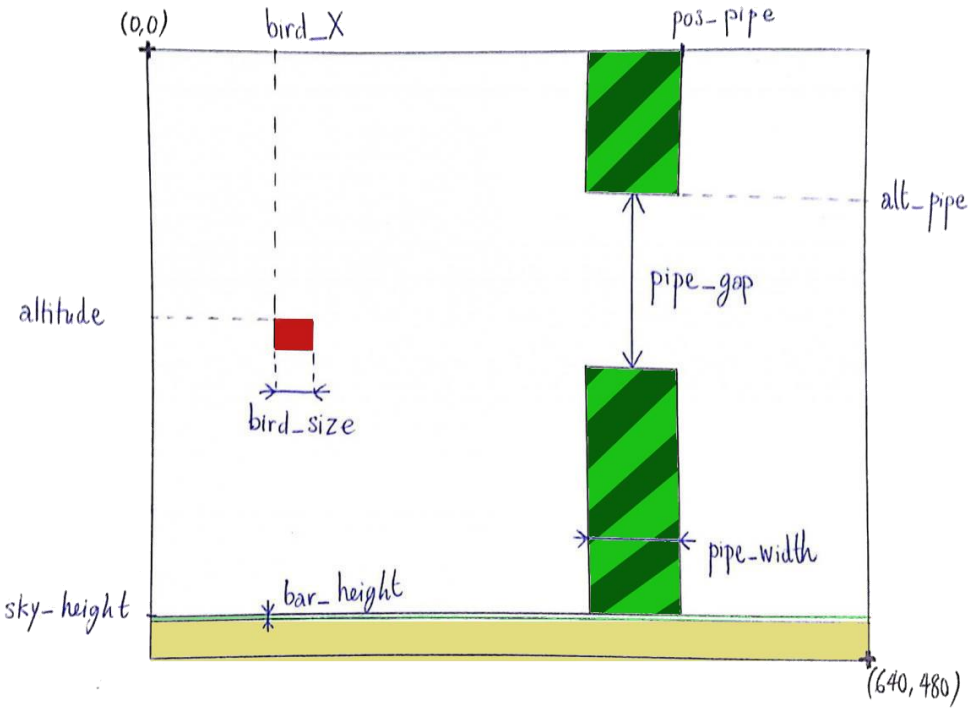


Budget Chicken
(Flappybird clone) UWP Project

CSTP 1302 COURSE PROJECT

Steven Uhm 000455287 | CSTP 1302: Windows Programming | July 8th, 2022

Introduction

This project was mainly motivated by my interest in testing my limits in trying out new programs and using my prior knowledge to adapt to new uses of programming knowledge in a new environment. As I've come to this point of this program at Vancouver community college, the skill that I'm reusing the most is adaptability. Every semester has been challenging in many ways but through each challenge, it builds my skills in learning new technologies and techniques. In this class I wanted to try to use the UWP C# techniques in creating a game.

This project gave me a chance to adapt the new integrations that Microsoft and visual studio have came out with this year and that's the Unity game engine that allowed developers to not only code in C# but utilize the ide in visual studio to allow for more seamless integrations for gaming development. As I applied the techniques using C# language, I've come to learn that the integrations between both the Unity game engine's IDE and visual studio allow for a various number of different methods and libraries to be utilized. I used Unity's game engine libraries to call on certain functions and objects to create parameters and methods to be used in the Flappy bird clone I made. Another great feature that I've also come to understand is that with Unity, creating a UWP application becomes seamless and when you fully implement the lines of code in Visual studio, in turn

it manipulates the functionality of the Unity's Game Engines IDE by allowing you to customize the settings that is used for your game. Further I will be diving more deeper into the UWP application that I've created with Visual studio and Unity.

Design Process

When I thought about creating a game, there were a few games that came to mind but when I thought of what games that are easy to understand and have a simple user interface which leads to easy gameplay, I had to think of mobile app games. There were quite a few different kinds of mobile games that have gain a lot of popularity, but the main genres were tower defense, a mobile clicker, and a side scroller game. When I thought of the side scroller that had those qualities in mind, the first concept of the game I wanted to emulate is Flappy Bird.

Flappy bird was initially released onto the internet in 2014 as a free online game that was popularized by an online forum called reddit. The game had a very simple user interface, and the main object of the game was to keep the animated flying fish up in the air by continuously tapping on your mouse button or space bar and navigate through the green pipes and not touch them. It was a side project created by Dong Nguyen, who was originally a game developer but had his game canceled in 2012.

