PONG clone development evaluation

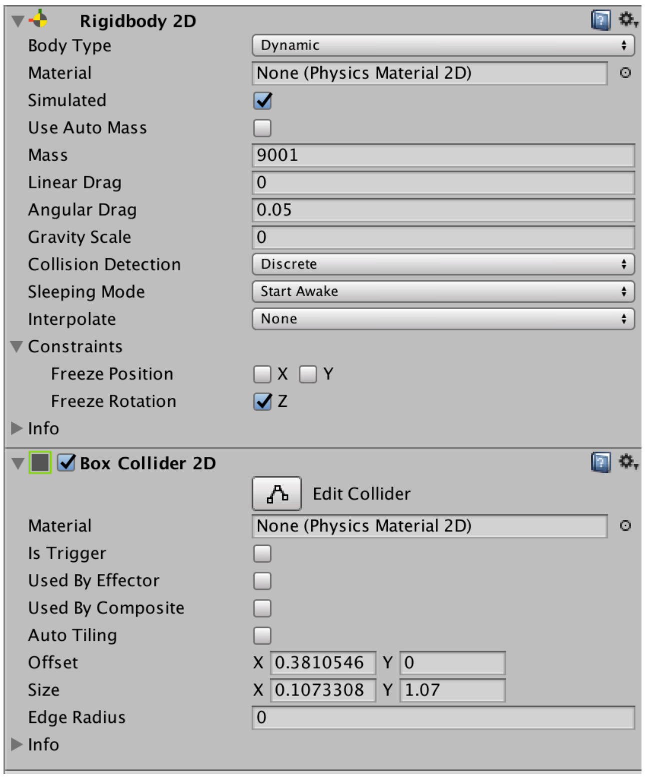
Introduction

As mentioned in the design documents, the aim was to create a clone of PONG which, for the most part, would work almost like the original. This document would be outlining the development process which was taken while the clone was being developed, whether the approaches taken for implementing the required elements and functionality were efficient or not, and how, for any project which would be created in the future, the overall process could be improved upon.

Movement

Initially, creating the movement was somewhat tricky, as a tutorial was followed which made the collision detection through the C# programming language, which made the whole process more complicated and time consuming than it should.

Eventually another tutorial was followed, which implemented only the movements through C# and a RigidBody2D component while the collision was done with a Box Collider 2D component instead, which saved a hefty amount of time.



Rigidbody 2D

Body Type	Dynamic
Material	None (Physics Material 2D)
Simulated	✓
Use Auto Mass	☐
Mass	9001
Linear Drag	0
Angular Drag	0.05
Gravity Scale	0
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
Constraints	
Freeze Position	☐ X ☐ Y
Freeze Rotation	✓ Z
Info	

✓ Box Collider 2D

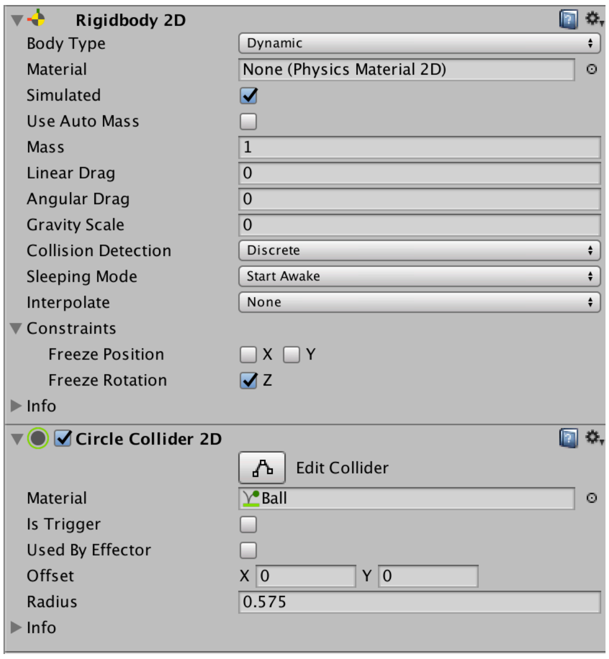
	Edit Collider
Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0.3810546 Y 0
Size	X 0.1073308 Y 1.07
Edge Radius	0
Info	

Figure 1 - Using a RigidBody2D and Box Collider 2D component for the player paddles

Ball Movements

Just like the paddles, the ball's collision with the walls as well as the player paddles was initially done through code, which caused the ball to bounce back incorrectly when colliding with the paddle, as the ball would stop on collision and move again only if the player would move their paddle.

