An “Open Robot Battle Near Earth” remake of Nether Earth" by Team 4: Brief Specification, Design, and User Manual, COMP371 Winter 2012

Jordan Victor: jordanvictor@msn.com

Taras Kamtchatnikov: eggmik@hotmail.com

Gianni Trotta: giannitrotta@gmail.com

Sebastien Sicard: sebastien.jh.sicard@gmail.com

Table of Contents:

1. Project Description

The goal of this project is to create a game based on Nether Earth using the OpenGL software for graphics. The project is coded using C++.

2. Design Methodology

2.1 Modeling

Models are made mostly using GL_QUADS and spheres. Objects that are using textures are made of several QUADS for each surface, that together form the whole object. Otherwise, pre-rendered objects (like glutSolidSphere) are used. Models are drawn to fit inside a 1x1 square, which is the size of one floor tile, to ensure proper collision interaction between all objects. Construction-style components like building blocks and robot parts are either one unit or half a unit tall to allow for the mixing and matching of parts and different building configurations.

2.2 Animation

2.2.1 Camera:

Camera is handled primarily through various manipulations of the gluLookAt() function. The main camera is designed to move around the field and rotate at its current position through the passing of various variables into the function. A similar technique is used for the control unit camera, which follows the control unit around. Additionally, there are several static camera modes that offer different views of the map.

The orbiting camera mode calculates points on a circle that represents the camera's orbit path. It then feeds the coordinates of that point into the gluLookAt() function to update the camera's position. This function is based on a sample mouse-controlled orbit function by H. Shirokawa, a link to which can be found in the references section. Instead of mouse coordinates, the orbit camera uses keys to orbit to the left or to the right. Additional calculations, such as the up-vector coordinates, which are not used by the simplified orbiting camera, are omitted from this version of the function.

Lastly, there is an orthogonal projection camera, which projects the map onto the screen and allows rotating for various viewing angles. The basis for this camera mode can be found in the sampleprogram.cpp file from class.

2.2.2 Object Movement:

The positions of all moving objects are stored as points, a structure consisting of an X & Z value, and those points are used to place the moving objects. The robot walking in circles is moved using a timer loop that has countdowns to limit how long it spends moving in one direction. The control unit is controlled by switching to its camera and using the WASD keys to change the position values. The other robot is controlled by hitting Enter to dock, but only if the control unit is directly above it. Then the robot is controlled the same way as the control unit but moves half the speed. The energy ball is animated the same way as the robot moving in circles but instead of circling it disappears after a certain distance moved. The robot moving in circles, the controllable robot and the energy ball all have collision detection in relation to each other, so if the robots are blocking one or the other they can’t move, only turn in the direction they want to move. The energy ball merely disappears upon hitting the circling robot.

2.3 Texturing

The texturing functions make use of the ImageLoader class that can be found on videotutorialsrock.com. Every surface of an object is textured separately, except for the bottom side, which is not visible to the user under normal circumstances. Textures for the building block components (wood, ice, mud, stone, gold and hole) were made using ArcSoft Photo Studio 2000. The rest have been taken from various sources, which are listed in the references section.

2.4 Difficulties

Lighting and shadows were the most problematic parts of the project. The lighting behaves bizarrely and is cut off at random points on the map. Earlier on in the project, there was some difficulty with the depth buffer and getting the models to align properly. Time management and loss of team member at the end of the project.

2.5 Extras

- The orthogonal camera mode, as described above, is an extra feature
- Sounds have been added for dock/undock, fire and destroy
- Project has been open sourced under BSD 3-clause license, located at: https://github.com/portaljacker/comp371_nether_earth

3. Program Structure

The project was made to be as object-oriented as possible. Each component is its own class for easy modularity.

Directory Tree of Project:

```
comp371_nether_earth
|  .gitignore
|  tree.txt
|  comp371_nether_earth.sln
|  README.md
|
----comp371_nether_earth
           Bipod.cpp
                 Bipod.h
                 Block.cpp
                 Block.h
                 camo1.bmp
                 camo2.bmp
                 Cannon.cpp
                 Cannon.h
                 cgold.bmp
                 comp371_nether_earth.vcxproj
                 comp371_nether_earth.vcxproj.filters
                 comp371_nether_earth.vcxproj.user
```

Controller.cpp

Controller.h

Cube.cpp

Cube.h

debris.bmp

dock.wav

eBrain.cpp

eBrain.h

Enumerations.h

explode.wav

gold.bmp

Grav.cpp

Grav.h

hole2.bmp

ice.bmp

ImageLoader.cpp

ImageLoader.h

lightPostConstruct.cpp

lightPostConstruct.h

Map.cpp

Map.h

metal.bmp

Missiles.cpp

Missiles.h

mud.bmp

Nuke.cpp

Nuke.h

pew.wav

Phaser.cpp

Phaser.h

Rubble.cpp

Rubble.h

SceneDriver.cpp

skeleton.cpp

steel.bmp

stone.bmp

Tile.cpp

Tile.h

Tracks.cpp

Tracks.h

trust.bmp

undock.wav

wood.bmp

4. Compilation Instructions

To compile the program, open the comp371_nether_earth.sln file in Microsoft Visual Studio. Once it loads, click on the run button to begin compiling. The program will start automatically when compilation is finished.

