2018

Fresh Avocado Development Evaluation

JEANLUKE ZAMMIT GUGLIELMI, JAMES DIMECH, ANDREW WORLEY

BA IN INTERATIVE MEDIA LEVEL 6 YEAR 2

Fresh Avocado Development Evaluation

Introduction

As mentioned in the design documents, the aim was to create an original game where the player would collect batteries in order to advance through the game. This document would be outlining the development process which was taken while the game was being developed, whether the approaches taken for implementing the required elements and functionality were efficient or not, any problems which were encountered and how could the game has been improved.

Initial ideas

The initial ideas for the game were to make a 2D platformer game where the player would navigate through 2 levels by moving and jumping, where the goal would be to collect a certain amount of a specific key item (batteries) in order to unlock the goal and gain the ability to progress to the next level. Said goal is a fridge which requires a certain amount of batteries to be powered up.

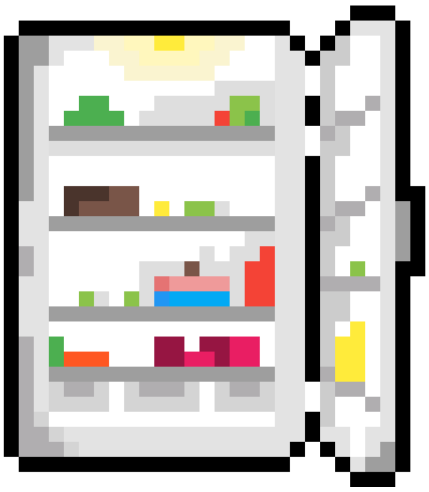


Figure 1 - The fridge

The player would have to avoid various enemies, in the form of bumble bees and snails as well as keep their eyes peeled on a time limit in the form of a day-night clock which would instantly kill the player should it reach Day Time.

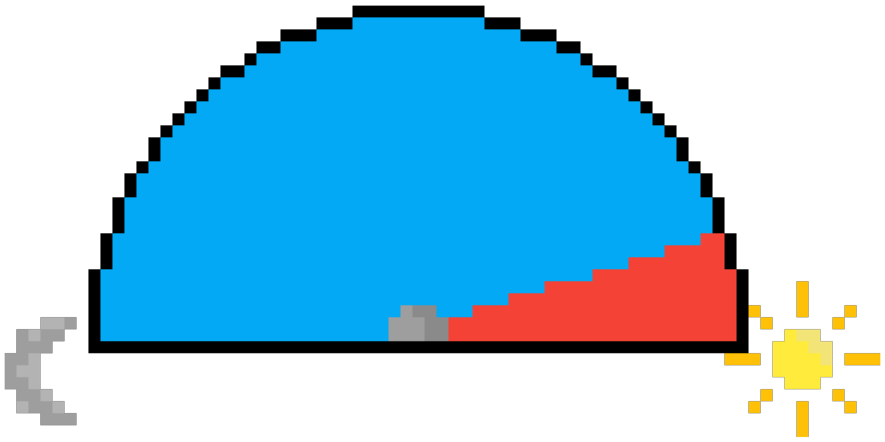


Figure 2 – Timer

The health system would consist of a health bar which would decrease in value after the player would take a hit, where once the bar drains completely a life would be deducted from a life counter.

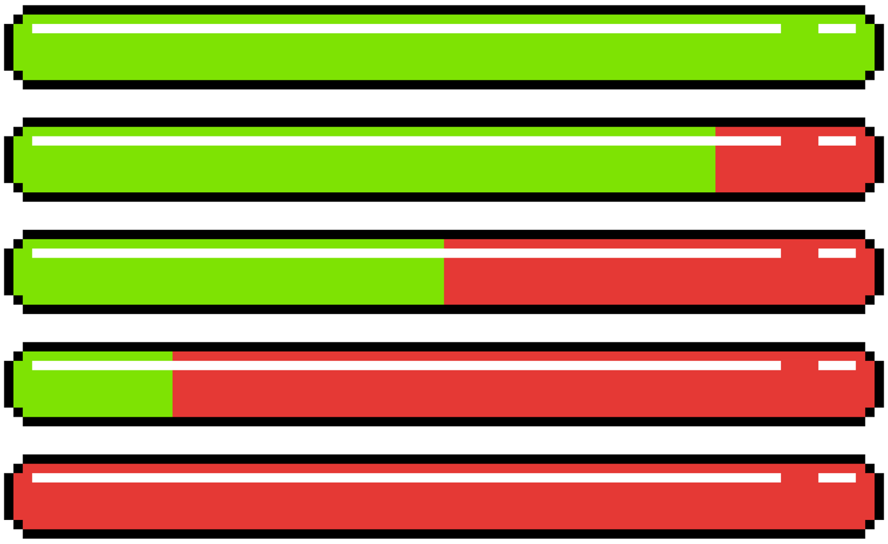


Figure 3 - Health bar

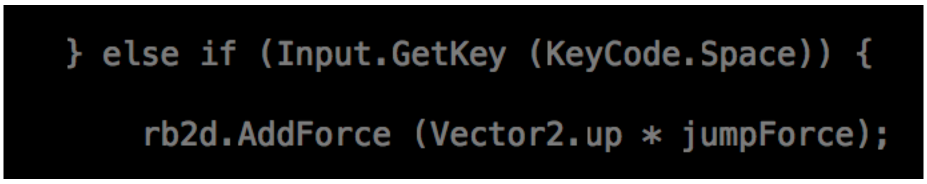
Roles

The development team consisted of three members who took on a different role. All three members discussed the overall concepts and initial ideas, James Dimech and Jeanluke Zammit Guglielmi handled the programming and Andrew Worley handled the graphics and sprite design.

