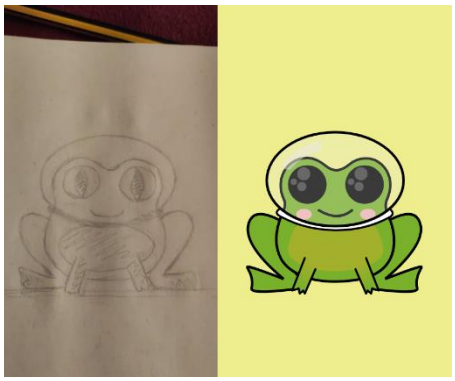
Basics of Game Engines ~ Task 3

A technical report explaining the process of how the game was developed



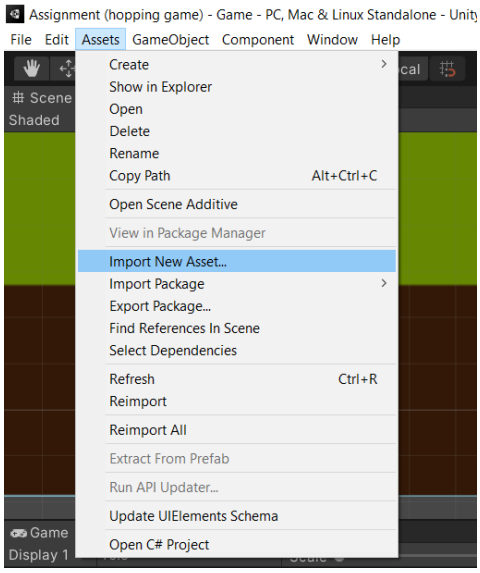
Graphical Assets

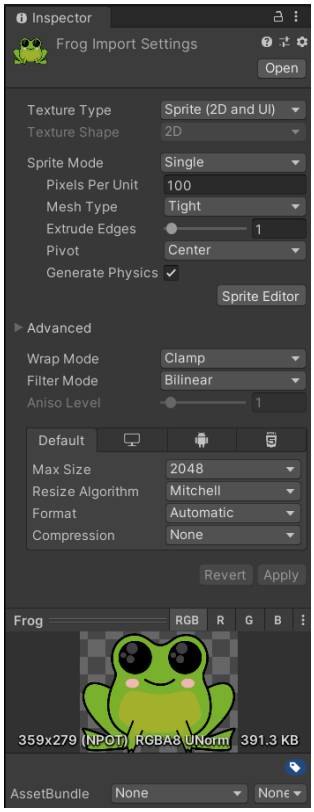
Illustration assets

The first step of the process of making the game assets was idea generation where we were thinking of options of how we can create our game. During this stage I came up with the idea that we should make the main character a little frog and that the game background would be divided into 6 different sections. The first section being grass, the second being trees, the third being the sky with birds, the fourth being clouds, the fifth being the moon with stars and the sixth being planets.

After generating these ideas, I started searching for some inspiration for the main character as well as for the background. This image to the right was used as inspiration for the game background. Then I made a rough sketch for the character design of the frog on my sketchbook. Then finally I designed both the background and the character on a vector illustrating software called Gravit Designer.

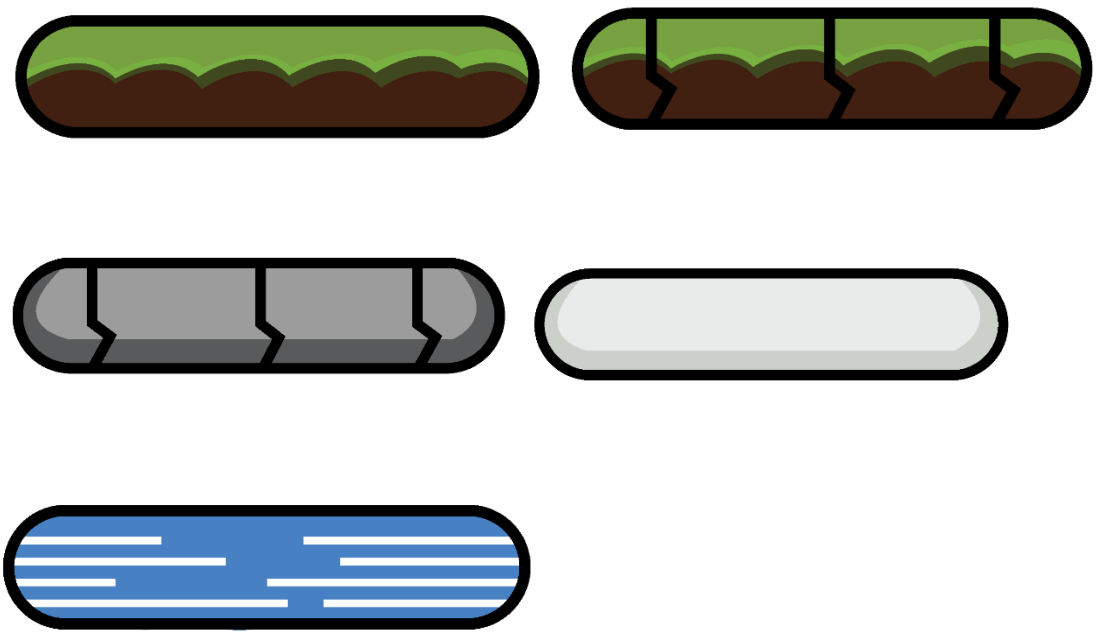
For importing the game graphical assets such as the game character and background into unity I went to the unity menu bar clicked on assets, then on import new assets and then located and imported the file which contained my assets. Once I imported everything, I adjusted their import settings by changing their texture type to Sprite (2D and UI (User Interface)) and set their quality to maximum size (2048), for them to be clear. Then I applied these changes as can be seen in one of these screenshots. Then I positioned all the sprites in their appropriate space.





