**Introduction**

There were two challenges for this lab that did work together in the end. The first task was to create a prefab from an object in the hierarchy inside unity. The second was automating the process of making an animation controller using an unity editor gui window. The goal of these two was to make a tool to help a animator or designer create an animation controller from an object and turning it into a prefab.

**Methods**

The first task was just grabbing the object from the hierarchy checking to see if the prefab existed in the asset folder already if it did create a popup window asking if you want to override the file with a new one. For the animation process you need the selected object then create a window where the user can type in the name of the controller and names for all the animations. User can choose the frame rate and time between frames for all animations, startframe, endframe, and if you want the animations to either pingpong or loop. If no object do nothing, but after collecting all the info create an empty animation for every animation create an animation clip and set if loop and add the clip to controller.

During the creation process the function will get the path to the sprite extract the sliced parts and determine how many frames it has and time between each frame. Then create an empty animation clip and set the frame rate. Create a curve on the animation on the sprite renderer and make the curve a sprite type property. If no pinpong initialize keyframes to calculated amount if it is pingpong it will be twice as long. Keeping track of how many frames create a keyframe for every frame in between start and end and store them into the keyframes array and keep track of every frame added. Same for pingpong. Assign he name of the clip and bind the curve and keyframes to the animation. Use this to create the animation clip and return the clip to the previous function.

**Conclusion**

For the first part I learned how to check files if they exist and how to overwrite files by using dialog windows. The second part of the lab showed a lot of the inner workings of animation controllers but showed how window forms work inside unity, the power of that tool was great to see. Everything seems really accessible now. I could use what I learned to any system inside unity to create new objects with pre-existing functions and access different properties all around to modify how a user wants it.

1. EditorGUI requires you place all the options manually with x’s and y’s in the window and EditorGUILayout automates the placing of items in the window.’
2. The OnGUI function lets you draw all the items on your window.
3. Yes
4. You could put in a variable that the user can edit
5. Instantiate makes a clone of the gameObject in the hierarchy and Instantiate prefab creates a new object with a prefab connection to the prefab.
6. Displays a message box in the editor gets a return of true if the default button is pressed.
7. I would use the animation creator for sprite sheets for an easy start to making an entire controller
8. Because the project window and the editor wont show or know about the updates you just made.

using UnityEngine;

using System.Collections;

using UnityEditor;

public class ScriptToolMakePrefab : MonoBehaviour {

[MenuItem("Project Tools/Create Prefab")]

public static void CreatePrefab()

{

GameObject[] selectedObjects = Selection.gameObjects;

foreach (GameObject go in selectedObjects)

{

string name = go.name;

string assetPath = "Assets/" + name + ".prefab";

if(AssetDatabase.LoadAssetAtPath(assetPath, typeof(GameObject)))

{

Debug.Log("Asset exists!");

if(EditorUtility.DisplayDialog("Caution", "Prefab already exists. " +

"Do you want to overwrite?", "Yes", "No"))

{

Debug.Log("Overwriting!");

CreateNew(go, assetPath);

}

}

else

{

CreateNew(go, assetPath);

}

//Debug.Log("Name: " + go.name + "Path: " + assetPath);

}

}

public static void CreateNew(GameObject obj, string location)

{

Object prefab = PrefabUtility.CreateEmptyPrefab(location);

PrefabUtility.ReplacePrefab(obj, prefab);

AssetDatabase.Refresh();

DestroyImmediate(obj);

GameObject clone = PrefabUtility.InstantiatePrefab(prefab) as GameObject;

}

}

using UnityEngine;

using System.Collections;

using UnityEditor;

public class MakeAnimations : EditorWindow {

//This is static because the menu function is also static\*

public static Object selectedObject;

int numAnimations;

string controllerName;

string[] animationNames = new string[100];

float[] clipFrameRate = new float[100];

float[] clipTimeBetween = new float[100];

int[] startFrames = new int[100];

int[] endFrames = new int[100];

bool[] pingPong = new bool[100];

bool[] loop = new bool[100];

/// <summary>

/// store the selected objects and open a popup window

/// </summary>

[MenuItem("Project Tools/2D Animations")]

static void Init()

{

selectedObject = Selection.activeObject;

if(selectedObject == null)

{

return;

}

MakeAnimations window = (MakeAnimations)EditorWindow.GetWindow(typeof(MakeAnimations));

window.Show();

}

void OnGUI()

{

if(selectedObject != null)

{

EditorGUILayout.LabelField("Animations for " + selectedObject.name);

EditorGUILayout.Separator();

controllerName = EditorGUILayout.TextField("Controller Name", controllerName);

numAnimations = EditorGUILayout.IntField("How many animations?", numAnimations);

for (int i = 0; i < numAnimations; i++)

{

//Get the animation name

animationNames[i] = EditorGUILayout.TextField("Animation Name", animationNames[i]);

//Start a section where the items are displayed horizontally

EditorGUILayout.BeginHorizontal();

startFrames[i] = EditorGUILayout.IntField("Start Frame", startFrames[i]);