Development

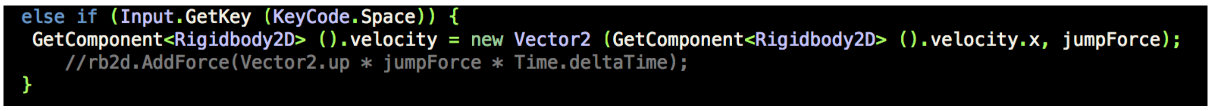
Movement

Initially, created the movement was not very difficult. However, the initial method which was taken caused problems with the jumping mechanics, as the player could only jump from a standstill, as the jumping was being performed by using a Vector 2. By using the RigidBody2D component instead, the problem was solved (please note that the code as shown in Figure 1 was commented out due to no longer being in use).



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

    rb2d.AddForce (Vector2.up * jumpForce);
```

Figure 4 - Using Vector2 for the jumping



```
else if (Input.GetKey (KeyCode.Space)) {
 GetComponent<Rigidbody2D> ().velocity = new Vector2 (GetComponent<Rigidbody2D> ().velocity.x, jumpForce);
    //rb2d.AddForce(Vector2.up * jumpForce * Time.deltaTime);
}
```

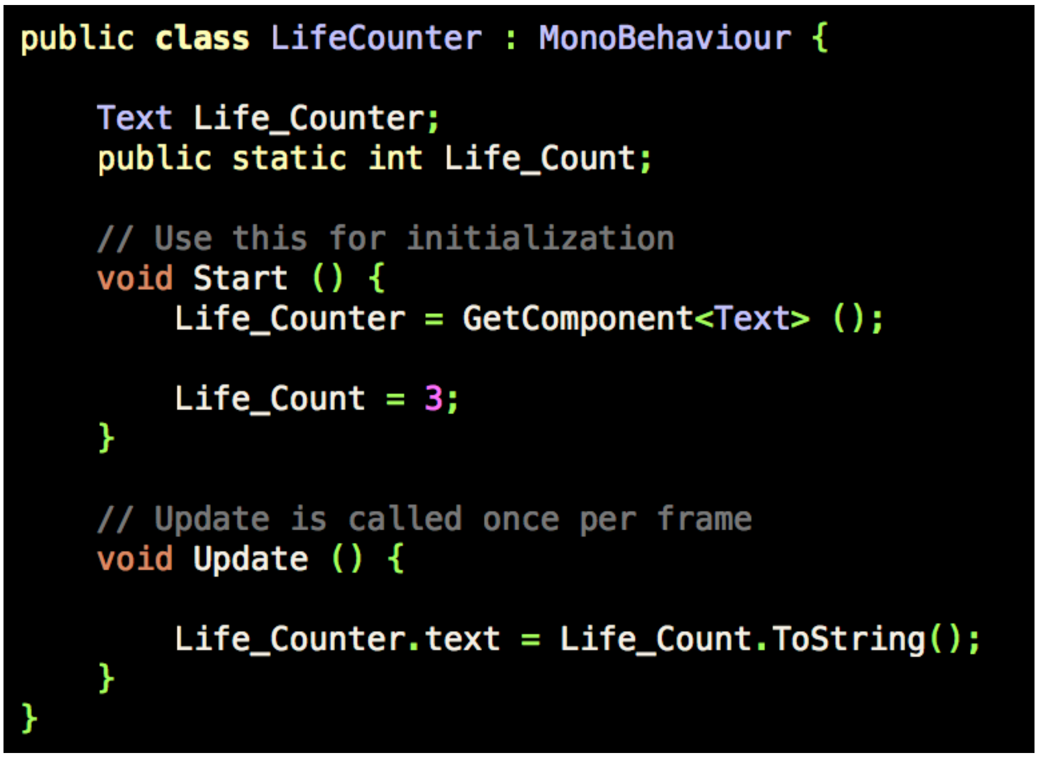
Figure 5 - Using RigidBody2D fixed the problem

Another problem encountered was that the player could jump infinitely throughout the level. The solution was to use a ground check component, although the process could not be completed, despite a variety of methods being experimented with, due to time constraints as well as the complexity of the problem.

Life Counter

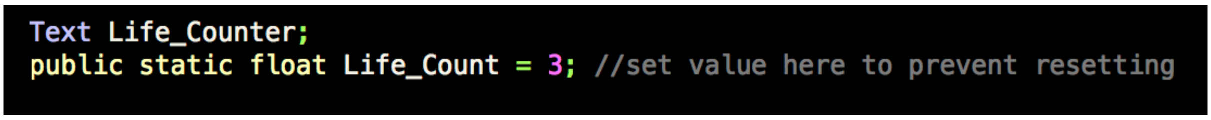
As explained in the initial ideas section, the player would have a health bar which would decrease every time they took a hit from an enemy, and would lose a life when the health bar was empty. This idea was scrapped due to lack of time available needed to learn how to make interactive UI elements in Unity, where the idea was replaced with a numeric life system which would decrease whenever the player took a hit.

Several problems were encountered with this idea: The levels needed to be reloaded whenever the player dies due to the collectibles being the main game mechanic, although reloading the scene also caused the life counter to be stuck at its set value, meaning if, for example, it was set to 3, it would remain at 3. This was caused due to the numeric value being set at *void Start()*. Eventually the value was set when the function itself was specified, which fixed the problem.



