W4160 Final Project

Space War Shooting Game

Guardians of the Earth

Members:
Yuan Feng (UNI: yf2338)
Mengyu Wu (UNI: mw2907)
Yiyang Qiu (UNI: yq2183)

Goal:

For this final project, we designed and implemented a video game, in which the main goal is to defend the attacks from aliens to the earth. The game contains fundamental interactions and also provides with different player views. We emphasis on some lighting effects and physical motion effects implementation, and certainly the modeling of all the objects in the scene. Below we will in detail introduce about the usage of our game, and the implementation details of our programs. User can have an excited experience in the process of playing our game, as well practice their reactions. The current stage is a simple prototype of the game. It can be expanded in the future so as to make it very complicated and product-like game.

Usage:

Directly run src/Main.java to start the game.

-Controls:

1. SHIFT to switch between the third viewpoint and the first viewpoint, in the third-view mode, player can see the astronaut defender.
2. LEFT/ RIGHT/ UP/ DOWN to control the movement of the player, to adjust the defender's position so as to fight with the coming enemies.
3. SPACE to fire.
4. ESC to escape the game.

-Game Settings:

1. There are in total 2 levels in the whole game, each level there will be 10 enemies attacking the earth.
2. In the first level, there is only one kind of enemy ship attacking towards the earth, each of which is with even speed.
3. In the second level, there is two different kinds of enemy ships attacking towards the earth, with one of them accelerating, and another one with even speed.
4. There are some other planets in the space, acting as helper to the player, i.e. once the randomly coming shape conflict with the planet, the ship will explode. But the planet itself will also be hurt and decreased of its size until explode when it is too small.
5. In the second level, there would be space station near the earth, which also acting like a protector.

What We Implement:

1. **3D space scene, with 2D texture of the randomly distributed star wall.**
   We implement the camera to look at a value large depth value, and also put the earth slightly in front of the viewer, so as to make the attackers come from a really far vanishpoint to make the whole scene seem to be infinite. The background stars in the space are drawn by randomly calculating the 2D coordinates each time user starts the game. So each time user would see a differently distributed star wall in the space. Each time there is an explosion in the game scene, the stars would be lighten, which is like a realistic effect when there is a space explosion.

2. **Object oriented model parsing and importing to the scene, using individual classes inherited from an abstract model class.**
   We maintain an abstractModel class to be the parent class of all the models which need to be initialized in the scene. They have some common methods for the position, rotation, model parsing and texture parsing. Each object in the scene has their own features, which are implemented by overwriting the parent methods or creating new. All the objects, i.e. the initialized models in the scene would be maintained using lists. Each frame is based on redrawing those existing objects in the lists.

3. **Texture mapping for the imported models.**
   All the textures mapped to the models are PNG or JPG files, which is parsed out from the "vt" data in OBJ file, so that each pixel on the texture picture is aligned onto the 3D model. All the parsers including the OBJ and texture ones are implemented in Face.java, Model.java, and OBJLoader.java.

4. **2D text displaying game status and player status.**
   We used UnicodeFont from Slick library, which can directly load the 2D texts to the 3D scene, and can be drawn or updated in each frame, that is how the user can see real-time status of their lives and the game levels.

5. **Lighting shaders.**
   To make the view more realistic, we also implement Texture smooth shader in this project like Figure 1. Each triangle is shaded using a color value from Blinn phong shader and multiplying the resulting color value by the current texture. The texture coordinate (given by gl TexCoord[0].st in GLSL) for each vertex are used to index into the texture.

Figure 1: Shader Effects

6. **Physical collision detection of different object to enable the explosions.**

Because this is a shooting game, we implement a naive collision detection algorithm to enable all the fire and fight effects. The algorithm can be summarized as: for specific different 2 classes of objects, calculate their distance between each one's center, and compare the distance with the scale of the larger object. If the distance is smaller than the scale, there should be a collision for these two objects. For different pairs of collision, we assign different explosion effects, so as to avoid confusion for users.

7. **Physical acceleration of the bullets and ships to make the motion effect realistic.**
   For the acceleration effect of bullets, we simply detect the depth, i.e. the z-coordinate of the moving earth, the bullet just appears has a higher speed, while with longer shooting distance, its speed is decreased.
   After shooting down all the enemy spaceships in level 1, there would be more upgraded spaceships, which accelerate when they move close to the earth. Here we implement acceleration for new spaceships. In level 1, all the spaceships moves with constant velocity, the distance formula:

   $$Snext=Sprev+V*dietaT.$$

   In level 2, parts of spaceships move with acceleration, the distance formula:

   $$Snext=Sprev+(time*V)*dieatT,\ time=time+1\ \text{(time increases by one each frame)}$$

8. **Bullet implementation.**
   We implement the bullet by simply representing each one of them using a 2D square and textured with uniform red color.

Result:

We made a complete video game, as shown in the video EarthGardiance.mp4.

Work Division:

-Yuan Feng (UNI: yf2338)

Implements the code framework of the game; the model & texture parsing and loading; the Object Oriented data structure to maintain the frame update, newal and removal of objects; the Collision Detection algorithm for the shooting logics.

-Mengyu Feng (UNI: mw2907)

Implements blinn phong texture shaders; Adding point lights, Implement Lander and UFO classes; Implements acceleration moving of spaceships for level 2. the model & texture parsing and loading for level 2.

-Yiyang Qiu (UNI:yq2183)

Helped implement debug and test shaders. Load new spaceship model and add textures in Level 2. Make Videos and documents.