Aaron Fox
CECS 528
Dr. Chang

CECS 528 Homework Assignment 1

1. *Study the PowerPoints, Game_history and do the following problems:*
   (a) *List two of the significant impacts of the Spacewar (1961, Stevel Russell) computer game.*

   Spacewar is widely considered the first “true” video game, for it is the first known video game to be played on many different computer systems rather than just on one computer system that the game was specifically designed for as many ports and emulation of the game were created. It was also hugely influential in other impactful video games in the coin-operated field, as its concepts can be seen in later games such as Asteroids and Computer Space as discussed in class. It was also influential in showing the power of video games to simulate real Newtonian physics with the gravity wells contained within the game.

   (b) *List two of the significant impacts of the Colossal Cave (1975, Will Crowther) computer game*.

   The *Colossal Cave* is known as the first interactive fiction game and first text adventure game to gain wide popularity in the video game world. It was very innovative for being the first popular game of its genre, leading to even more popular games such as *Zork*. Its lasting impact can be found in the video game genre of adventure games, as its fun text adventures can now be graphically realized using game engines such as Unity and Unreal. In addition to impacting the future of adventure games, *Colossal Cave* also had a large impact on the RPG genre of video games of the future.

   (c) *What is the first commercially successful coin-operated arcade game and why is it not Spacewar?*

   While Spacewar helped inspire and create many games behind the coin-operated arcade game industry, it was not the first commercially successful coin-operated arcade game as it focuses on being built for large computer systems such as the DEC PDP-1 and other similar ported systems. Computer Space, a derivative of Spacewar, also failed to be a commercial success because of its complex instructions and poor marketing. The title of the first commercially successful coin-operated arcade game belongs to ***Pong***, which was simple and fun for consumers to play, making it commercially successful to all.

   (d) *What is the video game considered to be the main cause of the game industry slump or crisis in 1983-1984?*

The video game often blame for the game industry slump in 1983-1983 is *E.T. the Extra Terrestrial*, which Atari expected to sell well, but the game was rushed in about 5 weeks, leading to the game being heavily criticized and Atari burying many of the games in New Mexico, costing the company and the video game industry a lot of money, wasted time, and much mocking.

2. Play the games Quick Draw and TotemGoals and write a short comment about these two games including your gameplay experiences.

Quick Draw: Initially, I found the gameplay rather difficult even on the easiest setting even though the rules were easy to understand and follow. But, after playing a few rounds on easy I was able to progress to medium and then to hard. I could not best the hardest difficulty no matter how much I tried, so I would encourage the game designer to possibly make the difficulties a little easier. Also, the random choices of the cards would often choose the same card several times in a row while I was playing; I would encourage the game designer to prevent this from happening. The music was very fast-paced and it made my heart race and made me get more excited and eager to select the correct card. Overall, this game is a very fast-paced, exciting, and difficult game that is fun to play but not very extensible (unlike TotemGoals below.)

TotemGoals: This 2D puzzle platformer was easy to pick up and learn the controls. It is a genuinely fun game that takes a bit of thought to defeat the levels, forcing intuition for the player to think about the capabilities of each animal that the totem animal represents. The background music and sound effects (especially the buzzing “incorrect” noise that signalled you weren’t supposed to perform an action) were enjoyable and carefree, and the instructions at the beginning were clear and concise. The one and only level of the game was creative and fun, requiring the player to think creatively and outside the box. I only wish there were more levels to play, for the game is genuinely a lot of fun and is very innovative.

3. Look up Unity documentation to learn how to use the transform class. It is important to understand how to use the transform class to use Unity effectively. Create a Unity project and create a simple scene of moving objects (e.g. cubes) of varied sizes as discussed in class. Make the objects move in an interesting formation or pattern (e.g. racing, dancing, etc.)

After reading the Unity docs on the transform class, a simple scene of several moving spheres of varied sizes were placed on a plane. An overall formation of six different spheres was created such that they never touch and that they always rotate around the same point at the same speed in sync. They thus always line up. Three of the green spheres rotated clockwise while the three blue spheres rotated counter-clockwise.

This created a neat visual effect of the six varied-sized spheres as they rotated around the origin of the world space and looked as if they would hit the other spheres in the formation spiral, but yet they never do hit.

transform.RotateAround was the main transform method used to create this formation.