Eventually the collision was handled through a Box Collider 2D component (Later on this was swapped in favour of a Circle Collider object as the ball's sprite was later edited in order to resemble a ball rather than a square) and the Ball's physics were handled through a Unity Physics Material, where a 'bounciness' value was given in order to make the ball bounce off the walls and paddles upon making contact.

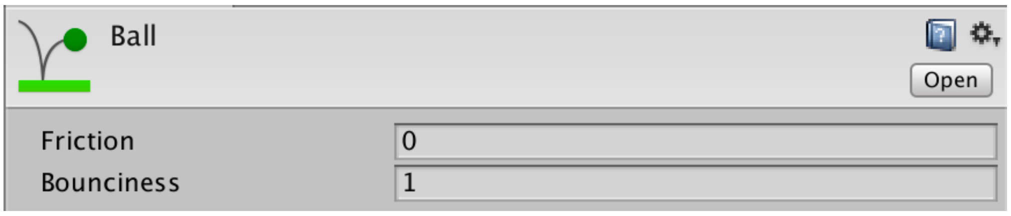

Rigidbody 2D

Body Type	Dynamic
Material	None (Physics Material 2D)
Simulated	✓
Use Auto Mass	☐
Mass	1
Linear Drag	0
Angular Drag	0
Gravity Scale	0
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
Constraints	
Freeze Position	☐ X ☐ Y
Freeze Rotation	✓ Z
Info	

Circle Collider 2D

	Edit Collider
Material	Ball
Is Trigger	☐
Used By Effector	☐
Offset	X 0 Y 0
Radius	0.575
Info	

Figure 2 - Using a RigidBody2D and Box Collider 2D component for the Ball


Ball

Friction	0
Bounciness	1

Figure 3 - Ball physics implemented through a Physics material

Scoring System

While coding the scoring system, one of the tutorials which was followed implemented the scoring functionality in a way that when the ball reaches a certain coordinate, the score would go up, however, during practise this method proved to be ineffective, as part of the code required the Ball game object to be found by the script, which said task failed to be completed.

Eventually, another method was implemented: where two walls were created outside the camera's range, and in one of the existing script files, the necessary commands were given to increment the score of the players by 1 should the ball make contact with one of them.

Despite being a much more efficient method to implement, the fact that an existing script (The Ball script in this case) was used in order to apply the incrementing command may cause some disorganisation when searching for code related to either the Ball or Score, due to the functionality of the two areas being in the same C# file.