```
public class LifeCounter : MonoBehaviour {

    Text Life_Counter;
    public static int Life_Count;

    // Use this for initialization
    void Start () {
        Life_Counter = GetComponent<Text> ();

        Life_Count = 3;
    }

    // Update is called once per frame
    void Update () {

        Life_Counter.text = Life_Count.ToString();
    }
}
```

Figure 6 - Life Counter's value set at *void Start()*



```
Text Life_Counter;
public static float Life_Count = 3; //set value here to prevent resetting
```

Figure 7 – Setting the value when it was initially specified solved the problem

The problems, however, kept coming, as when the player got a Game Over, the life counter would remain at 0 when the player would start the game again, resulting in an instant game over.

A solution which was considered was to use a function called *PlayerPrefs* in order to store the original value (3) on the title screen, then call the value when the game starts. The process only worked when calling the value in *voidStart()* on the level, which brought back the same problem mentioned in Figure 3. So in order to counter this, the value would be called on a level title card, but when applied the value was never retrieved. Due to lack of time at disposal, the life system was eliminated, and the player would get an instant Game Over upon contact with an enemy or if they fell off the stage.

Timer

As explained in the initial ideas section, the timer was planned to be a da-night clock which would kill the player should it reach day time. Due to the idea being to complicated to learn how to program in a limited amount of time, this was substituted for a simple numeric counter which would decrease in value every second until it reaches 0 where the player would be instantly killed.

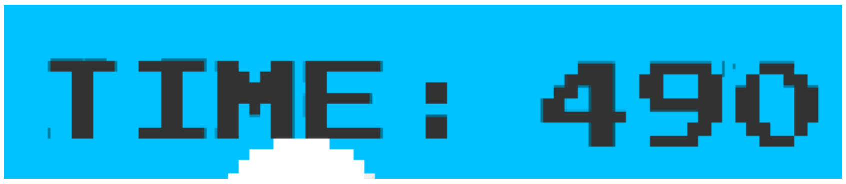
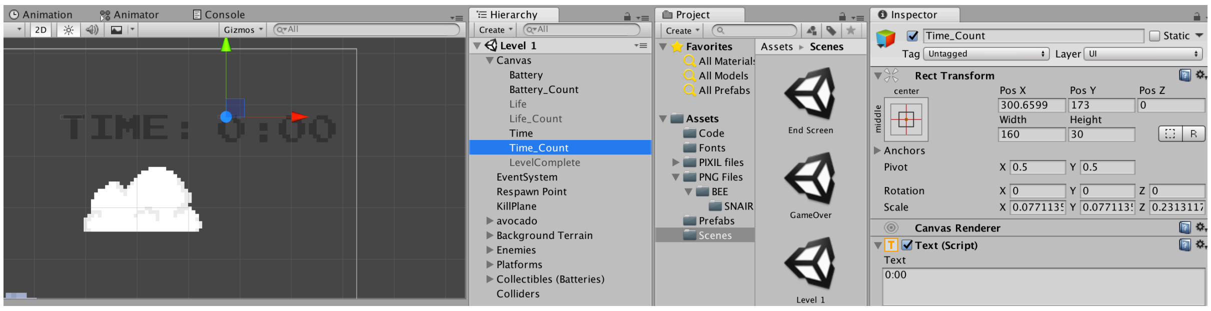


Figure 8 - Final in game timer

The timer itself was made by creating a text object and applying a script that would start it at a certain value and then proceed to kill the player it it hits 0



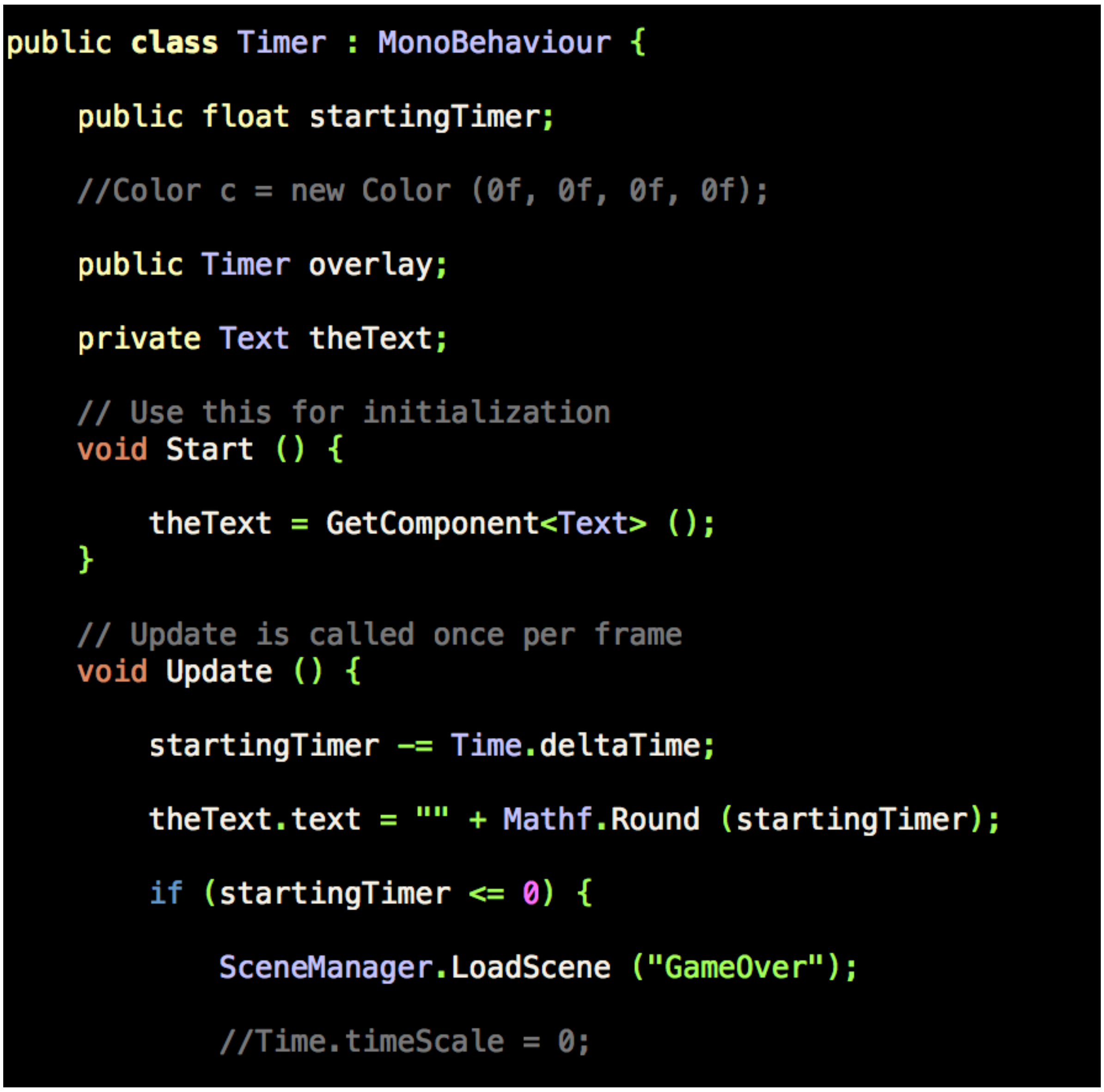
Time_Count Static
Tag Untagged Layer UI

Rect Transform
center / middle
Pos X 300.6599 Pos Y 173 Pos Z 0
Width 160 Height 30
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 0.077113 Y 0.077113 Z 0.231311

Canvas Renderer

Text (Script)
Text
0:00

Figure 9 - Text based game object



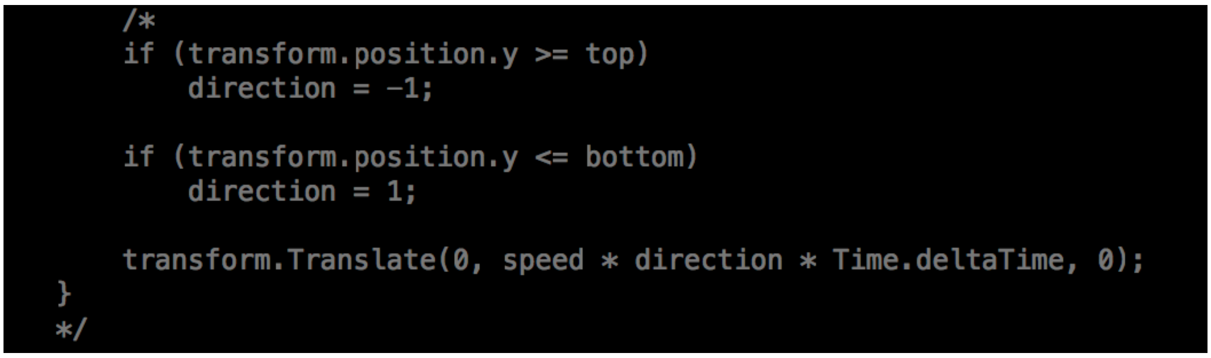
```
public class Timer : MonoBehaviour {

    public float startingTimer;