The scripts attached to the spheres, depending on if the spheres spiralled clockwise or counter-clockwise was:

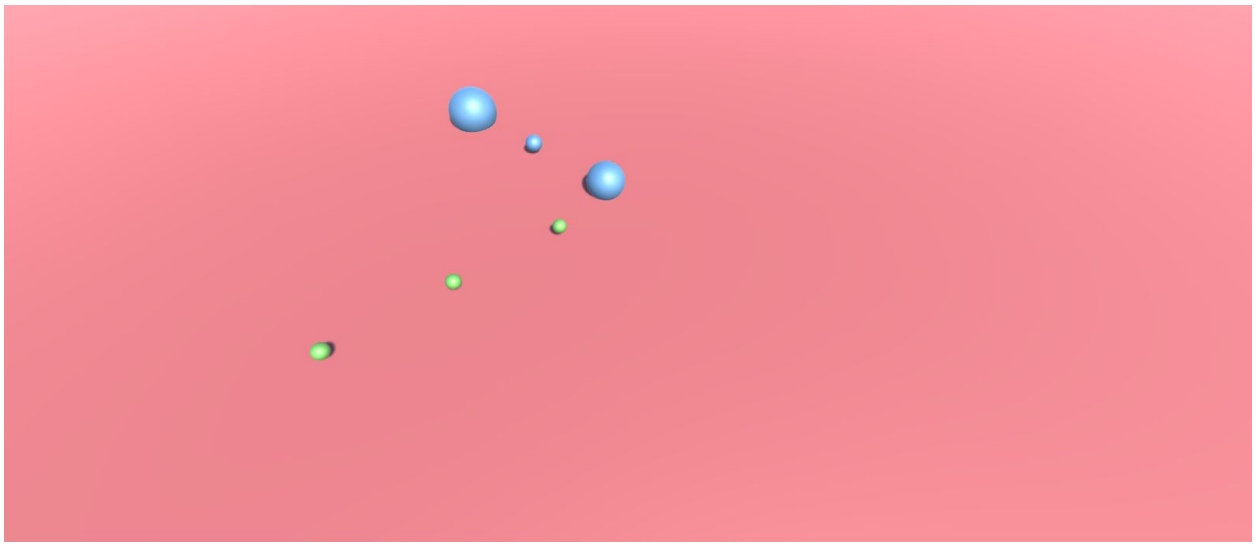
Spiral.cs:

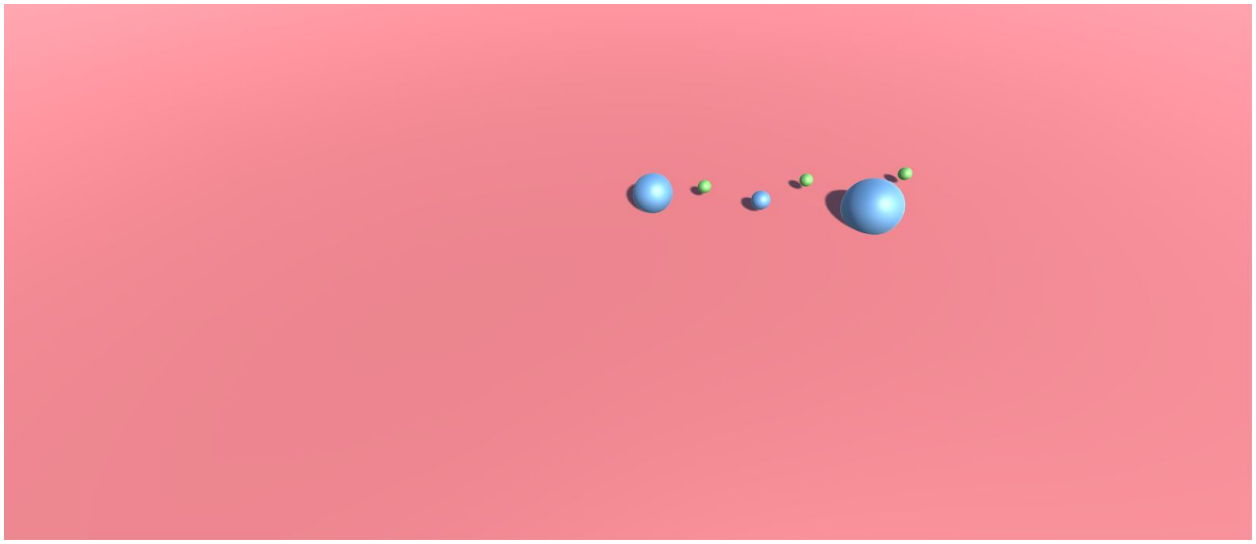
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Spiral : MonoBehaviour
{
    private Vector3 origin = new Vector3(0.0f, 0.0f, 0.0f);