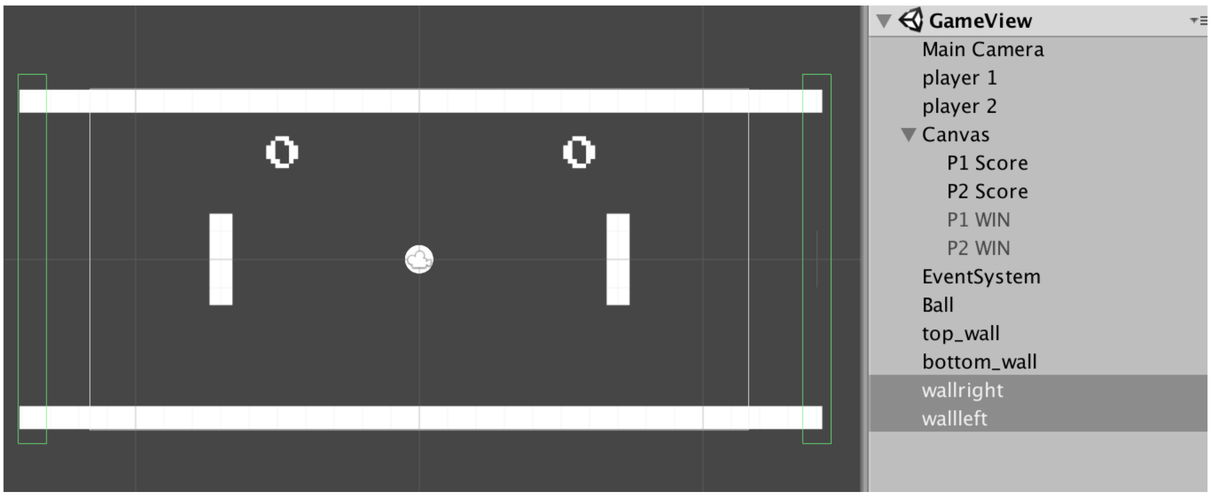
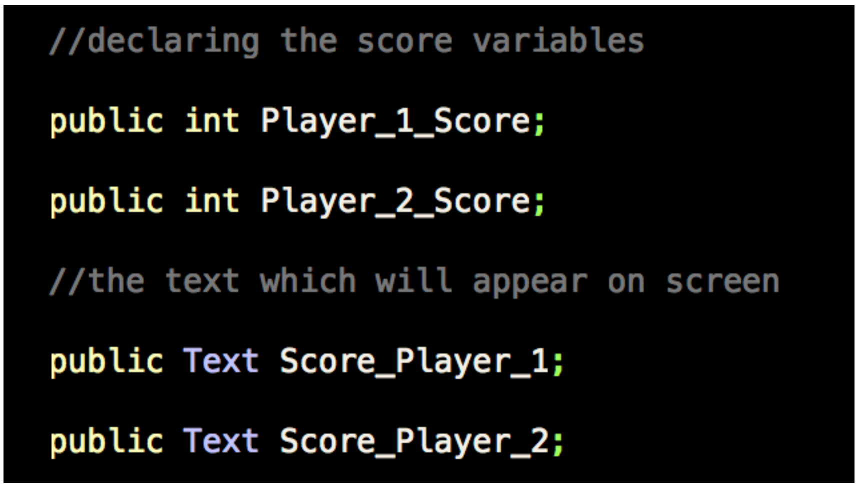
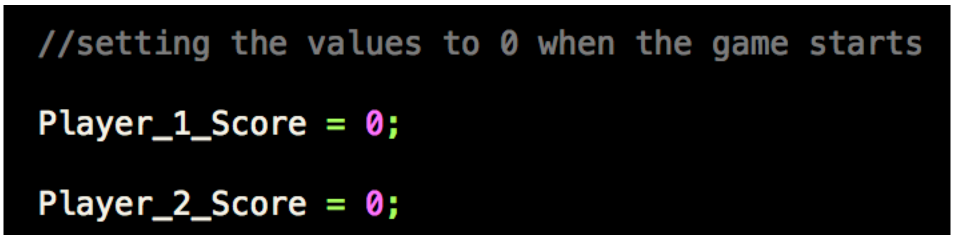


Figure 4 - Creating the outside walls



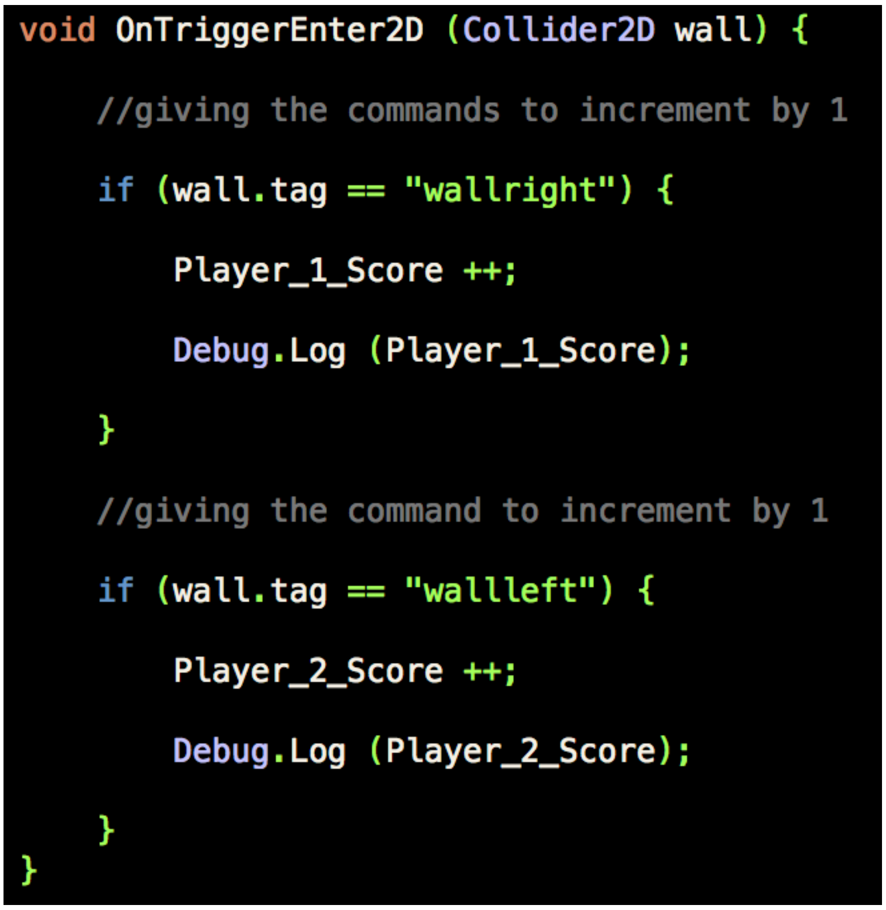
```
//declaring the score variables

public int Player_1_Score;

public int Player_2_Score;

//the text which will appear on screen

public Text Score_Player_1;

public Text Score_Player_2;
```