    //Color c = new Color (0f, 0f, 0f, 0f);

    public Timer overlay;

    private Text theText;

    // Use this for initialization
    void Start () {

        theText = GetComponent<Text> ();
    }

    // Update is called once per frame
    void Update () {

        startingTimer -= Time.deltaTime;

        theText.text = "" + Mathf.Round (startingTimer);

        if (startingTimer <= 0) {

            SceneManager.LoadScene ("GameOver");

            //Time.timeScale = 0;
```

Figure 10 - The timer script

Enemy Movement and Collision

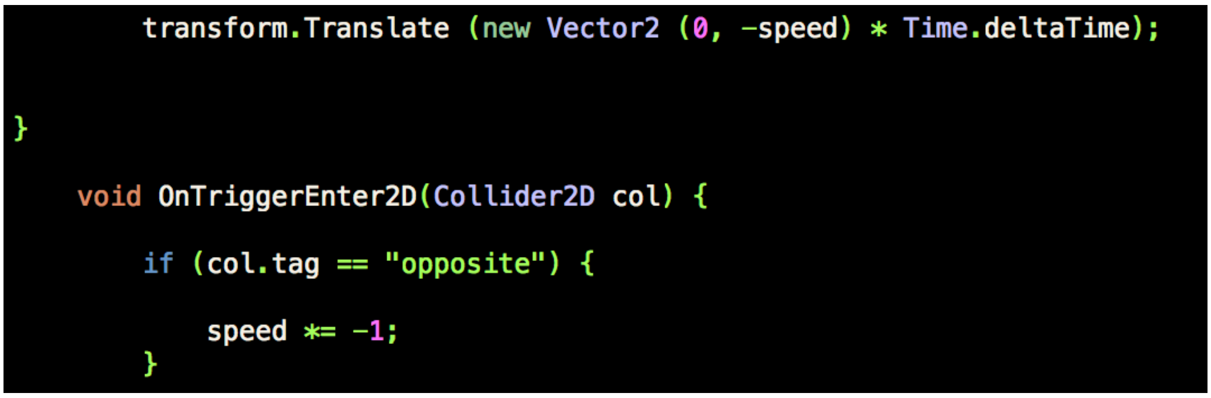
While coding the enemy movement, a script was found for moving the Bee enemy up and down on the Unity forum as shown in Figure 8. Initially, this allowed for an easy implementation of basic AI movement however, once more of the Bee enemy were placed throughout the level, the did not stay in the intended area (most of them moved through the ground).

This was fixed by making the bee move in one direction and by using an invisible collider, the enemy would switch direction. Upon testing this method and performed as intended, the same was implemented for the snail enemy.



```
        /*
        if (transform.position.y >= top)
            direction = -1;

        if (transform.position.y <= bottom)
            direction = 1;

        transform.Translate(0, speed * direction * Time.deltaTime, 0);
    }
    */
```

Figure 11 – Initial script for Bee Movement (script is commented out due to no longer being in use)



```
        transform.Translate (new Vector2 (0, -speed) * Time.deltaTime);

