Computer Graphics Final Project 2018

January 2019

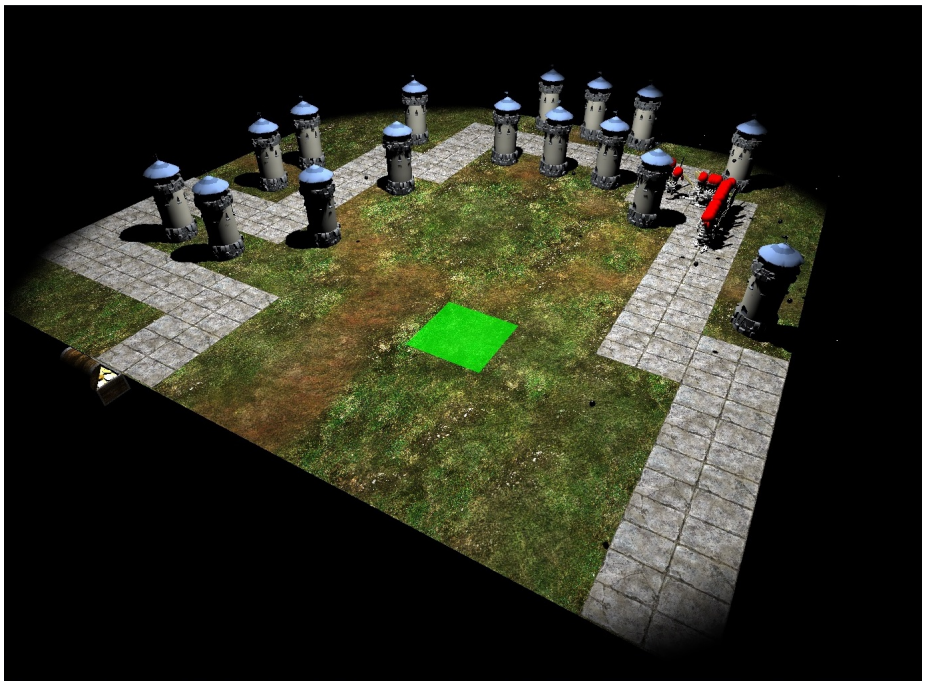

Figure 1: Tower Defence Game

1 Introduction

The goal of the project is to create a tower defence game using OpenGL, based on the OpenGL lab source code. The objective of the game is to prevent enemies from stealing the treasure by placing defence towers along the path that leads to the treasure chests. As part of the project, you will develop the loading, placing and interaction of all the game assets in the scene. To accomplish this, you will need to utilize techniques that you learned through the entire course and

lab lectures. You can also optionally experiment with the gameplay mechanics, more advanced graphics techniques and use more assets to enrich the game.

2 Data

The following models are provided:

- Defence Tower (tower.obj/mtl)
- Pirate Body (pirate_body.obj/mtl)
- Pirate Left Foot (pirate_left_foot.obj/mtl)
- Pirate Right Foot (pirate_right_foot.obj/mtl)
- Pirate Arm (pirate_arm.obj/mtl)
- Treasure Chest (treasure_chest.obj/mtl)
- Terrain (terrain.obj/mtl)
- Road (road.obj/mtl)
- Green Plane (plane_green.obj/mtl)
- Red Plane (plane_red.obj/mtl)

You can also use more assets from the web or your own creation.

3 Gameplay

The goal is to prevent the enemies (Pirates) from reaching the treasure chests by placing defence towers along the path that leads to the treasure. Upon reaching the treasure chests, pirates will steal coins from them and in order for the player to win, the player should prevent the enemies from stealing all the coins until the completion of the level. If all the coins are gone before the completion of the level, then the player loses. The number of coin chests available at the end of the path should be indicative of the amount of coins left. The duration of the level can be measured by waves of enemies or time passed. New scores of enemies should be more resilient to your salvo and next levels may give the player upgrades to defend the path more effectively against increasingly tough enemies.

Pirates will spawn in waves from the start of the path, every **20** seconds, and try to reach the treasure. They will follow the designated route and upon reaching the treasure, will remove 10 coins from one chest. Chests are placed at the end of the path and will hold 100 coins each. When a chest becomes empty, it should be removed from the map. If every chest is removed, then the player is defeated.

