**Lab Report**

**Lab 02 - Prefabs and 2D Sprites**

**Craig Broskow**

**September 2, 2015**

Introduction:

Custom tools assist game programming teams in multiple ways. For one thing, they can address the special needs of game developers working on a particular project. This can save valuable time for the game developers as well as improve their efficiency and morale. Just as importantly, custom tools can concentrate on common or highly detrimental errors that occur within a game development team. The prevention of errors before they occur also save time and money, as well as improve morale by eliminating "busy work".

In this lab, we created some tools that add menu items to the Unity editor that improve the efficiency of prefab creation and the automation of 2D sprite animations. These tools are faster to use and reduce the possibility of errors introduced by manually performing that work.

Methods:

The first menu item added to the Unity editor was "Create Prefab". This is a fairly simple script that: (1) Determines which game object(s) the user has selected; (2) Checks whether the game object(s) already have prefabs created for them; (3) Confirms with the user if a prefab will be overwritten; (4) Creates a new prefab and populates it with the game object; and (5) Deletes the game object(s), replacing them with the new prefab(s).

The second menu item added to the Unity editor was "2D Animations". This is a slightly more complicated script because it requires data input from the user. The data input was achieved by creating an Editor window that allows the user to enter several pieces of information regarding the user-selected sprite sheet, such as the number of animation clips it contains, the starting and ending frame numbers of the animation clips and other characteristics of the clips such as whether they loop or pingpong. Once the data has been entered by the user in the Editor window, an Animator Controller is created, the Animation Clips are added to the Animation Controller, and the parameters of the Animations Clips are set...including keyframes, frame spacing and animation curve.

Finally, we combined the essential parts of both of the initial portions of this lab into a third menu item added to the Unity editor which I called "On Your Own". This third script duplicates much of the code contained within the first two scripts...primarily due to time constraints. This tool accomplishes the following tasks for the user: (1) A user-selected sprite sheet is broken into one or more animation clips, based on data entered by the user; (2) Additional data validation is performed, specifically to prevent the entry of frame numbers greater than the total number of frames in the sprite sheet; (3) A sprite is created in the scene view at (0, 0, 0) with an Animation Controller containing the Animation Clips extracted from the specified sprite sheet; (4) The sprite is used to create a Prefab in the main "Assets" folder; and (5) The scene view sprite is replaced by the just-created Prefab.

Conclusions:

The Unity C# code developed in this lab will certainly be useful in creating similar, and even very different, tools in the future. Modifying the Editor window to allow users to select objects and then pick custom operations to perform on them opens the door to significant Unity extensions...especially for tool development. One of the main things learned in this lab was the creation of Editor windows that allow for simple, validated data entry by the user. The creation and manipulation of prefabs will allow users to create a variety of template objects that will assist game development teams. The manipulation of 2D animations seems to be primarily limited to 2D games, but perhaps it is more significant than that and I simply don't see a broader usefulness yet.

Postlab Questions:

1. What is the difference between EditorGUI and EditorGUILayout?

EditorGUILayout allows for the automatic placement of display items, while EditorGUI requires detailed specification of the display layout.

2. What is the OnGUI function?

At least in this lab, the OnGUI function is used to control the display and user interaction with the popup Editor window. I believe it has other uses in Unity, such as with the legacy GUI system.

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

No. The user will generally enter the data to populate the variables for the GUI items.

4. 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?

Since the path to the folder to store prefabs is simply a string, the popup Editor window could include a text field that would allow the user to specify an existing folder, or a new one to create.

5. What is the difference between Instantiate and InstantiatePrefab?

InstantiatePrefab is used specifically to create game objects from prefabs. Instantiate, on the other hand, is used to copy game objects to other game objects.

6. What are some things you can do with EditorUtility.DisplayDialog?

EditorUtility.DisplayDialog is very limited in that it can only ask yes/no type questions. However, it displays the dialog modally...so it can be used to confirm that a user **really** wants to do what they are requesting. That would be the most important usage I can see.

7. How would you use the utility created in this section?

At the current time, I do not foresee much work in 2D Unity. However, the future is very difficult to discern, so I can't say that for sure. Even so, I don't see working with sprite sheets and the creation of 2D animations to be all that difficult personally...so I would probably continue to do it manually. If I worked for a company that employed a lot of 2D artists, I can see offering a utility like this for their use.

8. Why must you refresh the asset database after creating the prefabs?

Prefabs are considered to be assets, and AssetDatabase.Refresh() updates the Unity project with the new additions.