}

    void OnTriggerEnter2D(Collider2D col) {

        if (col.tag == "opposite") {

            speed *= -1;
        }
```

Figure 12 – New method used for Bee Movement (And Snail movement eventually)

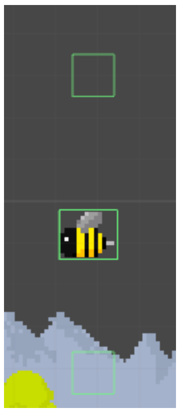
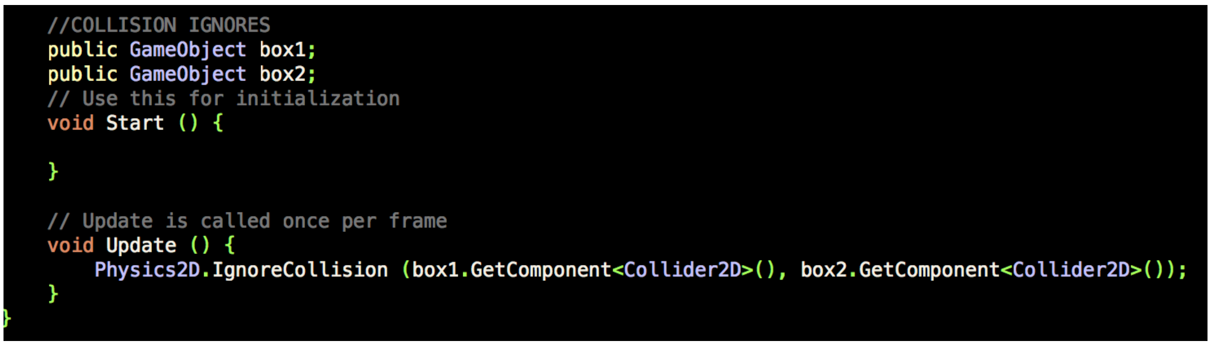


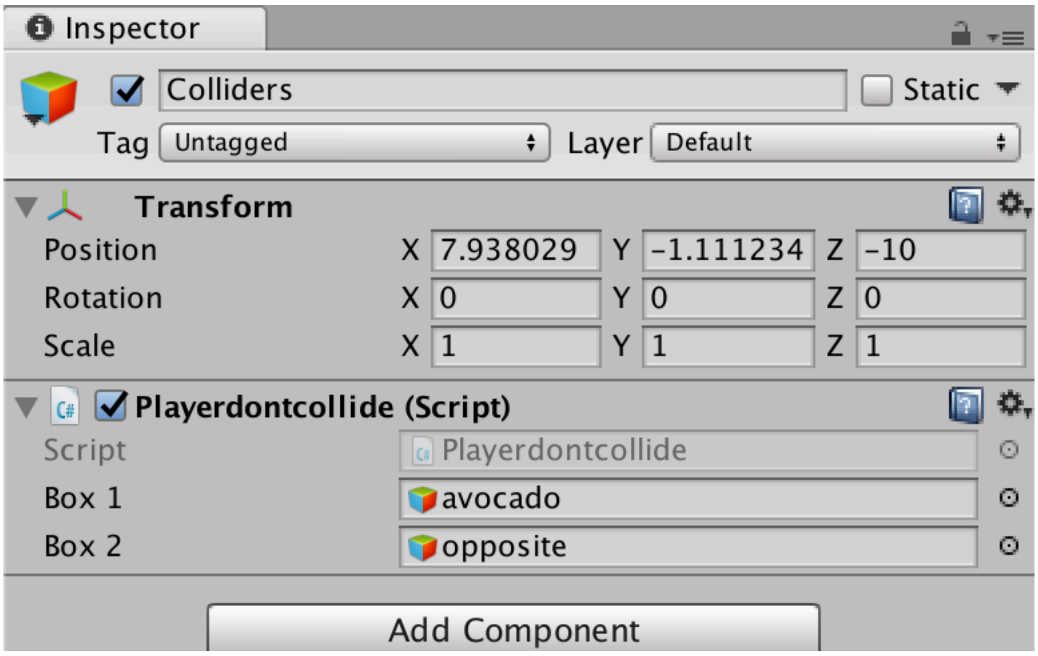
Figure 13 – Bee enemy with 2 colliders so it can switch directions

A new problem arose with this method: the player's movement would stop due to the colliders being solid. However, a simple fix was found after researching for a couple of hours by creating a new script which would tell the player game object to ignore the enemy's colliders.



```
    //COLLISION IGNORES
    public GameObject box1;
    public GameObject box2;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        Physics2D.IgnoreCollision (box1.GetComponent<Collider2D>(), box2.GetComponent<Collider2D>());
    }
}
```

Figure 14 – Setting up the script to make the player ignore the enemy colliders



```
Inspector
☑ Colliders                         ☐ Static
Tag  Untagged          Layer  Default

Transform
Position    X 7.938029   Y -1.111234   Z -10
Rotation    X 0          Y 0           Z 0
Scale       X 1          Y 1           Z 1

☑ Playerdontcollide (Script)
Script      Playerdontcollide
Box 1       avocado
Box 2       opposite