//get end frame

endFrames[i] = EditorGUILayout.IntField("End Frame", endFrames[i]);

EditorGUILayout.EndHorizontal();

//Start a section where the following items will be displayed horizontally instead of vertically

EditorGUILayout.BeginHorizontal();

//Determine the frame rate for the animation

clipFrameRate[i] = EditorGUILayout.FloatField("Frame Rate", clipFrameRate[i]);

//Determine the space between each keyframe

clipTimeBetween[i] = EditorGUILayout.FloatField("Frame Spacing", clipTimeBetween[i]);

//End the section where the previous items are displayed horitontally instead of vertically

EditorGUILayout.EndHorizontal();

//Start a section where the following items will be displayed horizontally instead of vertically

EditorGUILayout.BeginHorizontal();

//Create a checkbox to determine if this animation should loop

loop[i] = EditorGUILayout.Toggle("Loop", loop[i]);

//Create a checkbox to determine if this animation should pingpong

pingPong[i] = EditorGUILayout.Toggle("Ping Pong", pingPong[i]);

//End the section where the previous items are displayed horitontally instead of vertically

EditorGUILayout.EndHorizontal();

EditorGUILayout.Separator();

}

if (GUILayout.Button("Create"))

{

UnityEditor.Animations.AnimatorController controller = UnityEditor.Animations.AnimatorController.CreateAnimatorControllerAtPath(("Assets/" + controllerName + ".controller"));

for(int i = 0; i< numAnimations; i++)

{

AnimationClip tempClip = CreateClip(selectedObject, animationNames[i], startFrames[i], endFrames[i], clipFrameRate[i], clipTimeBetween[i], pingPong[i]);

if (loop[i])

{

//set the loop on the clip to true

AnimationClipSettings settings = AnimationUtility.GetAnimationClipSettings(tempClip);

settings.loopTime = true;

settings.loopBlend = true;

AnimationUtility.SetAnimationClipSettings(tempClip, settings);

}

controller.AddMotion(tempClip);

}

}

}

}

void OnFocus()

{

if (EditorApplication.isPlayingOrWillChangePlaymode)

{

this.Close();

}

}

public AnimationClip CreateClip(

Object obj,

string clipName,

int startFrame,

int endFrame,

float frameRate,

float timeBetween,

bool pingPong)

{

string path = AssetDatabase.GetAssetPath(obj);

Object[] sprites = AssetDatabase.LoadAllAssetsAtPath(path);

//check end frame

if (endFrame < sprites.Length)

{

//Determine how many frames, and the length of each frame

int frameCount = endFrame - startFrame + 1;

float frameLength = 1f / timeBetween;

//Create a new (empty animation clip

AnimationClip clip = new AnimationClip();

clip.frameRate = frameRate;

EditorCurveBinding curveBinding = new EditorCurveBinding();

//Assing it to change the sprite renderer

curveBinding.type = typeof(SpriteRenderer);

//assign it to the sprite renderer

curveBinding.propertyName = "m\_Sprite";

//Create a container for all of the keyframes

ObjectReferenceKeyframe[] keyFrames;

//Determine how many frames if ping ponging

if (!pingPong)

{

keyFrames = new ObjectReferenceKeyframe[frameCount + 1];

}

else

{

keyFrames = new ObjectReferenceKeyframe[frameCount \* 2 + 1];

}

//frame counter

int frameNumber = 0;

//loop from start to end

for (int i = startFrame; i < endFrame + 1; i++, frameNumber++)

{

//create empty keyframe

ObjectReferenceKeyframe tempKeyFrame = new ObjectReferenceKeyframe();

//assign it a time to appear in the anim

tempKeyFrame.time = frameNumber \* frameLength;

//assign it a sprite

tempKeyFrame.value = sprites[i];

//place it into the container for all the keyframes

keyFrames[frameNumber] = tempKeyFrame;

}

//If we are pinponging this aimation

if (pingPong)

{

//create keyframes starting at the end and going backwards

//continue to keep track of the frame number

for (int i = endFrame; i >= startFrame; i--, frameNumber++)

{

ObjectReferenceKeyframe tempKeyFrame = new ObjectReferenceKeyframe();

tempKeyFrame.time = frameNumber \* frameLength;

tempKeyFrame.value = sprites[i];

keyFrames[frameNumber] = tempKeyFrame;

}

}

//Create the last sprite to stop it from switching quickly from the last frame to the first one

ObjectReferenceKeyframe lastSprite = new ObjectReferenceKeyframe();

lastSprite.time = frameNumber \* frameLength;

lastSprite.value = sprites[startFrame];

keyFrames[frameNumber] = lastSprite;

//Assign the name

clip.name = clipName;

//apply the curve

AnimationUtility.SetObjectReferenceCurve(clip, curveBinding, keyFrames);

//create the clip

AssetDatabase.CreateAsset(clip, ("Assets/" + clipName + ".anim"));

//return the clip

return clip;

}

else

{

Debug.Log("endFrame is greater than amount of slices the sprite has.");

return null;

}

}

}