Code:

// MakeAnimations.cs

// Craig Broskow - GAME 221 - Lab 02

using UnityEngine;

using System.Collections;

using UnityEditor;

public class MakeAnimations : EditorWindow {

// Will hold the object that the user has selected when the script is run

public static Object selectedObject;

// Will store how many animations will be created

int numAnimations;

// Name of the controller to be created

string controllerName;

// Name of each of the animations to be created

string[] animationNames = new string[100];

// The frame rate for each animation

float[] clipFrameRate = new float[100];

// The time between each animation

float[] clipTimeBetween = new float[100];

// What frame each animation starts at

int[] startFrames = new int[100];

// What frame each animation ends at

int[] endFrames = new int[100];

// If each animation should pingpong

bool[] pingPong = new bool[100];

// If each animation should loop

bool[] loop = new bool[100];

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

static void Init()

{

// Grab the active object

selectedObject = Selection.activeObject;

// If the object doesn't exist, do nothing

if (selectedObject == null)

return;

// Otherwise, create a new window

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

// Show the window

window.Show();

} // end method Init

void OnGUI()

{

if (selectedObject != null)

{

// Display the object's name that the animations will be created from

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

// Create a space

EditorGUILayout.Separator();

// Get the name for the animation controller

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

// Determine how many animations there will be

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

// Loop through each theoretical animation

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

{

// Determine a name for the animation

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

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

EditorGUILayout.BeginHorizontal();

// Determine the start frame for the animation

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

// Determine the end frame for the animation

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

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

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 horizontally 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 horizontally instead of vertically

EditorGUILayout.EndHorizontal();

// Create a space

EditorGUILayout.Separator();

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

// Create a button with the label "Create"

if (GUILayout.Button("Create"))

{

//Create an animator controller

UnityEditor.Animations.AnimatorController controller =

UnityEditor.Animations.AnimatorController.CreateAnimatorControllerAtPath(("Assets/" +

controllerName + ".controller"));

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

{

// Create animation clip

AnimationClip tempClip = CreateClip(selectedObject, animationNames[i], startFrames[i],

endFrames[i], clipFrameRate[i], clipTimeBetween[i], pingPong[i]);

// Determine if the clip should loop

if (loop[i])

{

// If so, capture the settings of the clip

AnimationClipSettings settings = AnimationUtility.GetAnimationClipSettings(tempClip);

// Set the looping to true

settings.loopTime = true;

settings.loopBlend = true;

// Apply the settings to the clip

AnimationUtility.SetAnimationClipSettings(tempClip, settings);

}

// Add the clip to the Animator Controller

controller.AddMotion(tempClip);

}

}

}

} // end method OnGUI

public AnimationClip CreateClip(Object obj, string clipName, int startFrame, int endFrame,

float frameRate, float timeBetween, bool pingPong)

{

// Get path to the object

string path = AssetDatabase.GetAssetPath(obj);

// Extract the sprites

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

// 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();

// Set the framerate for the clip

clip.frameRate = frameRate;

// Create the new (empty) curve binding

EditorCurveBinding curveBinding = new EditorCurveBinding();

// Assign it to change the sprite renderer

curveBinding.type = typeof(SpriteRenderer);

// Assign it to alter the sprite of the sprite renderer

curveBinding.propertyName = "m\_Sprite";

// Create a container for all of the keyframes

ObjectReferenceKeyframe[] keyFrames;

// Determine how many frames there will be if we are or are not pingponging

if (!pingPong)

keyFrames = new ObjectReferenceKeyframe[frameCount + 1];

else

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

// Keep track of what frame number we are on

int frameNumber = 0;

// Loop from start to end, incrementing frameNumber as we go

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

{

// Create an empty keyframe

ObjectReferenceKeyframe tempKeyFrame = new ObjectReferenceKeyframe();

// Assign it a time to appear in the animation

tempKeyFrame.time = frameNumber \* frameLength;

// Assign it to a sprite

tempKeyFrame.value = sprites[i];

// Place it into the container for all the keyframes

keyFrames[frameNumber] = tempKeyFrame;

}

// If we are pingponging this animation

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;

} // end method CreateClip

} // end class MakeAnimations

// MakePrefabs.cs

// Craig Broskow - GAME 221 - Lab 02

using UnityEngine;

using System.Collections;

using UnityEditor;

public class MakePrefabs : 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"))

{

CreateNew(go, assetPath);

}

}

else

{

CreateNew(go, assetPath);

}

}

} // end method CreatePrefab

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;

} // end method CreateNew

} // end class MakePrefabs

// OnYourOwn.cs

// Craig Broskow - GAME 221 - Lab 02

using UnityEngine;

using System.Collections;

using UnityEditor;

public class OnYourOwn : EditorWindow {

// Will hold the object that the user has selected when the script is run

public static Object selectedObject;

// Will store how many animations will be created

int numAnimations;

// Name of the controller to be created

string controllerName;

// Name of each of the animations to be created

string[] animationNames = new string[100];

// The frame rate for each animation

float[] clipFrameRate = new float[100];