Figure 5 - Declaring the variables



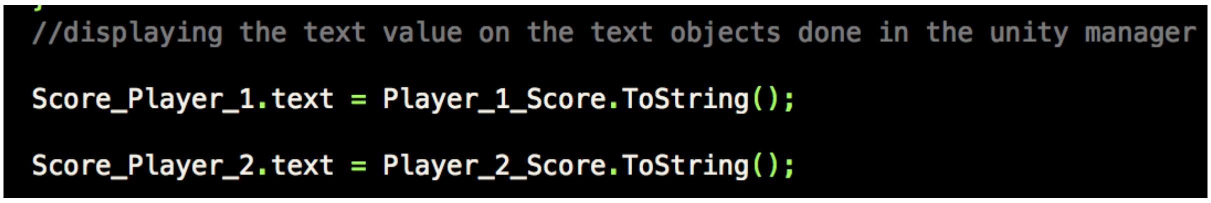
```
//setting the values to 0 when the game starts

Player_1_Score = 0;

Player_2_Score = 0;
```

Figure 6 - Setting the values to 0 upon the game's startup



```
void OnTriggerEnter2D (Collider2D wall) {

    //giving the commands to increment by 1

    if (wall.tag == "wallright") {

        Player_1_Score ++;

        Debug.Log (Player_1_Score);

    }

    //giving the command to increment by 1

    if (wall.tag == "wallleft") {

        Player_2_Score ++;

        Debug.Log (Player_2_Score);

    }
}
```

Figure 7 - Giving the command to increment the score's value



```
//displaying the text value on the text objects done in the unity manager

Score_Player_1.text = Player_1_Score.ToString();

Score_Player_2.text = Player_2_Score.ToString();
```

Figure 8 - Converting the values to text strings so as to be displayed using text objects in-game