He took the assets of the game he was creating in 2012 and created Flappy Bird. It took him several days to create the game and even though the mechanics

of the game weren't conventional, and some would say a bit tedious because of the controls, it created an addicting phenomenon that had an easy learning curve but a high degree of difficulty due to the mechanics. It earned 50,000$ from in-app advertisements and sales but became an international hit among mobile app users on IOS and android.

Beginning my development process, I had to think of designing the mechanics where I could keep the object which is the Flappy bird and creating the green pipes in a random order so the object would have to traverse the pipes, which in turn creates the difficulty. I also thought of when the object hits the pipes it should end the game for the player and restart. While the object goes through the pipes, I would also need to create an element to when the object passes through the pipes, the game would keep score and the player would gain a point every time that the object passed through the green pipes.

In unity you're able to create collision boxes around textures that are created which can interact with the object and create a Boolean which works when the object hits the object it stops the game and restarts. Another aspect of that I had to think about is when the object is side scrolling through the game, the background would have to continuously move with the object as having a static background wouldn't feel animated enough for a game. In turn I took my thought process behind my design and began to code out the implementation.

Features and specifications

Features

- Unity Game Engine
- Visual Studio IDE (Unity UWP, UWP application, C# Console Application)
- Game Assets (Animation Sprite Images, C# Scripts)

Specifications

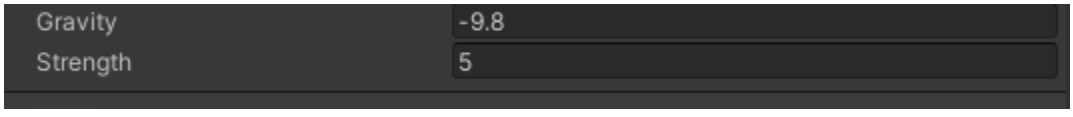
- Inheritance
- Classes (Public and Private)
- Objects
- Event Handlers
- Void C# Reference
- Parent and child references
- User inputs
- Responsive design
- Casting int to string
- For loops
- If / else statements

Implementation

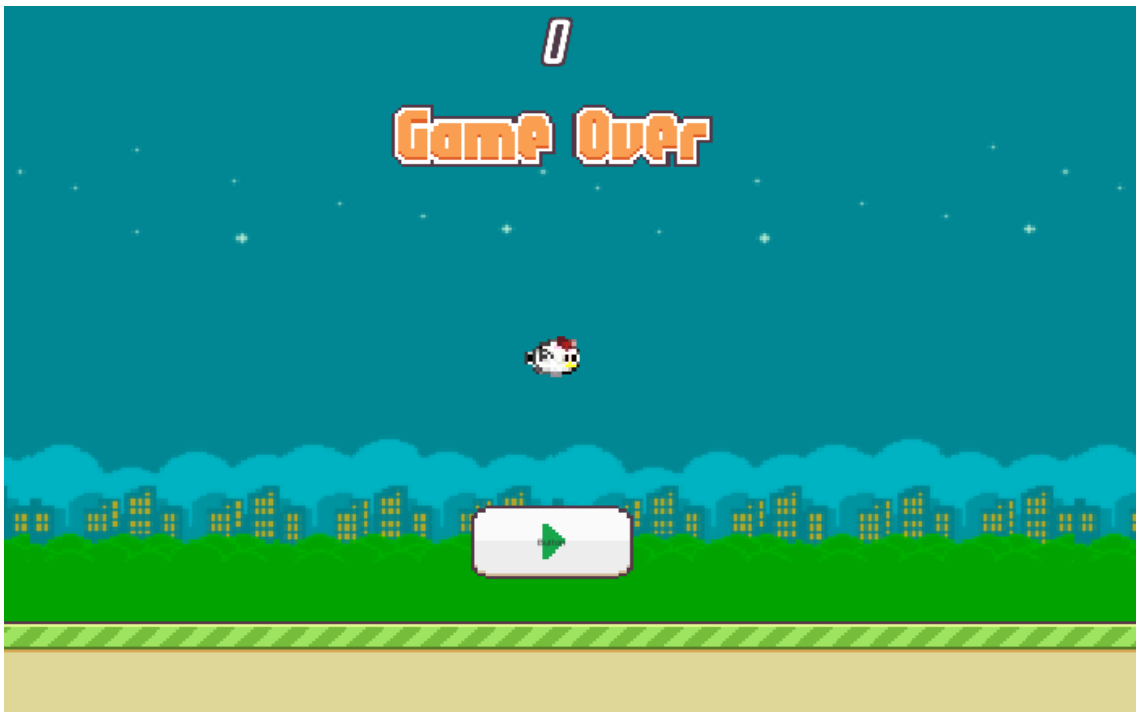
First, I had to start with the base implementations to get the game engine to recognize the player object and a feature that's implemented with Visual Studio and Unity's IDE is that when you create a variable using certain keywords, for example gravity and strength. Unity's IDE inherits the values and assign them inside the IDE for you to manipulate and use for your game functionality. I used -9.8f which is -9.8float which assigns a gravity that pulls down the character to the bottom of the screen and strength is a value set at 5float which is the speed that the player object gets pulled down to the bottom.

```
using UnityEngine;

1 reference
public class Player : MonoBehaviour
{

    2 references
    private SpriteRenderer spriteRenderer;
    2 references
    public Sprite[] sprites;
    4 references
    private int spriteIndex;
    5 references
    private Vector3 direction;

    1 reference
    public float gravity = -9.8f;

    2 references
    public float strength = 5f;
```