Platform Game Assets

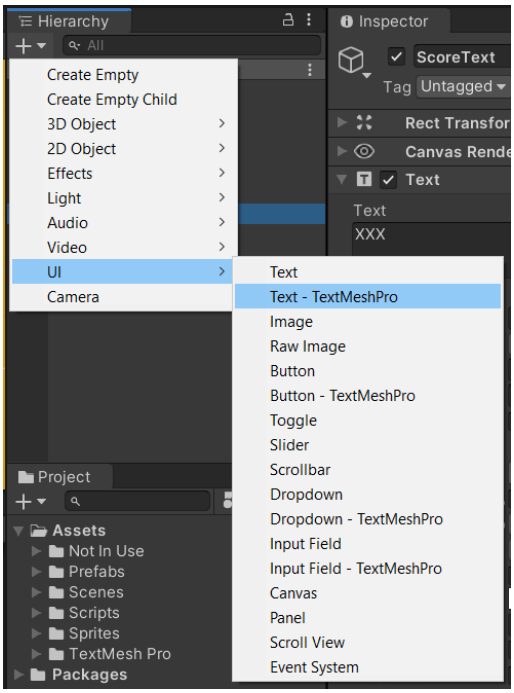
For the platforms, we started by sketching out ideas of how we would like it to look. We started thinking about their uses and coming up with designs that would make it easier for the players to understand. We than took our sketches to illustrator and Decided to start perfecting these platforms. We than exported them into the game with a transparent background.

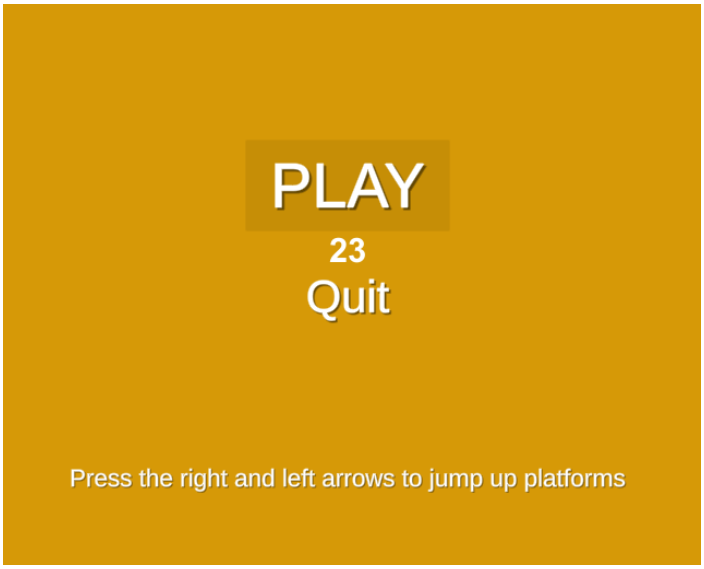


UI assets

To create my first UI assets, which was the in-game score, I followed the steps that were shown to us during our last tutorial lecture. For this I created a UI text object and then added a script to it so that this text shows the game's score.

For the UI objects I created for the menu page I first made the UI Text set the colour and positions and all their attributes and then I added UI buttons and replaced their text with the text that I had already prepared. Then I decided to make the text look a bit better, so I downloaded a unity package called TextMeshPro and remade the text, which made it clearer and gave it a bit of a shadow.





Scripts Overview

Player script

Here we are giving our "player" character, its controls and movement.

1. In the start function we are getting the rigid body object from the inspector
2. In the update function we are creating a vector 2 object equal to the velocity of our rigid body
3. In our update function we are checking if our rigid body is grounded or not. If our rigid body is grounded, we are setting velocity "X" to Zero which will stop the rigid body from moving left and right when grounded. We are capturing when the user hits the left or right arrow which will call the "Jump" function. We are also setting a bool true or false depending on which arrow the user pressed.

```
56      Vector2 velocity = rb.velocity;
57      isGrounded = Physics2D.OverlapCircle(groundCheck.position, 0.2f, GroundLayer);
```

4. In our fixed update function on line 57 we are checking if our rigid body is on the "ground layer" or not. We are calculating how far the rigid body moves from stationary to its next jump. This value will determine when to move to the left or right after jumping upwards. If the rigid body jumped higher than "2f", we will than check with our bool to determine which direction the rigid body will move (Left or right). In line 98 we are checking if the rigid body is grounded to reset the distanced travelled value to zero and saving the "y" position of the rigid body into a variable If the rigid body moves of a total distance of "6f" or more this means that the rigid body has fallen from the platform and the restart level function will be called.
5. The jump function work by simply multiplying "vector2.up" by a value that increases the height of the rigid body's velocity.
6. Restart level function includes the "scenemanager.loadscene(0)" to restart the scene.

