3d Gaming Assignment2: Character control

Andres Ponce 0616110 andreseeponce@gmail.com

October 30, 2019

THIS MUST BE YOUR OWN WORK: YES

1 Introduction

For this assignment, the main purpose revolved around creating entities and making a type of strategy game. We draw a window and calculate the entities in the selection window. Then, we calculate the trajectories to the destination, and set the animations accordingly. The different systems involved in making the queries can involve many different objects and calculations. Also, we need the different `CEGUI` or `mTrayMgr` classes.

Another purpose of the assignment revolved around creating a particle system around every entity, and change their state when a key is pressed. We could also create new particle systems. Setting them on and off required keeping track of the particle entities and attaching it to the sceneNode of the entity it was supposed to follow around. **Word Count:** 125

2 System Archtecture

The system usually involves creating a selection window and continuously updating it whenever the mouse moves or is released. We need to account for the right and left mouse buttons. The particle system, as mentioned, is attached to the sceneNode when we create each individual robot. These systems require the introduction of the `selection_rectangle` class and the `CEGUI` GUI API.

Word Count: 62

3 Methods

3.1 Task 1

For the first task, we merely change the id set at the top similar to the way in the first assignment, by modifying the `BasicApplication` function in charge of starting the window.

Again, similar to the first assignment, we create a viewport in the `createViewport` function. Here, we also set the ambient light.

Then, we move to the `createScene_00` method, which is in charge of creating the light, robots, and the particle effects. Here, the sections that implement the individual entities are implemented in separate functions, so as to lessen the clutter in the main function. In the `creteRobots` function, we create a circle of 30 robots each, and first attach a particle system. We maintain an array for almost all parameters, since some functionalities require us to change the `SceneNode` or the `Entity` attributes. We attach the (quite bright) smoke particle system to each robot, and we can disable it by pressing the 'M' key.

We then create the plane and the sphere in their separate funcitons. Pretty straightforward definitions, except for these two objects we do not want them to be selectable, so we update their attributes. For the light creation, I tried to continuously calculate their position using `sin`, `cos`, and a constant radius. Although their calculations were wrong, I believe it was merely an issue of properly calculating it.

Then, for the biggest part of the assignment, we create a `selectionRectangle` in `createScene`, and when the left mouse button is pressed, we record the mouse coordinates. When the mouse is released, four

`rayQueries` are performed to determine the coordinates of the rectangle. When we perform a `PlaneBoundedVolumeQuery`, we get the selectable items that lied within the coordinates of the selectionRectangle.

When we press the right mouse button after selecting the items, the calculations are performed to move them towards their desired target points. We switch their animation to "walking", and continuously keep moving them towards the point. Ideally, we create a `Quaternion` object to keep rotating the sceneNode while it is travelling towards the point.

We also perform rudimentary collision detection in relation to the sphere, and translate the items a short distance away if we detect imminent collision.

3.2 Task 2

The objectives in the second task revolved around creating a new camera and viewport. We set the background color to yellow (255, 255, 0). Afterwards, we modify the viewport `mCamera2->AspectRatio(4* vp2->getWidth()/vp2->getHeight)`. Disabling the sky and overlays is just a matter of calling `mCamera2->setSkiesEnabled(` and `vp2->setOverlaysEnable()` methods and setting them to false. **Word Count:** 440

4 Discussion

4.1 How do you draw the selection rectangle?

First, the `selection_rectangle` class must be defined, since we need the coordinates for the edges. Then, while we move the mouse, the coordinates of the rectangle keep changing. The `setCoordinates()` method might be changed with the `CEGUI` API if available, or the `mTrayMgr` interface.

4.2 How do you copute the intersection point between a ray and a plane?

A ray is merely a line extending infinitely in only one direction. From that line, we can calculate the precise point on the plane which it intersects by merely following the ray down. With the help of geometry, calculating the intersection only requires knowing the coordinates of the ray and the angle of incidence with the plane.

4.3 How do you disable skies for the second viewport

Disabling the skies for the second viewport requires the `setSkiesEnabled(false)` method call. Because the viewport is already pointing at the first SceneMgr, the end result remains the same scene, without a skybox image.

4.4 How do do so that the robots walk to the target position?

To ensure that the robots walk to their intended destination, we need to first know the speed at which the robots move and the distance we have to go in each direction. Since we intend to walk in a straight line whenever possible to the destination, subtract the speed per frame from the x, y, and z value of the `distance[i]` vector. Once these values reach 0, the robots have arrived at their destination, since the distance separating them is 0. During this entire process, special care needs to be given to the collision between the robots and the sphere, as well as other robots in the scene. If there is a collision, we merely translate the robot a short distance away in every direction, in order to make the robot enevtually move around the object.

4.5 The background color of the first viewport is set to black. However, why does the background color of the first viewport appear to be white?

The discrepancy is due to the fact that the fog filters are applied to objects a certain distance away. If the object, such as a background, is set far away, it will almost certainly be entirely covered.Thus, even if there

should exist a black background, technically it remains present, however it is obscured from vision due to the fog effects.

4.6 How do you do collision handling for the robots and the sphere?

The main method is to create a sphere around the robot, and enusre that the robot is at least that much distance away from all the other entities. This can be done for the robots as well as the center circle. If we have the coordinates of the periphery of the circle, we can just apply a translation to a safe distance for the robot.

4.7 Did you find any bugs in the program?

The biggest source of frustration were definitely the audio libraries and the `sound` files. Visual Studio continuously complained about their absence from the project, even if their files were present in the folder where they belonged. As a Visual Studio novice, I did not know that we needed to add an existing item, rather than just open the file normally. This was the cause of many nights of frustration and an unanswered email.

Word Count: 479

5 Conclusion

The main takeaway from this assignment remains the sheer complexity of something seemingly so simple as a selection box. The coordinates of the viewport need to be maintained throughout and ray intersections need to be calculated frequently. Since Ogre already does most of the heavy calculations, the user just has to know which objects and methods are required to maintain the coordinates and actually perform the volume selection. The introduction to particle systems also showed the potential for some dynamic encounters once something like a gun can be implemented. Although the one used in the assignment was too bright for my personal taste, as a damage indicator or something of that sort it can still remain a tremendous help for visual effects. **Word Count:** 122