Inherited Values in Unity



I also created a class which is called player which is the parent class that manipulates the MonoBehaviour child class. The mono behaviour child is a class that gets manipulated by the C# script values in the parent class PLAYER and changes the IDES interface with the values inputted into the C# script. I had to create a Sprite index array that held the images of the cartoon animations which are inputted in the game such as the chicken, the city night background, the green pipes, and the games text. SpiritRenderer is a private variable that initializes the sprites to show up on the UWP interface. Vector3 is an inbuilt variable that handles the direction in which the player object goes. Another method that I involved from my knowledge that I've gained in class is the private and public classes. If the class is private, unity can't access the information provided in the script but if the class is public then it can be accessed by unity's IDE and you're able to use the values assigned which is shown in the example of Gravity and Strength.



I created event handlers that were triggered by user's input. The Update function kept checking for users input when they hit the space bar or the left mouse button which is set by the parameter zero. Every time that the buttons were clicked a event handler would check to see if it was pressed and when it was pressed it would jump the player object up in the game. The input.touchCount would continuously get triggered by the user input and if the user input triggered then the direction of the player object would keep on going up. Unity has a built-in variable called deltaTime which just keeps the player object going at the same speed as the frames per second and not any faster.

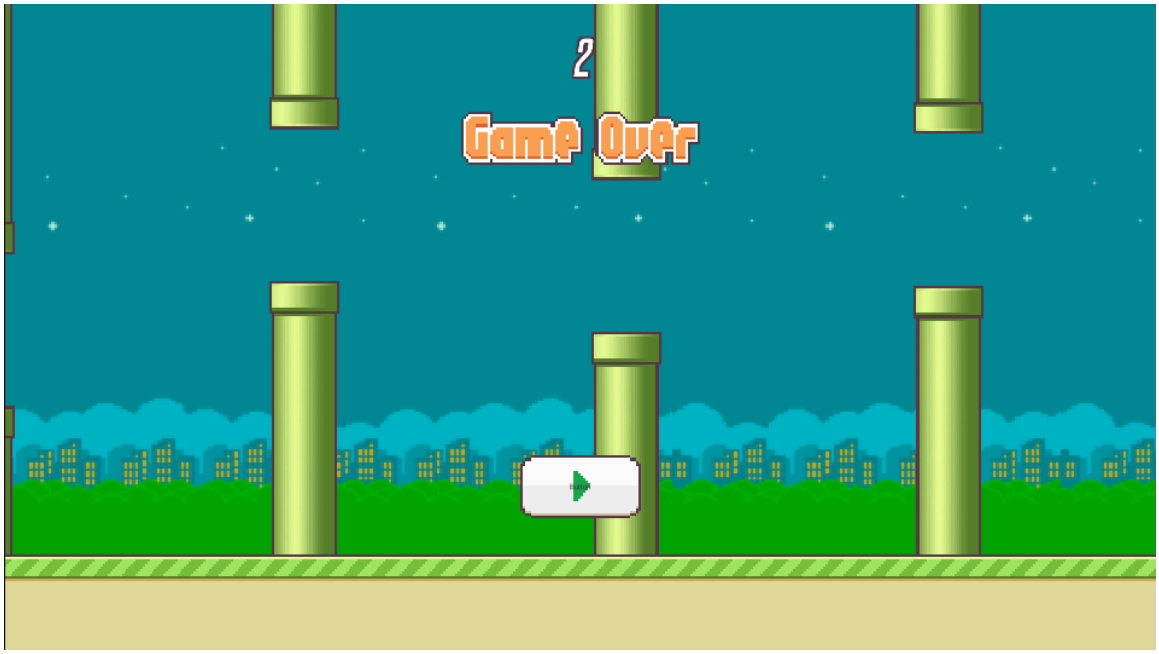
```
// Update function for USER INPUT
0 references
private void Update()
{
    // User input for player object to jump up by either pression space bar or left mouse button
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0)) {
        direction = Vector3.up * strength;
    }

    if (Input.touchCount > 0)
    {
        Touch touch = Input.GetTouch(0);

        if (touch.phase == TouchPhase.Began) {
            direction = Vector3.up * strength;
        }
    }

    direction.y += gravity * Time.deltaTime;
    transform.position += direction * Time.deltaTime;
}
```