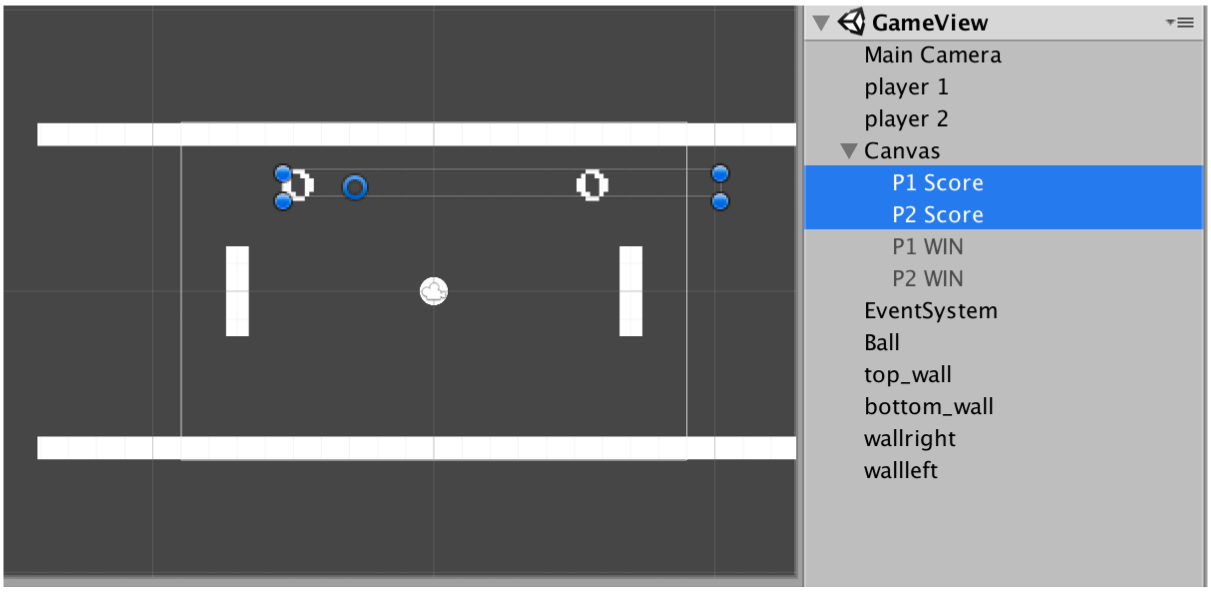
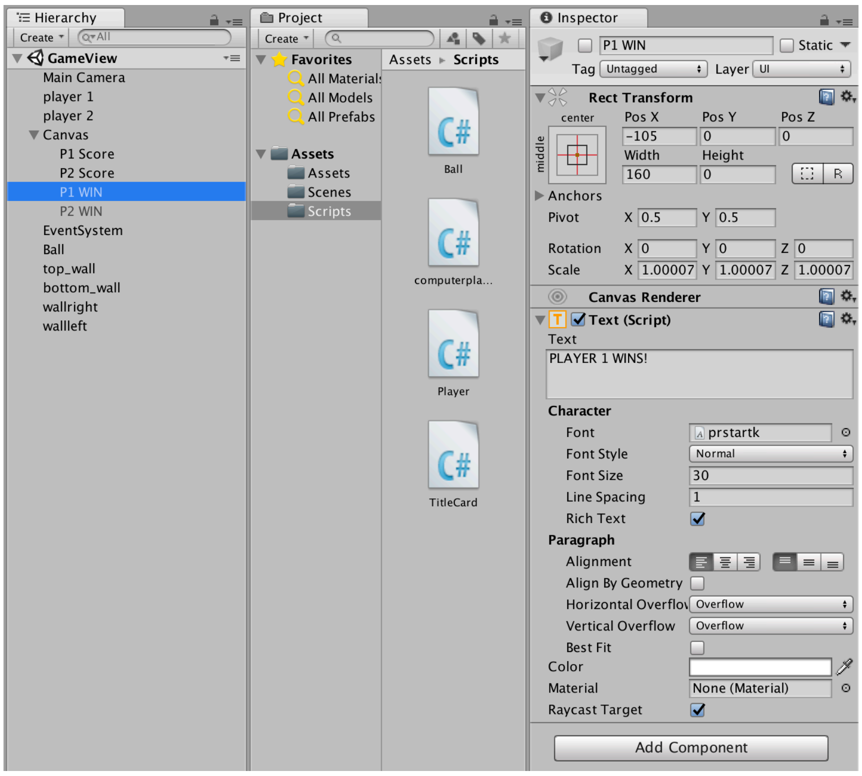


Figure 9 - Text objects used for displaying the score

Player Winning Messages

Now that a basic idea has been achieved in understanding how to change text after certain action have been performed, giving the command to display the text which declares either Player 1 or 2 as the winner was not a very difficult task to perform.

Just like with the scores which are displayed in-game, two text objects were created and unchecked, in order to hide them from the scene.



P1 WIN — Static
Tag: Untagged — Layer: UI

Rect Transform
center / middle
Pos X: -105 — Pos Y: 0 — Pos Z: 0
Width: 160 — Height: 0
Anchors
Pivot: X 0.5, Y 0.5
Rotation: X 0, Y 0, Z 0
Scale: X 1.00007, Y 1.00007, Z 1.00007

Canvas Renderer

Text (Script)
Text: PLAYER 1 WINS!
Character
Font: prstartk
Font Style: Normal
Font Size: 30
Line Spacing: 1
Rich Text: ✓
Paragraph
Alignment
Align By Geometry: ☐
Horizontal Overflow: Overflow
Vertical Overflow: Overflow
Best Fit: ☐
Color
Material: None (Material)
Raycast Target: ✓

Add Component

Figure 10 - Creating the text objects

Next, two variables were created for the text objects

```
//calling the Player 1 Wins text (text value wont allow set.active so use GameObject)
public GameObject P1_Win;
public GameObject P2_Win;
```

Figure 11 - The variables for the Player scores

Next, the text objects were turned off by default upon starting up the game regardless if in the scene view they are shown or not.

```
P1_Win.SetActive (false);
P2_Win.SetActive (false);
```