5. User Manual

Camera controls:

Perspective mode:

w, a, s, d: Move the camera around.

q, e: Move the camera up or down.(Closer or farther from the "ground".)

left & right arrow keys: Rotate camera horizontally.

up & down arrow keys: Rotate camera vertically.(Look up or down.)

Orbit mode:

a, d: Make camera orbit left or right

Orthogonal mode:

left & right arrow keys: Rotate around y axis.

up & down arrow keys: Rotate around x axis.

Controller mode:

w,a,s,d: move the control unit

Enter: dock/undock

Space: shoot if docked

Controls common to all modes:

=,-2: Zoom in and out.

F1, F2: Rotate around Z axis.(May get clippy in Perspective mode.)

c: Toggle Perspective / Orbit / Orthogonal / First Person View modes.

r: Reset camera.

1-8: switch to camera modes directly.

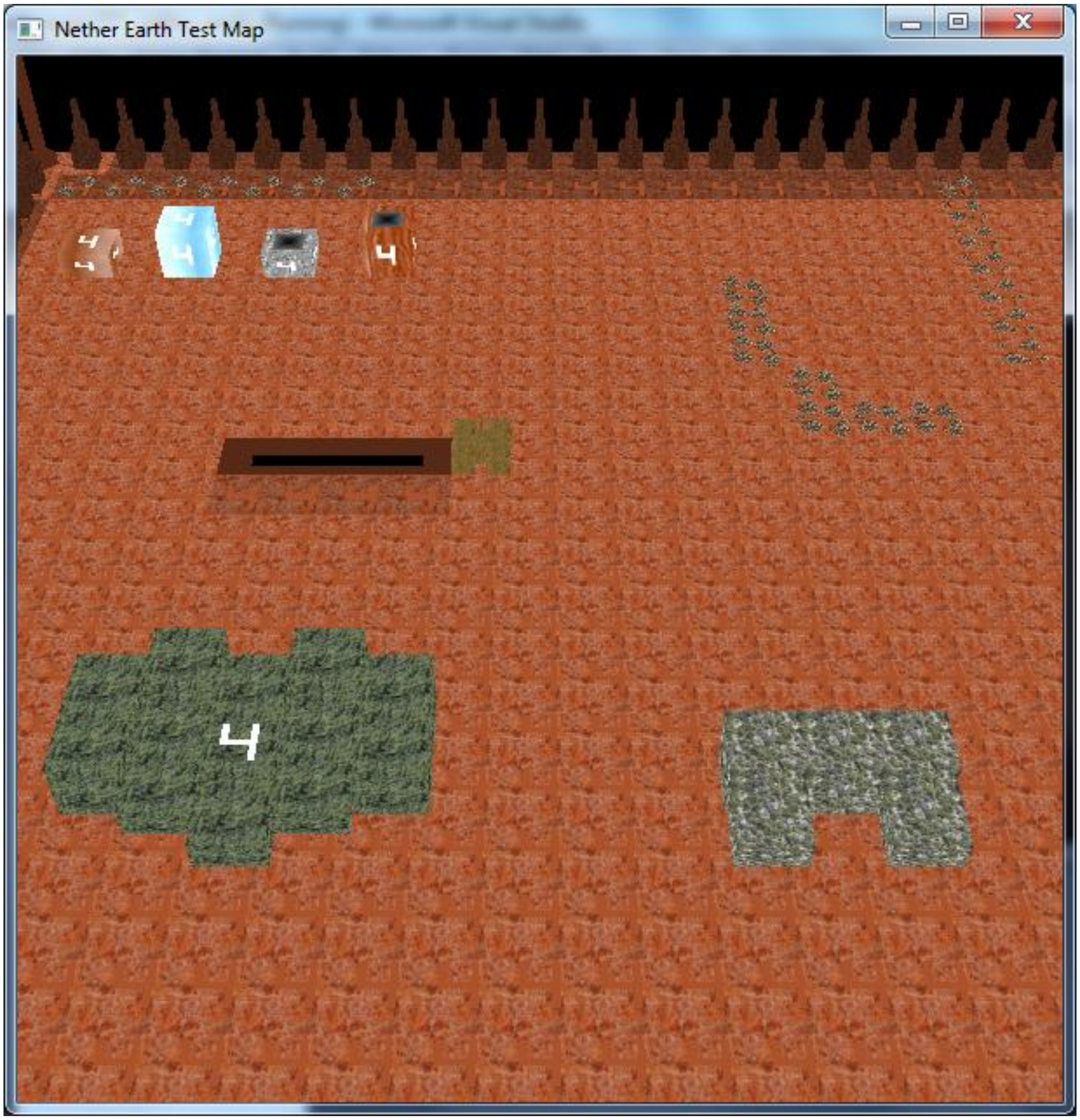
Other controls:

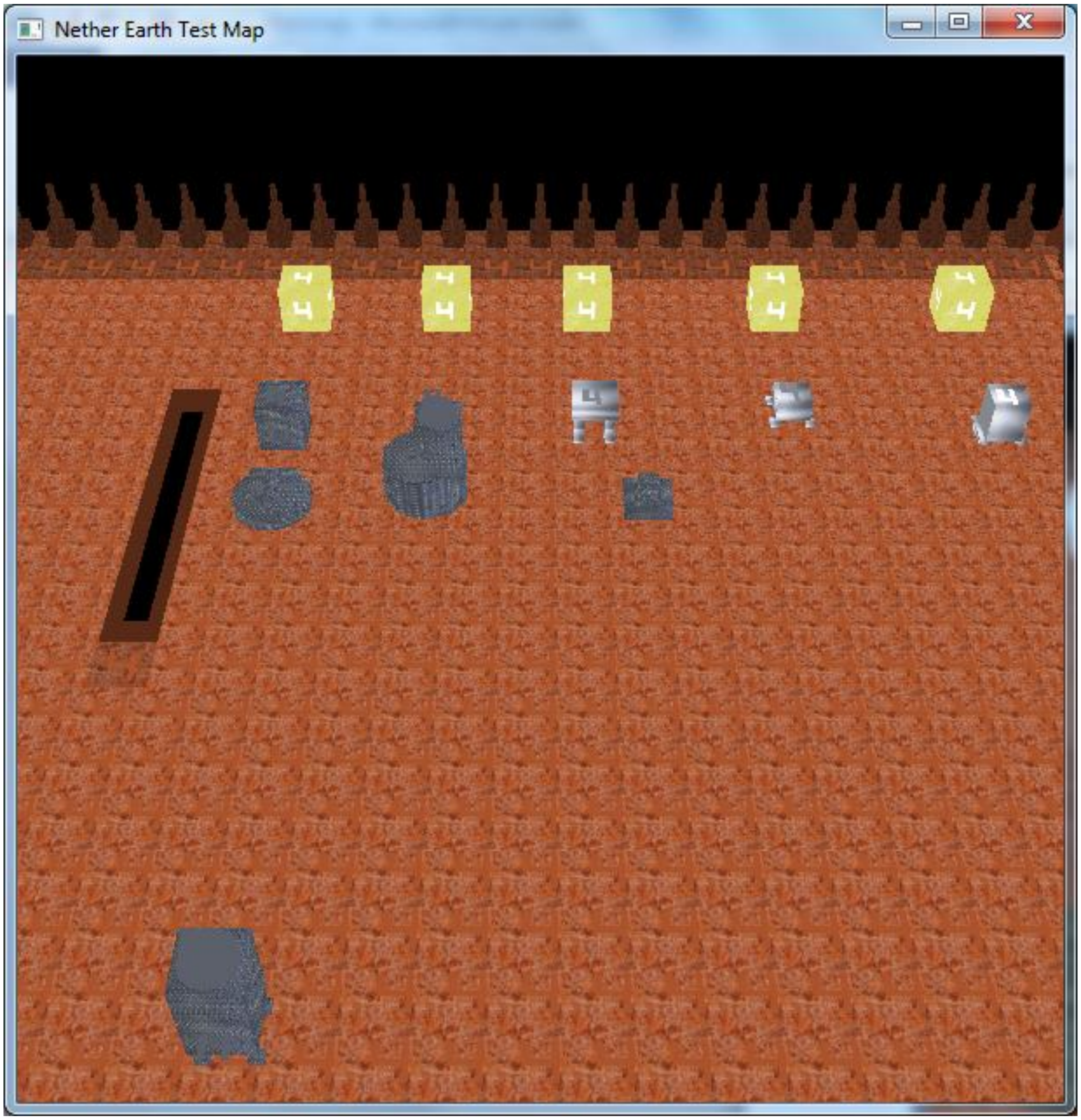
t: Toggle wireframe/shading modes.

