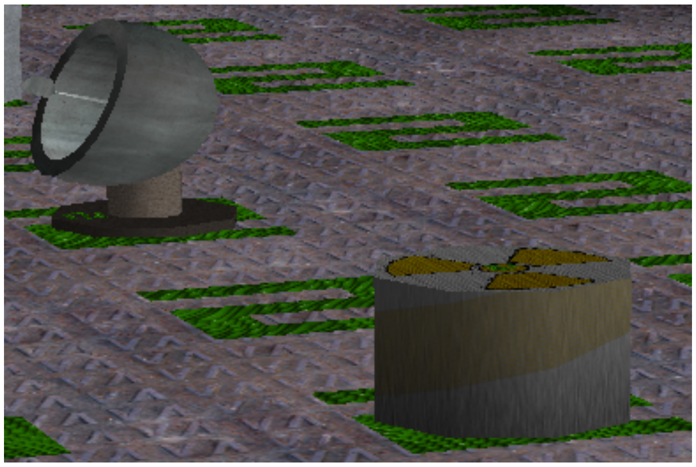

Material
Class
Fields
diffuse
refleciton
shininess
Methods
apply
Material
private
DefaultMaterial
Class
→ Material
Methods
DefaultMaterial
private
MetalMaterial
Class
→ Material
Methods
MetalMaterial
private
OrganicMaterial
Class
→ Material
Methods
OrganicMaterial
private
RockMaterial
Class
→ Material
Methods
RockMaterial

Exhibit 2 - Model Screenshots







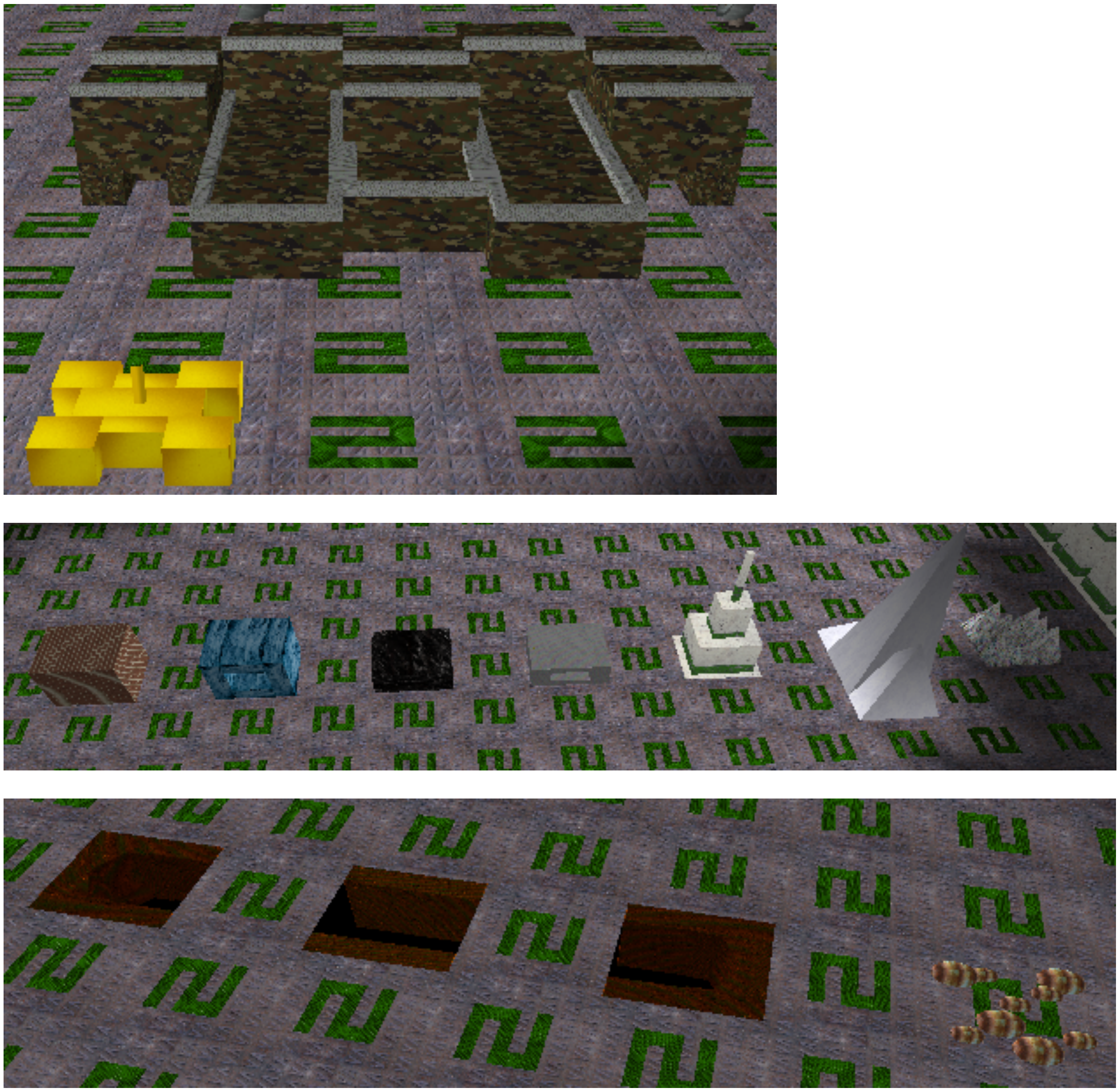


Exhibit 3 – Bounding Box Screenshot

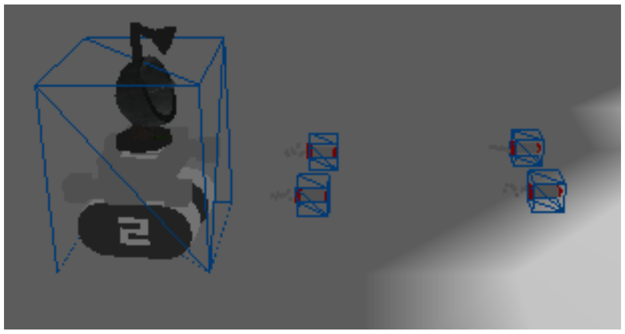


Exhibit 4 – Bounding Box Screenshot