Figure 12 - Turning the text off upon start-up

Finally, the commands were given to display the text when either Player 1 or Player 2’s scores are equal to 10.

```
  if (Player_1_Score >= 10) {

      P1_Win.SetActive (true);
```

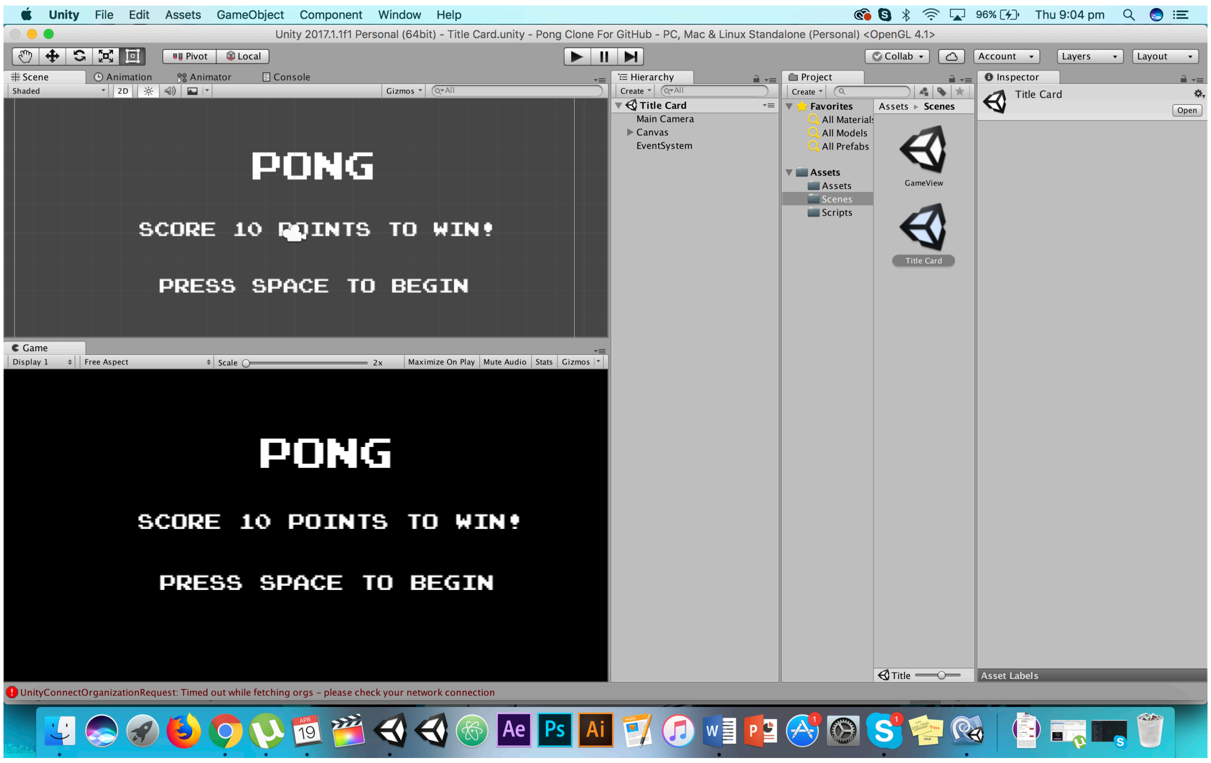
Figure 13 - Activating the Player 1 wins text when their score is at 10

```
if (Player_2_Score >= 10) {

    P2_Win.SetActive (true);
```

Figure 14 - Activating the Player 2 wins text when their score is at 10

Creating the Main Menu

In order to create the Main Menu, a new scene was created with the game name and instructions added using Text objects.


Unity 2017.1.1f1 Personal (64bit) - Title Card.unity - Pong Clone For GitHub - PC, Mac & Linux Standalone (Personal) <OpenGL 4.1>

PONG

SCORE 10 POINTS TO WIN!

