CS 484/684 Homework 03
Game Jam: Halloween

Daniel Lind

September 23, 2019

Source Code Link: `https://github.com/dlind9/cs484/tree/master/HW3`
This homework took approximately 6 hours to complete.

1 Design

I spent about 30 minutes trying to figure out what kind of game I could actually make that fell in line with this weeks game jam. After settling on making a platformer where you had to collect a certain amount of candy throughout the level. The candy falls in line with the Halloween theme. Once I got to there I figured out what kind of textures I would want to use and then I got to making the game.

2 Post Mortem

I spent about an hour figuring out how to get collisions working between the tiles and the player so he wouldn't just fall through. I ran into a bit of issue as there are two sorts of colliders in unity. There are the 3D colliders and then there are 2D, the problem being that they do not interact with one another. After that I figured out how to set up a counter with the candy using colliders for triggering the pick up and an object that handled the counter.

3 Answers to Questions

In this section, you will write the answers to the questions in the homework assignment.

- Vertex shader is used for every vertex while a fragment shader is for every pixel. …
- You would use a ”uniform” variable …
- You would need a window and vertices. …
- A pixel is a screen element while a sample is literally a point on a screen so one pixel can have multiple samples. …
- …

4 Sample Output

My program generated the image in Figure 1.

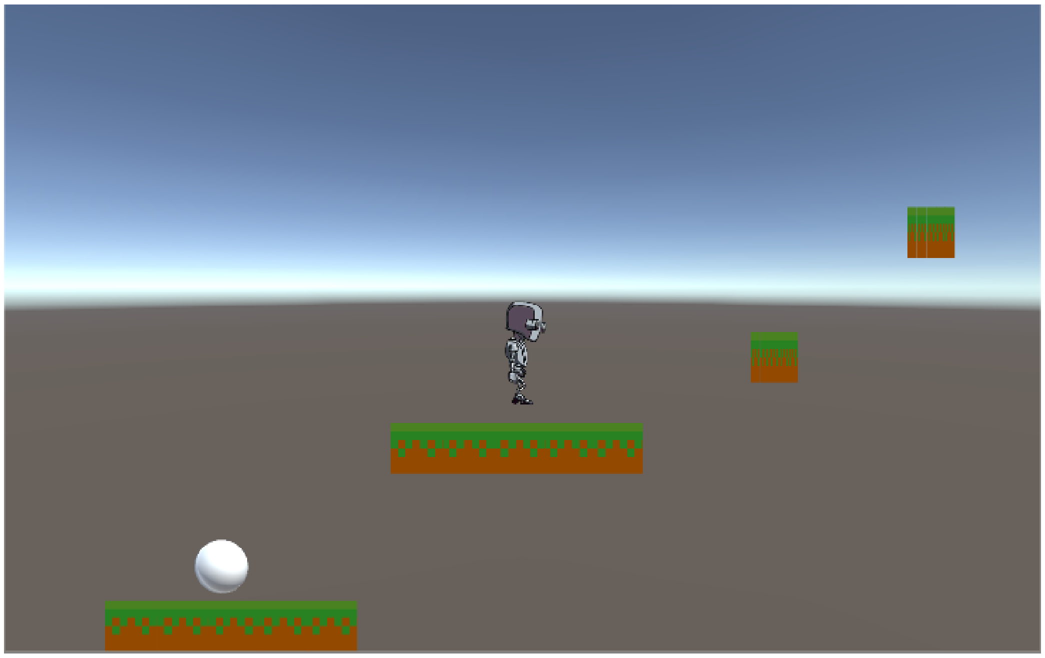

Figure 1: Screenshot of my program

5 Main Requirement Source Code

5.1 Player Controls Script

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets._2D
{
    [RequireComponent(typeof (PlatformerCharacter2D))]
    public class Platformer2DUserControl : MonoBehaviour
    {
        private PlatformerCharacter2D m_Character;
        private bool m_Jump;


        private void Awake()
        {
            m_Character = GetComponent<PlatformerCharacter2D>();
        }


        private void Update()
        {
            if (!m_Jump)
            {
                // Read the jump input in Update so button presses aren't missed.
                m_Jump = CrossPlatformInputManager.GetButtonDown("Jump");
            }
        }


        private void FixedUpdate()
        {
            // Read the inputs.
            bool crouch = Input.GetKey(KeyCode.LeftControl);
            float h = CrossPlatformInputManager.GetAxis("Horizontal");
            // Pass all parameters to the character control script.
            m_Character.Move(h, crouch, m_Jump);
            m_Jump = false;
        }
    }
}
```

5.2 Kill Restart Script

```
using System;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace UnityStandardAssets._2D
{
    public class Restarter : MonoBehaviour
    {
        private void OnTriggerEnter2D(Collider2D other)
        {
            if (other.tag == "Player")
            {
                SceneManager.LoadScene(SceneManager.GetSceneAt(0).name);
            }
        }
    }
}
```

5.3 Candy Scripts

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class candyCounter : MonoBehaviour
{
    private int maxCandy;
    public int currentCandy;
    // Start is called before the first frame update
    void Start()
    {
        maxCandy = 4;
        currentCandy = 0;
    }

    public void candyGrabbed()
    {
        if (currentCandy < 4)
        {
            currentCandy++;
        }
        else
        {
            return;
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class candyGrabbed : MonoBehaviour
{
    public candyCounter candyCounter;

    private void OnTriggerEnter2D(Collider2D other)
    {
        candyCounter.candyGrabbed();
        Destroy(this.gameObject);
        Debug.Log("Candy grabbed");
    }
}
```

6 Additional Requirement: 3D Engine Shaders

In Unity there are several different types of shaders that you can implement. There are surface shaders, vertex and fragment shaders, and fixed function shaders. However, no matter which kind of shader you use they are all wrapped in the

ShaderLab language. When it comes to surface shaders and vertex and fragment shaders, all you have to is write a couple lines in the Cg/HLSL language and then your ShaderLab code will be generated. Surface shaders are primarily used when you're dealing with shadows and lighting on surfaces. Vertex and fragment shaders are required if you aren't dealing with either of those. But vertex and fragment shaders are the most flexible of them and you can implement the most exotic effects with them and even make your own lighting/shadow effects. Fixed function shaders are the simplest and used for very basic effects. Shaders are applied by creating a material asset within your unity assets and attaching the shader using a dropdown menu.

7 Additional Requirement: ShaderToy

I spent quite a bit looking up basic tutorials on how to use ShaderToy to get an idea of where to start. I also looked at quite a bit of examples that I could find on ShaderToy. Afterwards I just spent most of my time messing around with cos and sin functions so see if I could get interesting effects in the Shader. Once I figured that out I also included time for change over time in my shaders.
Source Code Link: `https://www.shadertoy.com/view/Wdd3Rs`