Another event handler that I've used is when the game an obstacle is hit it calls a function called OnTriggerEnter2D which if the event handler is triggered when the player object hits an obstacle such as the green pipe or ground then stops the game and when the player object goes through the pipes and doesn't trigger the obstacle event handler then it would call the increases score function which kept the score of the game.

```
// When player object collides with other objects it triggers an event handler
0 references
private void OnTriggerEnter2D(Collider2D other)
{
    // if player object hits obstacle then game ends
    if (other.gameObject.tag == "Obstacle") {
        FindObjectOfType<GameManager>().GameOver();
    // if player object doesn't hit obstacle then a score is kept
    } else if (other.gameObject.tag == "Scoring") {
        FindObjectOfType<GameManager>().IncreaseScore();
    }
}
```

I also utilized a for loop for the green pipes to randomize the pipes as the game plays to change the min and max height as the player goes through the game but as well when the green pipes hit the left edge of the screen it destroys the green pipe but continuously spawns new randomized green pipes from the right of the screen.

```
public void Play()
{
    // intializes games score to 0
    score = 0;
    // casts score text from int to string
    scoreText.text = score.ToString();

    // play button and game over text are set to false to not show up during gameplay
    playButton.SetActive(false);
    gameOver.SetActive(false);

    // The timescale starts the game and player enabled checks to see if player has started game
    Time.timeScale = 1f;
    player.enabled = true;

    // Array that creates random green pipes
    Pipes[] pipes = FindObjectsOfType<Pipes>();

    for(int i = 0; i <pipes.Length; i++){
        Destroy(pipes[i].gameObject);
    }
}
```



This image below shows how the green pipes are created when the game starts and ends and how the spawn function creates the green pipes as the game goes on

```
// when game starts it spawns the green pipes repeadtly
0 references
private void OnEnable()
{
    InvokeRepeating(nameof(Spawn), spawnRate, spawnRate);
}

// when game stops On Disable is called to clear the repeating green pipes
0 references
private void OnDisable()
{
    CancelInvoke(nameof(Spawn));
}

// Spawn randomizes the values min/max height of the green pipes and changes the position with Quaternion.identity and transform.position
2 references
private void Spawn()
{
    GameObject pipes = Instantiate(prefab, transform.position, Quaternion.identity);
    pipes.transform.position += Vector3.up * Random.Range(minHeight, maxHeight);
}
```

The image below shows how I utilized casting integers to a string utilizing the score text in the game.

```
// Increase score function increments score text and casts scoreText int to string
1 reference
public void IncreaseScore()
{
    score++;
    scoreText.text = score.ToString();
}
```

Conclusion

NEW CONCEPTS LEARNED

I've come to learn more about how to utilize C# with different IDES and how much integration that Visual studio can utilize with a game engine such as unity. I was able to apply a various number of concepts that I've learned in class and as well as how diverse UWP can be used other than just using Visual studio. Unity was able to create a UWP application in a different interface and I was able to create a fully functional game that involves a lot of different aspects that utilizes the same tools in UWP console applications. I was able to incorporate text boxes in a different manner and manipulating those text boxes to change while incorporating for loops. I was also able to learn how to utilize event handlers such as on click and user input that I was taught in class to manipulate different actions that happen in my game. Overall, it was a very worthwhile experience to see visually how the concepts that I was taught in class can be turned into a very tangible product.