PRESS SPACE TO BEGIN

UnityConnectOrganizationRequest: Timed out while fetching orgs – please check your network connection

Figure 15 - New scene with the game title and instructions

A new C# script file was created called *TitleCard* and the following lines of code were added, which would allow the player to go to the main game by pressing the SPACE key on their keyboard:

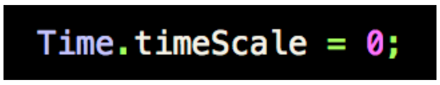
```
void Update() {

    if (Input.GetKey (KeyCode.Space)) {

        SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex + 1);

    }

}
```

Figure 16 - Pressing the SPACE key will take the player to the main game

Freezing Game Play & Restarting the game

After one of the Players reaches 10 points, the intention was to display the text declaring the player who scored 10 points the winner, with the text initially intended to be shown on a separate screen, then on said screen, allow the player to return to the main menu upon pressing the SPACE key. However, upon considering this initial idea, the decision was made to having the text appear on screen instead due to the feeling that having the text displayed on a separate screen would look slightly out of place.

In order to freeze the gameplay when the text appears, the following line was code was added:

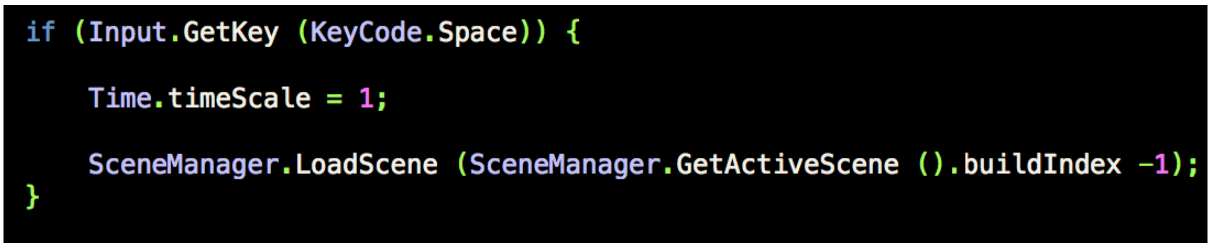


```
Time.timeScale = 0;
```

Figure 17 - Time.timeScale is used to freeze the in-game elements

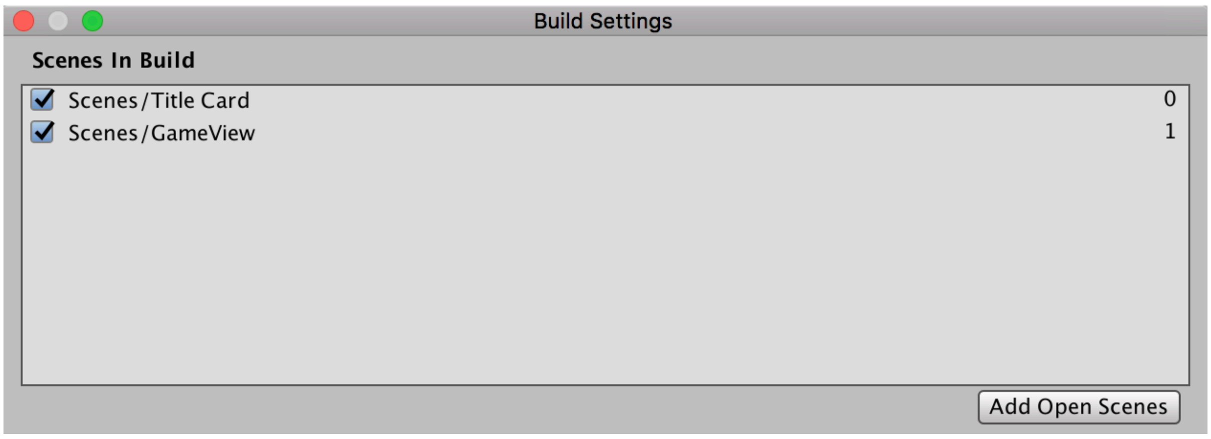
To restart the game, as well as unfreeze the in game elements, the following lines wee added:

Time.timeScale = 1 unfreezes the in-game elements while *SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex -1);* goes back to the previous scene in the build which is the main menu.



```
if (Input.GetKey (KeyCode.Space)) {

    Time.timeScale = 1;

    SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex -1);
}
```

Figure 18 - Setting the value of *Time.timeScale* to 1 unfreezes the elements



Build Settings

Scenes In Build

✓ Scenes/Title Card	0
✓ Scenes/GameView	1

Figure 19 - Scenes in the Unity build

For Future Projects

For future projects which would be created in Unity, one thing which would be noted is to make use of Unity’s built in tools and components rather than doing literally everything via code in order to simplify the overall development process as well as save enough time in order to focus on the other required areas of the game.