To defend the chests, the player can place defence towers along the path, that will shoot and eliminate pirates. A new defence tower can be placed every **2** minutes. Defence towers can also be rearranged. Every **30** seconds, one defence tower can be selected, removed and placed again in a new tile, in order to better cover the path and destroy the pirates.

You can use your creativity and alter the gameplay mechanics, freely.

4 Scene Description

The terrain is provided in the object (*terrain.obj*). The original dimensions of the terrain are in the interval [-1 1]. This should be scaled to [-20 20]. The terrain should be divided into a regular grid of 10x10 tiles, each tile measuring 2x2 units. In each tile, road geometry can be placed based on the configuration of the scene. Each road geometry piece should be scaled to fit the tile dimensions. The original size of each road piece is [-1 1] and the model can be loaded from (*road.obj*). An indicative configuration of the position for each road-covered tile is provided below:

i	j
0	0
0	1
0	2
0	3
1	3
1	4
1	5
1	6
1	7
2	7
2	8
3	8
4	8
5	8
6	8
6	7
6	6
7	6
7	5
7	4
7	3
8	3
9	3
9	2
9	1
8	1
7	1
6	1
6	0

The order of the road tiles is very important and should be from the start of the path up to the end. The order will be used to navigate the pirates to the chests. **3** chests should be placed at the end of the path. Each chest will hold 100 coins. The treasure chest model is provided in (*treasure_chest.obj*) and should be scaled down with a scaling factor of 0.9. The original measurements of the treasure chest are provided below:

Min	(-12.2044, 0.0226, -15.6902)
Max	(11.8524, 21.6121, -0.4337)
Size	(24.0568, 21.5895, 15.2565)
Center	(-0.1760, 10.8174, -8.0619)

The camera should be placed at (0.720552, 18.1377, -11.3135) and the target will be (4.005, 12.634, -5.66336) with a FOV of 60 degrees.

A spot light should be used to illuminate the scene with position(16, 30, 16),

target(16.4, 0, 16) and hotspot/falloff zones of (73, 80). The color of the light source should be (140,140,140) but you are free to experiment with the intensity and color value of the light source. The light should use a shadow map with heavily blurred shadow edges (wide PCF kernel).

4.1 Pirates

Pirates are the enemies of the game. They want to go to the end of the path and steal the treasure. Pirates will spawn in waves from the starting point every **20** seconds. Pirates can spawn at random positions inside the wave or spawn based on a non-uniform distribution function. The Pirate model should be animated. Each pirate consists of the main body (*pirate_body.obj*), the arm with the sword (*pirate_arm.obj*), left foot (*pirate_left_foot.obj*) and right foot (*pirate_right_foot.obj*). Pirate should swing his sword and move his feet as he moves. Pirate arm origin (0,0,0) is at the center of the hand (see Fig. 2). It should pivot from the shoulder (4.5, 12, 0) of the pirate at a distance of 3 units and swing back and forth as the pirate moves. The left foot should be placed at (-4, 0, -2) and the right foot should be placed at (4, 0, -2) and they should both move back and forth as the pirate moves. The whole pirate model should be scaled down by factor of 0.09.