Add Component
```

Figure 15 – A new game object was made and the script was applied with the corresponding values for Box 1 and Box 2 respectively

Creating the Main Menu

Creating the Main Menu also did not take up much time, again due to already having prior knowledge on how to implement it. It was created by creating a new scene with the game name and instructions added using Text objects.

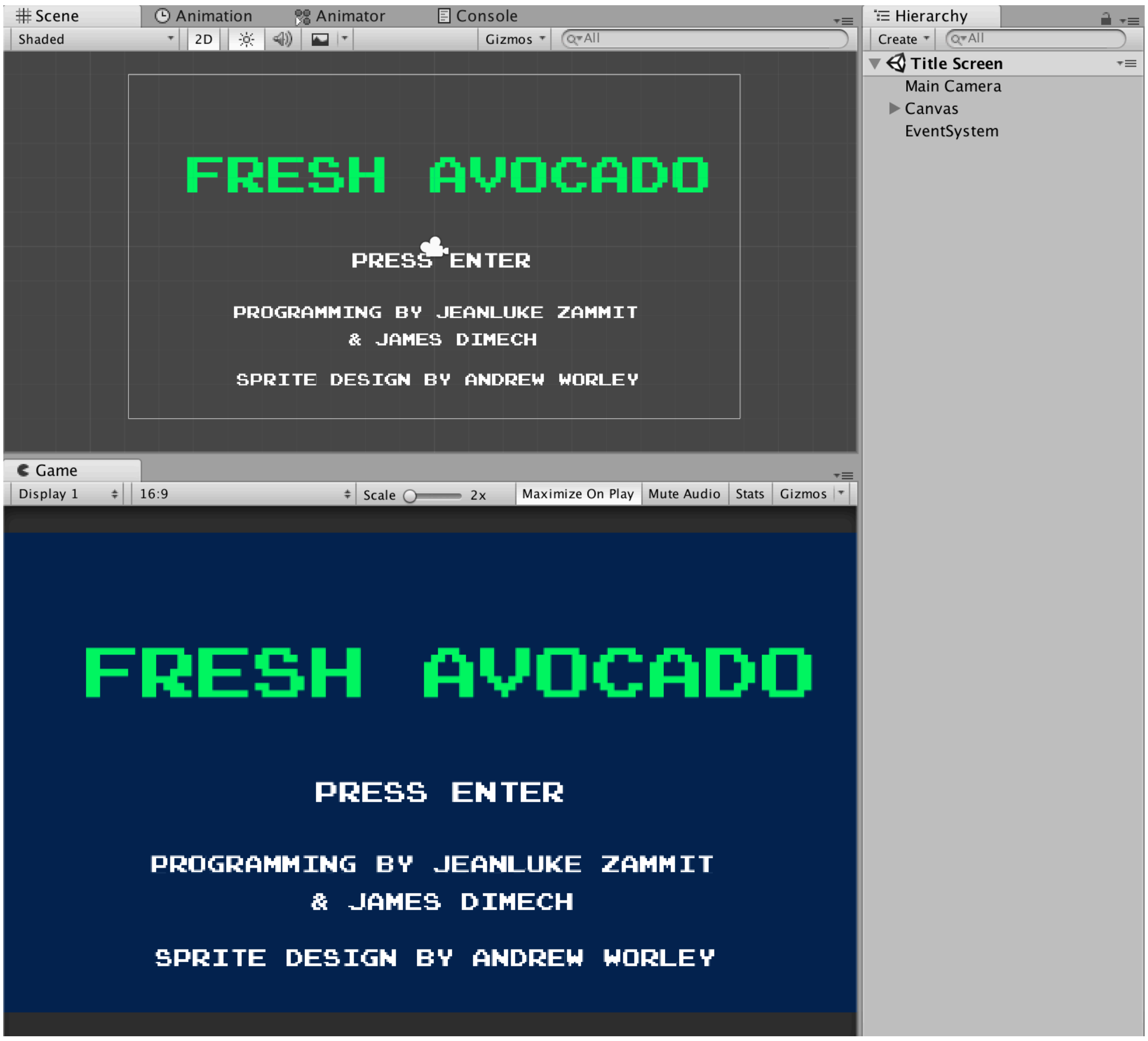
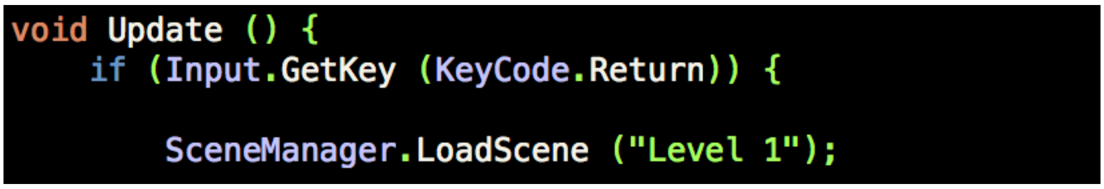


Figure 16 - New scene with the game title

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