    // Update is called once per frame
    void Update()
    {
        // Spin the object around the world origin
        transform.RotateAround(origin, new Vector3(0, 1, 0), 100 * Time.deltaTime);
    }
}
```

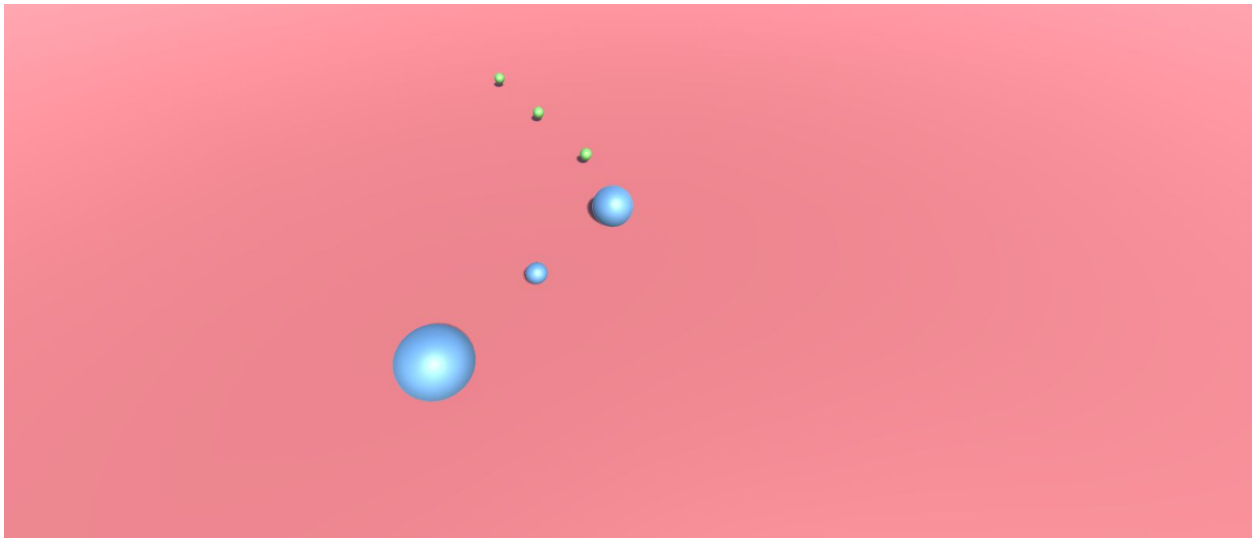
ReverseSpiral.cs:

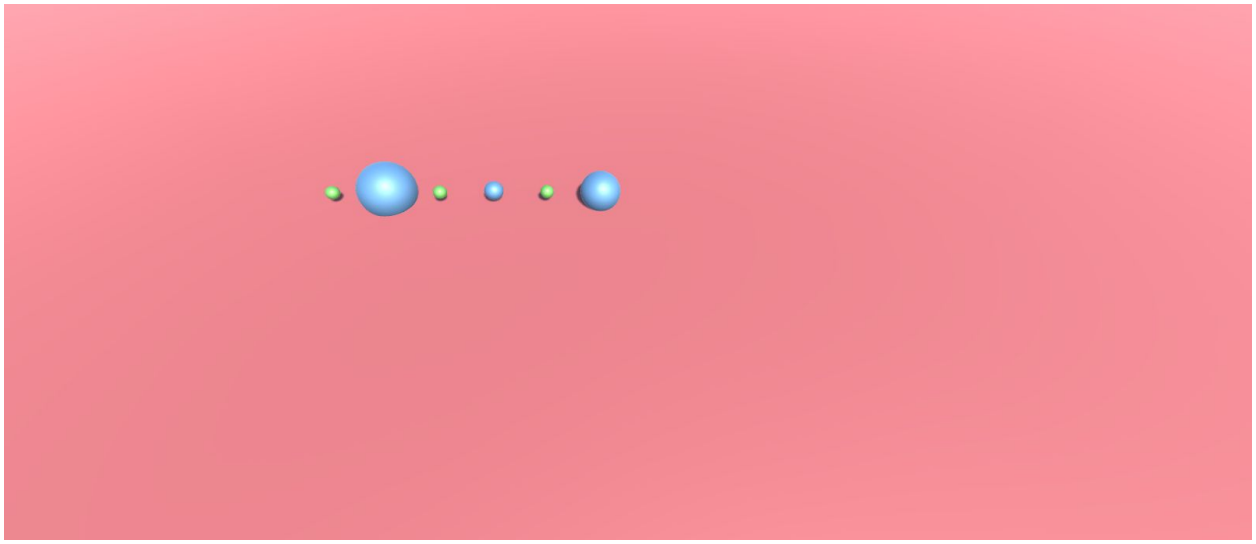
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReverseSpiral : MonoBehaviour
{
    private Vector3 origin = new Vector3(0.0f, 0.0f, 0.0f);

    // Update is called once per frame
    void Update()
    {
        // Spin the object around the world origin
        transform.RotateAround(origin, new Vector3(0, 1, 0), -100 * Time.deltaTime);
    }
}
```

