CMSC 425 : Fall 2019 Roger Eastman

Programming Assignment 2: RPG Basics

Handed out: Thursday, Oct. 3rd. **Due:** Friday, Oct. 18th, 11:00pm. Late policy: up to 6 hours late: 5% of the total; up to 24 hours late: 10%, and then 20% for each additional 24 hours.

Overview: The objective of this assignment is to learn more about Unity through the design of the basic structure of an RPG-style game (but not a full game). This includes importing character and object models into Unity, using navigation meshes to control movement, scripted camera control, transitioning between animations using Unity's animation controller system. To make your task simpler, we have based the principal elements around the *Brackeys/Lague RPG Tutorial* (Videos E01, E02, E03, and E10 particularly):

`https://www.youtube.com/watch?v=nu5nyrB9U o&list=PLPV2KyIb3jR4KLGCCAciWQ5qHudKtYeP7`

Caution: Note that it is possible to download the complete project and models from the resources provided with the tutorial. I would caution against this because: (1) you will learn more by following the step-by-step instructions in the video and testing them as you go and (2) there are many things in the final project that you do not need to implement, and figuring out what to delete and what to keep is quite complicated. The Brackeys tutorials are very useful because he is quite proficient in Unity scripting, and you can learn a lot by simply following along as he develops his programs.

Major Components:

Environment: The environment consists of two principal elements, a ground surface and obstacles. You can either use the environment model provided in the tutorial or build your own. The credit you receive will depend on which option you choose.

Ground: This is the (major portion) of the surface on which your characters move. The objective is to demonstrate that you can construct interesting ground models, import them, and build navigation meshes on them.

- Half credit: Use the ground model from the Brackey's tutorial –or– a ground asset downloaded from elsewhere (e.g., the Unity Asset Store) –or– a ground model constructed from primitive Unity 3D shapes.
- Full credit: Build a ground model using Unity's terrain editor with texturing added –or– a polygonal model you build using modeling software, such as Blender (see, e.g., this YouTube tutorial on low-poly terrains.)
- Extra credit: A procedurally generated model of your own design, for example, one based on Perlin noise

Obstacles: These are decorative elements in your environment. The objective is to demonstrate that you can build such models, import them into Unity.

- Half credit: Use the environment models from the Brackey's tutorial –or– assets downloaded from elsewhere (e.g., the Unity Asset Store) –or– a constructed from primitive Unity 3D shapes

- Full credit: Combine downloaded models or Unity primitive shapes together with at least one non-trivial object that you built using modeling software (such as Blender or Tinkercad). This built model might be a low-poly tree, bush, or rock model, a primitive bridge, a small structure, etc.
- Extra credit: Create multiple models of your own, or create a model that is animated (like a windmill with a spinning fan).

How complicated do the models need to be? Since this is not a course in art technology, we are not expecting the models you build to be at all sophisticated (but we will be happy to give extra credit for added complexity). Your model should be complex enough that it is clear that you have an understanding of the basic elements of the modeling software you are using. Also, it will help us greatly if you can *document* your experience with the software. (E.g., How long did it take? What issues did you face? What tutorials did you use?)

Player: The player is a humanoid character, of your chosing. (You may use the character provided in the tutorial or import one either from the Unity Asset Store or from some other resource, such as Adobe's Mixamo Store. For extra credit, build your own, e.g., using Blender. This is a good deal of work though!)

The player character must support at least three modes of navigation: idle, walking, and running. (You may select others if you like, but at a minimum you need to support the ability to stand still, move at a standard speed, and move at a rapid speed).

Following the tutorial, the player navigates about the environment using a click-to-move approach. Left-clicking indicates a point on the ground to which the player should start walking (standard speed). Unlike the tutorial, however, we will use a different mechanism for changing speed. While the user holds down the Shift key, the player character's speed increases (which can be done by setting the `speed` parameter of the nav-mesh agent) and the animation should change from walking to running. When the Shift key is released, the speed returns to normal, and the animation should change to walking (until the velocity falls to zero or a very small value, when it reverts to the idle animation).

Note: The Brackey's tutorial uses a Unity blend tree to control the animation based on the character's actual speed. While this mechanism can be applied to implement this, it is also possible to do this done without the use of a blend tree, simply by triggering transitions between animation states in your script.

Enemy: The player character is chased by a non-player character, called the *enemy*. (You may have more than one if you like.) The enemy is also a humanoid character, and it should have a different appearance than the player character. It need only support two animations, idle and walking. As in the tutorial, whenever the player character comes sufficiently close to the enemy character, the enemy starts chasing the player character. As with the player, the enemy uses the navigation mesh to move. If the player moves to a sufficient large distance from the enemy, the enemy stops chasing the player.

There is no requirement for any action if the enemy catches up with the player. You may implement some sort of behavior for extra credit.

Camera Control: The camera control should be the same as in the tutorial, following the player from a fixed direction, but the user can zoom in and out, can yaw around the

player character. In addition to the tutorial, you should allow the camera to crane up and down by accessing the vertical input (using, for example, Input.GetAxis("Vertical"). In our implementation, this input simply moved the camera up and down, but you can implement this in any reasonable manner.

Online Resources and Tutorials: In addition to the Brackeys/Lague RPG tutorial, there are a number of online resources that were helpful in designing our implementation:

- https://unity3d.com/learn/tutorials/topics/navigation/navmesh-agent: The Unity video tutorial on NavMeshes. Also check out the videos under the section "Live Sessions on Navigation"
- https://docs.unity3d.com/Manual/Navigation.html: The Navigation section from the Unity manual. Check out the subsection on "Navigation How-Tos" especially "Moving an Agent to a Position Clicked by the Mouse" and "Coupling Animation and Navigation." (Note that with the simple animations that I used, I had some difficulty getting the method described in "Coupling Animation and Navigation" to work
- https://www.youtube.com/watch?v=*...rest of link omitted*: A tutorial by Holistic3d entitled "Mechanim & Mixamo in Unity 5". This explains how to create and download a Mixamo model and transition between animations. (I would recommend using this method for our project, rather than the blend-tree approach that is discussed in the Brackeys/Lague tutorial.)