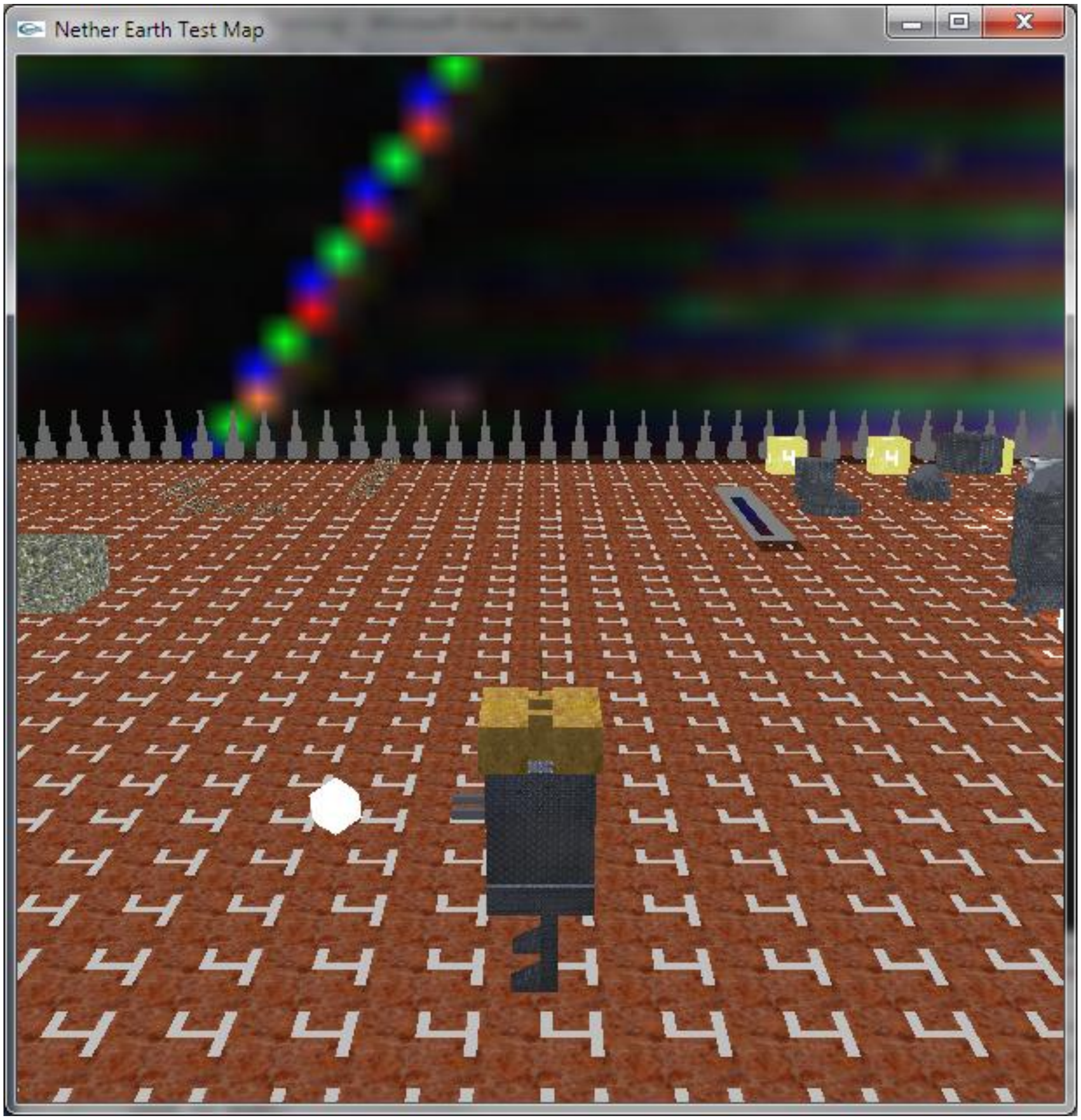
h/H: Display list of controls in console window.

esc: Close the program.

6. Images




Nether Earth Test Map


Nether Earth Test Map

7. References

7.1 Texture resources:

http://www.cgtextures.com used for:

camo1.bmp

camo2.bmp

cgold.bmp

debris.bmp

trust.bmp

http://www.jbdesign.it/idesignpro/images/metal/steel%20steel_sheetmetal_rpost.jpg for:

steel.bmp

http://designm.ag/images/0409/metal/41.jpg f or:

metal.bmp

7.2 Source Code:

http://homepages.ius.edu/rwisman/b481/html/notes/FlyAround.htm Basis for orbit() finction.

sampleprogram.cpp and skeleton.cpp   Basis for various functions in scenedriver.cpp.

7.3 Sounds:

dock.wav, explode.wav, pew.wav & undock.wav were all created using the tool found at http://www.bfxr.net/