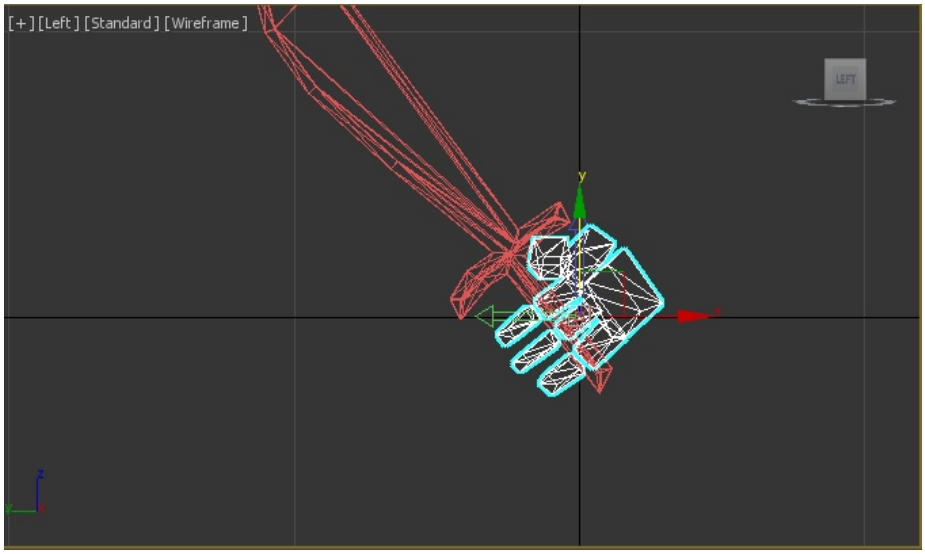

Figure 2: Coordinate system origin of pirate arm

Each Pirate will spawn at the start of the path. The path that the pirates will follow is given in the scene description. Pirates will follow the path in the order that the road tiles have been declared. The speed of the pirate can be **1** tile per second.

To implement the movement, you need to keep for each pirate the how many seconds have passed from the time that he spawned and the *current tile* index, which is the tile that the pirate is standing on. To compute the position of each pirate, we need to interpolate between the position of the *current* and *next* tile, based on the time and speed of the pirate. You can use the function *glm::mix* to interpolate between two variables. Additionally, each pirate should be rotated in order to always look forward. To compute the angle that the pirate should be rotated in order to always face forward along the path, we need the direction from the current to the next tile and compute the rotation angle using the function *atan2(direction.y, -direction.x)*. Then, we add to the rotation angle 90 degrees (in radians), to additionally rotate the model and align it with the face-forward axis and use the computed rotation angle to rotate the whole model.

4.2 Towers

Towers are allowed to shoot nearby enemies at specified intervals. The shooting range of each tower will be 2 tiles and they will shoot the closest enemy that enters their proximity.

Towers can be selected and moved every **30** seconds. You start with **3** Towers and every **2** minutes you get another one. You can freely modify the game mechanics and adjust the timers.

For the tower, the (tower.obj) model will be used. The dimensions and shooting position of the tower are provided below:

Min	(-2.6035, -0.0626, -2.6373)
Max	(2.5833, 12.7077, 2.4856)
Size	(5.1868, 12.7704, 5.1229)
Shooting position	(-0.0101, 9.5, -0.0758)

The tower should be scaled using a scale factor of **0.4** and placed at the appropriate tile. The projectile will be a cannonball, loaded from *cannonball.obj*, which is a black glossy sphere with radius 1.0. It should be scaled to 0.1 radius and spawn from the shooting position with direction towards the selected pirate and speed 9.0 units per second.

5 Collision Detection

Collision detection should be supported between the pirates, cannonballs and the treasure. Collision detection will be performed using sphere intersection. If the two centers of the spheres are at a distance smaller than the sum of both their radii, then the spheres are colliding. For each asset we will use the circumscribed sphere of the model's triangles, so the collision detection of two assets will be simplified to a simple collision detection between their spheres.

Each model center and radius is provided below:

Model	Center	Radius
Defence Tower	(-0.0101, 6.3225, -0.0758)	6.3852
Pirate	(-0.5957, 11.683, -4.274)	12.87075
Treasure Chest	(-0.1760, 10.8174, -8.0619)	12.0284
Cannonball	(0, 0, 0)	1.0

Each center and radius should be scaled based on the scale of each model.

6 Selection

The player can select the tile to perform an action by using the arrow keys. In the selected tile, the player can press the key "R" to remove the selected tower or the key "T" to add a tower. To provide visual feedback about whether an action can be performed in the selected tile or not, you will use a transparent quad either in red or green color (plane_green.obj/plane_red.obj). If the selection is permitted the green plane will be rendered, otherwise the red plane will be used. The plane should not be illuminated or receive/cast shadows and its original dimensions are in the [-1 1] range.

7 Enhancements

Feel free to optionally add various enhancements, either visual, inspired by the Lab examples, or otherwise, including night vision, bloom filter, blur, power-ups, more projectiles, laser beams, spells, traps, particles, better game level assets or sound effects and music (using for example OpenAL or SDL_mixer)

8 Project Team

The project will be assigned to teams consisted of 1-3 persons. Project will be submitted and examined after the end of the final semester examination period. Examination of the project for the Erasmus students will be held during the regular examination period.