Pictures documenting the spiral formations are included below:









4. In this problem you will use the PinBall_MusicNotes Unity project.

(a) Which game object has the C# script PinballGame.cs component?

The **Main Camera** game object contains the C# script PinballGame.cs.

(b) Open the C# script PinballGame.cs in the text editor (e.g. VS 2017) and edit the script as follows:

```
void Awake()
{
      SP = this;
      gameState = PinballGameState.playing;
      Time.timeScale = 2.0f;
      SpawnBall();
}
```

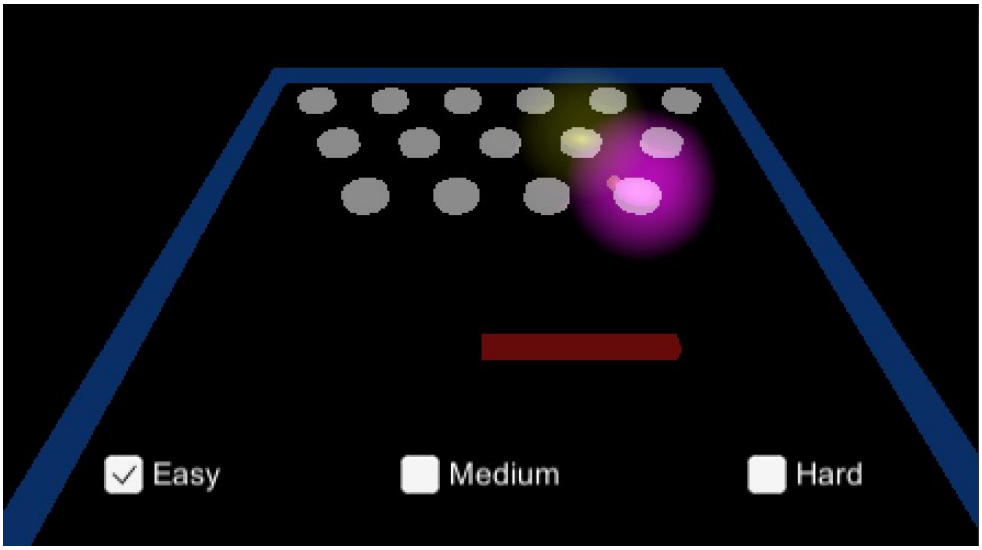
Save the Script and play the game. Experiment with different values set to Time.timeScale and describe the changes these different Time.timeScale values made to the game play.

When changing Time.timeScale to 2.0f from 1.0f, the game's speed and time was twice as fast. This meant that the player's paddle moved twice as fast as before, the ball moved twice as fast as before, and all lighting effects of the cylinders were twice as short. The sound, however, was at the same timeScale as the 1.0f timescale. It looks like the script must adjust audio.pitch to equal Time.timeScale for the audio to be changed as well. After a Time.timeScale of about 18.0f, the gameplay was too fast for my eyes to perceive, and I immediately lost. In the reverse side, a timeScale any less than 0.8f became unbearably slow, and I became very impatient having to wait for the ball to

come down so that I could use my slow-moving paddle to hit the ball. This illustrates the importance of having the proper timescale in games so that they are fun to play and not too difficult.