```
void Update () {
    if (Input.GetKey (KeyCode.Return)) {

        SceneManager.LoadScene ("Level 1");
```

Figure 17 - Pressing the ENTER key will take the player to the main game

Beating the levels

After the Player collects 10 points, the intention was to have the player walk over to a fridge at the end of the level, where they would be able to enter it in order to progress to the next level. However, due to time constraints, this had to be scrapped and replaced with a simpler alternative: Display a line of text which says 'Level Complete' when the player collects all the batteries, since this was already done in a previous project done in Unity.

The process was achieved by creating a text object and hiding it firstly

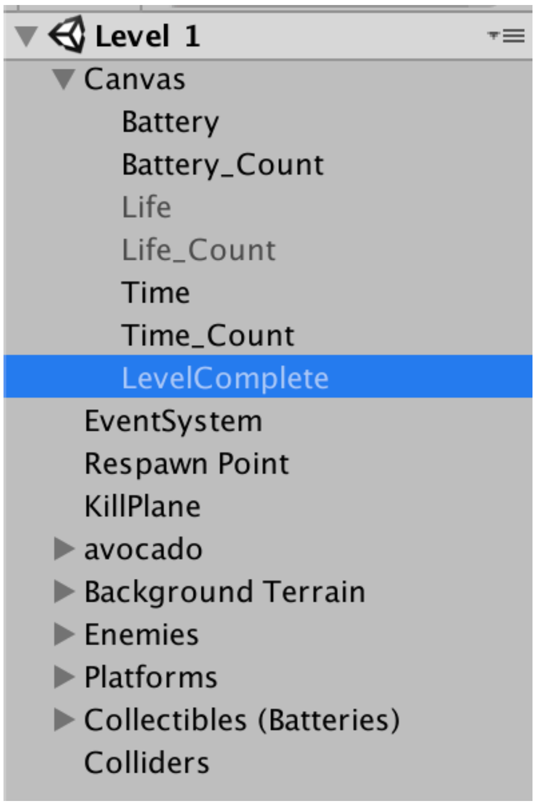

Level 1
Canvas
Battery
Battery_Count
Life
Life_Count
Time
Time_Count
LevelComplete
EventSystem
Respawn Point
KillPlane
avocado
Background Terrain
Enemies
Platforms
Collectibles (Batteries)
Colliders

Figure 18 - Creating the text object

Next, a variable was created for said text object,

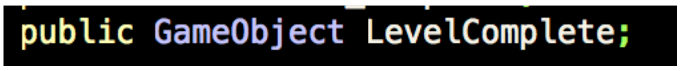
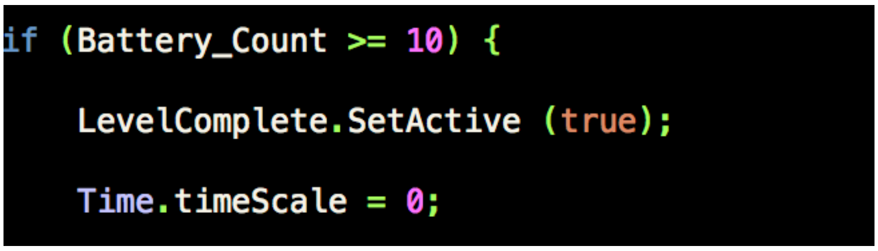

```
public GameObject LevelComplete;
```

Figure 19 - The variable

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

Finally, the commands were given to display the text when the battery count was equal to 10.

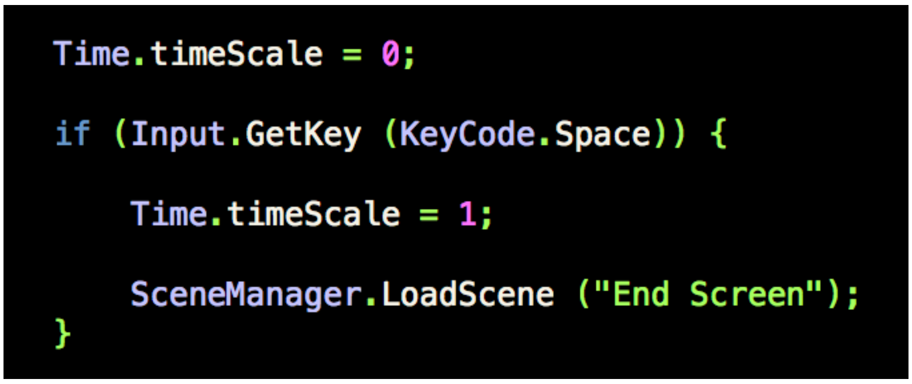
In order to freeze the gameplay when the text appears, the timescale property was used to free time.



```
if (Battery_Count >= 10) {

    LevelComplete.SetActive (true);

    Time.timeScale = 0;
```

Figure 20 - If the battery counter is 10, freeze time and display the 'Level Complete' text

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



```
Time.timeScale = 0;

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

    Time.timeScale = 1;

    SceneManager.LoadScene ("End Screen");
}
```

Figure 21 - Setting the value of Time.timeScale to 1 unfreezes the elements

Multiple Levels

Initially the game was supposed to have 2 levels each with a different theme: The first level would consist of a grassy terrain while the second would take place in a kitchen, as outlined in the initial ideas section. While sprite assets were created for the second level, due to time constraints the second level was omitted, leaving the final game with only one level, which is the one consisting of grassy terrain.

Figure 22 - Unused kitchen terrain

Player observations

When the game was finalised two people were asked to play the game and put forward their opinions on it.

One of them stated that he is not very good at platformers so the infinite jumping helped him navigate throughout the level, even though keeping the infinite jumping was not the intended functionality for player movement.

The other person found the level design confusing, which led to frustration at points, although felt a sense of accomplishment whenever he managed to secure a battery, which encouraged him to collect the remaining batteries which has has not collected yet.

How could the game be improved?

If more time was available, the level design would be refined further so as to allow for a less confusing experience and the intended UI elements which were initially planned would be implemented due to having more time with experimentation with regards to understanding how to implement said features.

One strength that the project had was that despite all of the problems encountered, the initial ideas, were, in a way, successfully implemented, and that everyone gave efficient contribution towards the project.

One weakness, however, was bad time management, as the project started being developed at a very late stage due to the development team being established late during the production timeline, resulting in lacking time to properly develop the game as intended. For future projects, the team would be established earlier so more time could be spent during the development stage, allowing for a more refined video game.