// The time between each animation

float[] clipTimeBetween = new float[100];

// What frame each animation starts at

int[] startFrames = new int[100];

// What frame each animation ends at

int[] endFrames = new int[100];

// If each animation should pingpong

bool[] pingPong = new bool[100];

// If each animation should loop

bool[] loop = new bool[100];

[MenuItem("Project Tools/On Your Own")]

static void Init()

{

// Grab the active object

selectedObject = Selection.activeObject;

// If the object doesn't exist, do nothing

if (selectedObject == null)

return;

// Otherwise, create a new window

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

// Show the window

window.Show();

} // end method Init

void OnGUI()

{

if (selectedObject != null)

{

// Determine the maximum frame number in the sprite sheet.

int maxFrameNumber =

AssetDatabase.LoadAllAssetsAtPath(AssetDatabase.GetAssetPath(selectedObject)).GetUpperBound(0);

// Display the object's name that the animations will be created from

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

// Create a space

EditorGUILayout.Separator();

// Get the name for the animation controller

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

// Determine how many animations there will be

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

// Loop through each theoretical animation

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

{

// Determine a name for the animation

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

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

EditorGUILayout.BeginHorizontal();

// Determine the start frame for the animation

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

// Determine the end frame for the animation

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

// Limit the end frame number to the number of frames in the sprite sheet.

if (endFrames[i] > maxFrameNumber)

endFrames[i] = maxFrameNumber;

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

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 horizontally 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 horizontally instead of vertically

EditorGUILayout.EndHorizontal();

// Create a space

EditorGUILayout.Separator();

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

// Create a button with the label "Create"

if (GUILayout.Button("Create"))

{

//Create an animator controller

UnityEditor.Animations.AnimatorController controller =

UnityEditor.Animations.AnimatorController.CreateAnimatorControllerAtPath(("Assets/" +

controllerName + ".controller"));

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

{

// Create animation clip

AnimationClip tempClip = CreateClip(selectedObject, animationNames[i], startFrames[i],

endFrames[i], clipFrameRate[i], clipTimeBetween[i], pingPong[i]);

// Determine if the clip should loop

if (loop[i])

{

// If so, capture the settings of the clip

AnimationClipSettings settings = AnimationUtility.GetAnimationClipSettings(tempClip);

// Set the looping to true

settings.loopTime = true;

settings.loopBlend = true;

// Apply the settings to the clip

AnimationUtility.SetAnimationClipSettings(tempClip, settings);

}

// Add the clip to the Animator Controller

controller.AddMotion(tempClip);

}

// Create a sprite in the scene view at (0,0,0) with the new animation controller.

GameObject newSprite = new GameObject();

newSprite.transform.position = Vector3.zero;

newSprite.name = controllerName + "Sprite";

newSprite.AddComponent<SpriteRenderer>();

newSprite.AddComponent<Animator>();

newSprite.GetComponent<Animator>().runtimeAnimatorController = controller;

// Make the sprite a prefab in the main "Assets" folder.

Object prefab = PrefabUtility.CreateEmptyPrefab("Assets/" + controllerName + "Sprite.prefab");

PrefabUtility.ReplacePrefab(newSprite, prefab);

AssetDatabase.Refresh();

DestroyImmediate(newSprite);

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

} // if (GUILayout.Button("Create"))

}

} // end method OnGUI

public AnimationClip CreateClip(Object obj, string clipName, int startFrame, int endFrame,

float frameRate, float timeBetween, bool pingPong)

{

// Get path to the object

string path = AssetDatabase.GetAssetPath(obj);

// Extract the sprites

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

// 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();

// Set the framerate for the clip

clip.frameRate = frameRate;

// Create the new (empty) curve binding

EditorCurveBinding curveBinding = new EditorCurveBinding();

// Assign it to change the sprite renderer

curveBinding.type = typeof(SpriteRenderer);

// Assign it to alter the sprite of the sprite renderer

curveBinding.propertyName = "m\_Sprite";

// Create a container for all of the keyframes

ObjectReferenceKeyframe[] keyFrames;

// Determine how many frames there will be if we are or are not pingponging

if (!pingPong)

keyFrames = new ObjectReferenceKeyframe[frameCount + 1];

else

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

// Keep track of what frame number we are on

int frameNumber = 0;

// Loop from start to end, incrementing frameNumber as we go

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

{

// Create an empty keyframe

ObjectReferenceKeyframe tempKeyFrame = new ObjectReferenceKeyframe();

// Assign it a time to appear in the animation

tempKeyFrame.time = frameNumber \* frameLength;

// Assign it to a sprite

tempKeyFrame.value = sprites[i];

// Place it into the container for all the keyframes

keyFrames[frameNumber] = tempKeyFrame;

}

// If we are pingponging this animation

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;

} // end method CreateClip

} // end class OnYourOwn