(c) *Based on (b), design and implement a scheme to vary gameplay difficulties. Describe your design and implementation. Moreover, you can base on the total time (in seconds) the player can play until game over as the scoring systems.*

The scheme I created allowed for the player to toggle between three game difficulties: easy, medium, and hard. In the easy game difficulty, Time.timeScale is set to the normal 1.0f. In the medium game difficulty, Time.timeScale is set to 2.5f. In the hard game difficulty, Time.timeScale is set to 4.0f. For the game difficulties, I used three Toggle buttons that acted like radio buttons in a Toggle Group so that one and only one difficulty had to be selected at all times. I then added the relative amount of time, based on Time.deltaTime in the FixedUpdate (which would increase faster for a higher difficulty level) to the overall final score of the player so that the length of the player's playtime led to a higher score. An example GUI of the player's ability to choose between the difficulties is seen below:



The current Toggle difficulty button was found using System.Linq's's ElementAt function to along with the ToggleGroup methods, like:

```
Toggle active = toggleGroup.ActiveToggles().ElementAt(0);
```

Since there was only ever one and exactly one toggle selected at all times, using `ElementAt(0)` always selected the currently active game difficulty. In the added

`FixedUpdate` function of the PinballGame.cs script, an additional score was added to the player's overall score at the end based on how much time passed relative to the game difficulty. This was accomplished by

```
totalTime = totalTime + 1 * Time.deltaTime;
```

By multiplying the score variable (here, it's simply 1) by Time.deltaTime inside the FixedUpdate function for physics allowed the player to earn a higher score based on their difficulty. For example, if the player toggled the more difficult "Hard" setting, the timescale would be increased and so Time.deltaTime would increase, meaning the player would earn more points. Finally, the final score that the player earned was set by

```
if (hasClickedTryAgainYet == false)
      {
          score = score + (int)totalTime;
          hasClickedTryAgainYet = true;
      }
```

This ensured that the player's score was only added once when the player lost the ball instead of continually adding to the score when PinballGameState was in the losing state. The boolean `hasClickedTryAgainYet` is set to `false` once the player clicks the Try Again button.

This added more difficulties and fun to the PinballMusicNotes game, and I know that I personally enjoyed playing the game much more after adding these tweaks to the game. The hard difficulty made the ball go very quickly and thus made the game much more difficult. The medium difficulty made the ball go a little slower than it did in the hard mode, and so on...

Appendix

The entire edited PinballGame.cs script is here:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Linq;

public enum PinballGameState {playing, won, lost};

public class PinballGame : MonoBehaviour
{
    public static PinballGame SP;
    public ToggleGroup toggleGroup;

    public Transform ballPrefab;
```

```
        private int score;
    private PinballGameState gameState;

    private float totalTime = 0;

    private bool hasClickedTryAgainYet = false;

    void Awake()
    {
        SP = this;
        gameState = PinballGameState.playing;
        Time.timeScale = 1.0f; // was 1.0f
        SpawnBall();
    }

    void SpawnBall()
    {
        Instantiate(ballPrefab, new Vector3(0f, 1.0f , 4.75f), Quaternion.identity);
    }

    public void FixedUpdate()
    {
        // Add to total time with a number that is relative to deltaTime so the player earns more
points for
        // more difficult play levels
        totalTime = totalTime + 1 * Time.deltaTime;
        Toggle active = toggleGroup.ActiveToggles().ElementAt(0);

        //Debug.Log("//" + active.ToString().Substring(0, 1) + "//");
        string letter = active.ToString().Substring(0, 1);
        if (letter == "E")
        {
            Time.timeScale = 1.0f;
        }
        else if (letter == "M")
        {
            Time.timeScale = 2.5f;
        }
        else
        {
            Time.timeScale = 4.0f;
        }
    }

    void OnGUI()
    {

        GUILayout.Space(10);
        //GUILayout.Label("  Score: " + score.ToString());

        if (gameState == PinballGameState.lost)
        {
            GUILayout.Label("You Lost!");
            GUILayout.Label("  Score: " + score.ToString());
            if (hasClickedTryAgainYet == false)
            {
                score = score + (int)totalTime;
```

```
                hasClickedTryAgainYet = true;
            }
            if (GUILayout.Button("Try again"))
            {
                hasClickedTryAgainYet = false;
                Application.LoadLevel(Application.loadedLevel);
            }
            //gameState = PinballGameState.playing;
        }
        else if (gameState == PinballGameState.won)
        {
            GUILayout.Label("You won!");
            if (GUILayout.Button("Play again"))
            {
                Application.LoadLevel(Application.loadedLevel);
            }
        }
    }

    public void HitBlock()
    {
                score += 10;
    }

    public void WonGame()
    {
        Time.timeScale = 0.0f; //Pause game
        gameState = PinballGameState.won;
    }

    public void LostBall()
    {
        int ballsLeft = GameObject.FindGameObjectsWithTag("Player").Length;
        if(ballsLeft<=1)
        {
            //Was the last ball..
            SetGameOver();
        }
    }

    public void SetGameOver()
    {
        Time.timeScale = 0.0f; //Pause game
        gameState = PinballGameState.lost;
    }
}
```