Victor Haskins

GAME236

Lab Report and post Questions for Lab 2

**Introduction**

Compounding on our work last week with tool creation, we are set to create two new tools: prefab generation and sprite animation creation. After successfully building them, we are tasked with combining them to create a tool that will make a prefab of a sprite animation.

**Content**

We started by creating a tool for creating a prefab of an object or series of objects. Our goal is to create a prefab of every object selected in the hierarchy when the tool is activated. The first line is to create an array of objects filled in with the selected objects. Then, for each item in the array, we take the name, and create the location string of where it will be created. Then we created an if-else statement that checks if a prefab of that name already exists using the line "AssetDatabase.LoadAssetAtPath(string path, string type)" in the qualifying statement. We also get our first use of the EditorUtility.DisplayDialog option which pops up a warning window for us in an if-else statement giving us a choice to continue with our current action of overriding a prefab, or to cut the action altogether.

The next step is to call another function to keep from overloading the front end of this script. It is called around the load asset if-else statement when we either don't have a pre-existing asset or don't care if one gets overwritten. Our CreateNew(object, location) function creates a new, empty prefab at the location we created earlier with the action, "PrefabUtility.CreateEmptyPrefab(location)", followed by ReplacePrefab(object, prefab) to swap out the newly created prefab object with the object we wanted made. Finally, we refresh the project with AssetDatabase.Refresh(), Destroy the instance of the object passed into the function that we no longer need with DestroyImmediate(obj) because a normal Destroy() function doesn't work outside of runtime, and then places the prefab with the actionPrefabUtility. InstantiatePrefab(prefab) as GameObject.

After our prefab generation tool was finished we were asked to reaquaint ourselves with 2D animations created from sprite sheets. Truth be told I spent a lot of time on this and got hung up a few times, despite working on this quite a bit for the final project of last class.

To begin, we need to have a sprite sheet loaded to the assets, switched from texture to sprite, from single to multiple, and already been sliced into the individual sprites.

From here, we start off similar to the last tool by having the sprite sheet selected, but this time it is just the sprite sheet as it will automatically pull all the individual sprites later. We will also create a buch of global variables for later use in the tool. We start with a public static object for the sprite sheet selected, and then a string for the name of the controller, an array of strings for each animation name, arrays for each animation's frame rate, time between frames, start frame, end frame, if each should loop, and if each should ping pong, which apparently means to reverse through the sprites after playing forwards through them.

Now we begin the tool creation; we start with checking to see if an object is selected, if not, then the action is aborted. Otherwise, we open a new window for the user to input information into. This brings us to point out that the class type, written after the colon on the class declaration line, needs to be switched to an EditorWindow from MonoBehavior. This will allow us to create that much needed new window. We then show it to the user.

The next function is to create what the window will display with the OnGUI() function. We display the name of the sprite sheet entered into the tool, and then ask for information. This is carried out with the help of the command, "EditorGUILayout". We use them instead of EditorGUI because they will automatically format to the window, whereas the latter needs to be manually set. We start with asking for the name of the animator and the number of animations to be created. After the number of animations is entered, a for loop runs to give us the opportunity to enter information for each instance. Most options appear one piece of data per vertical line unless given a grouping to be placed horizontally. After the line to take a name for the individual animation, this is how most everything else is represented. We then have variable inputs created for each of the variables listed above in arrays and finish it with the calling a create button with "if(GUILayout.Button("Create"))..."

When the create button is pressed, we start with creating the controller for the animator with the line, "UnityEditor.Animations.AnimatorController controller =UnityEditor.Animations.AnimatorController.CreateAnimatorControllerAtPath(("Assets/" + controllerName + ".controller"));" Then we create a tempClip item that will call a new function designed to plot out the playing pattern of the animation based on whether or not it will ping pong, then will be set to loop, affecting both the time and blend values for looping, if it should, and finally added to the animator controller.

The CreateClip function needs to be passed the individual object, clip name, start and end frames, frame rate, and timeBehavior from the for loop to know just what it needs to make. The frame rate is divided by the number of frames from the start to end, (times two if the animation is set to ping pong,) and then sets the each sprite at a new keyframe as needed in the time line. For a normal animation it starts with the first and ends with the last sprite provided, but a ping pong animation will cycle through all of those and then add all but the last sprite again, but in reverse order each at the frame interval specified. Finally the animation is completed and sent back to be added to the animator.

Finally, we are asked to create a tool that will combine the previous two tools that will make a prefab out of an animator as well as add some error handling to make sure that no animation tries to go past the number of sprites sent into the tool.

**Conclusion**

I am very bad at trying to find object types apparently, as I spend over five hours trying to transfer the animator controller I had created to a new function. Even after finding a workaround, my work still doesn't seem to work right. Though I think it's supposed to be working, it isn't. In the long run, I am excited to see this used in the future and like the idea of preparing now to save a lot of time later, especially with how crazy this semester looks to become.

**Post Lab Questions**

*What is the difference between EditorGUI and EditorGUILayout?*

Every instance of EditorGUI needs to have the view information edited while the EditorGUILayout setup automatically sets up.

*What is the OnGUI function?*

It shows what the developer creates when the current script calls for a new application window.

*Do variables have to be initialized before they can be used for GUI items?*

Yes, unless they are items to show information, and even then they often show priorly collected information.

*Instead of hard-coding the prefabs to go into the asset folder, how could you allow the user to choose what folder to create the prefab?*

Add an option in collecting information that could grab the folder location.

*What is the difference between Instantiate and InstantiatePrefab?*

The former places and instance of a new object while the latter places a new prefab instance.

*What are some things you can do with EditorUltility.DisplayDialoge?*

To give caution warnings in case the user was about to do something that might erase previous data or create errors in the new data.

*How would you use the utility created in this section?*

It offers the groundwork for creating nearly any prefab the class wants to in the future. As of right now, it could be invaluable in creating several animations quickly.

*Why must you refresh the asset database after creating the prefabs?*

To show the changes made to the project.