Level Generator

In this script we will be randomly generating our platforms.

1. In our start function we are adding our platform prefabs into a "list" of type "gameObjects."

```
32          // loop for the amount of levels
33          for (int i = 0; i < numberOfLevels; i++)
```

2. On line 33 we will see that there are 2 "forloops" to generate our platforms. In the outer loop we will be generating the platforms for the number of levels needed. In the inner loop we will be generating the platforms for the number of platforms needed at that level. In the inner lop, from line 38 to 49 we will be introducing the harder platforms as the loop continues to iterate since after iteration the range of our randomiser continues to increase. Then we will instantiate our platform prefab depending on the random number generated within the range.

```
38          var _i = i /2;
39          if (i < 5)
40          {
41
42              //
43              prefabIndex = UnityEngine.Random.Range(0, _i);
44
45          }
46          else
47          {
48              prefabIndex = UnityEngine.Random.Range(0, 5);
49          }
```

3. On line 54, after we generated a platform, we will move the spawn position for our platform to the right. This will be done until the inner loop has stopped looping. On our "outerloop" we are moving the platform spawn position back to the original spawn point and increasing the "Y" value of the spawn position to go up another level.

```
54          spawnPosition1.x = spawnPosition1.x + platformSpacerX;
```

4. On line 67 we can see that the "forloops" are identical, the reason for this is because due to the game design of the platforms, the platforms must be in a chequered fashion. Therefore, the platform levels are being generated depending on their beginning spawn point to give the platforms their proper spacing.

```
67          for (int i = 0; i < numberOfLevels; i++)
```

Platform behaviour

In this script we are giving our platforms some characteristics whenever our “player” lands on some of our platforms

1. In our 2d collision function, depending on the platform ID we are destroying the object after a specific time whenever the “player” lands on the platforms with ID “0” and “1”. Platform “0” Being the grass platforms and platform “1” being the broken grass. As for ID “2” When the player lands on the platform the player will lose, and the game will restart.

```
22      void OnCollisionEnter2D(Collision2D other)
23      {
24          //Adding timers to platforms
25          if (other.gameObject.tag == "Player" && _blockID == 0)
26          {
27              Destroy(this.gameObject, 3f);
28
29          }
30          else if (other.gameObject.tag == "Player" && _blockID == 1)
31          {
32              Destroy(this.gameObject, 1f);
33          }
34          else if(other.gameObject.tag == "Player" && _blockID == 2)
35          {
36              SceneManager.LoadScene(0);
```

Ghost Platform

This script will give our ghost platform its disappearing and reappearing feature.

1. In our update function we are calling the pulsate method.
2. In our pulsate function we are using the "Mathf.PingPong" method which will reach to 2 seconds and saved into a variable called "time".
3. If our "time" variable is less than 1, we are setting our platforms renderer and collider to false, removing its appearance and the collider will stop working.
4. If our "time" variable is greater than 1, we are setting our platforms renderer and collider to true, making its appearance and the collider to work again.

```
1 reference
void Pulsate()
{

    time = Mathf.PingPong(Time.time, 2);

    if (time < 1f)
    {
        BrokenPlatformPrefab.GetComponent<Renderer>().enabled = false;
        BrokenPlatformPrefab.GetComponent<Collider2D>().enabled = false;
    }
    else if (time > 1f)
    {
        BrokenPlatformPrefab.GetComponent<Renderer>().enabled = true;
        BrokenPlatformPrefab.GetComponent<Collider2D>().enabled = true;
    }
}
```

Camera Follow Script

In this script we will be setting the camera to move with our "player" object only on the" Y" Axis

1. If the "playerobject" moves upwards the camera will follow its coordinates.

```
    2 references
7   rm target;
8
9
10  to follow frog on Y axis
    0 references
11  e ()
12
13  goes up in y axis, camera goes up in y axis also until centered
14  .position.y > transform.position.y)
15
16  3 newPos = new Vector3(transform.position.x, target.position.y, transform.position.z);
17  orm.position = newPos;
18
```

Scorekeeper script

The first thing I did was I declared that I am using unity's UI system.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
```

Then I added fields which are visible in unity. Through these fields you can link which text will show the specific function assigned to it in the script. I made two separate fields even though the same result needs to be shown, because whenever I linked the menu's score text to the score the score did not show.

```
    2 references
6   public class ScoreKeeper : MonoBehaviour
7   {
8       [SerializeField]
9
10      //Adds Score field in Unity
        1 reference
11      public Text Score;
12
13      //Adds menuScore field in Unity
        1 reference
14      public Text menuScore;
```

Before I started coding, I create a private integer called currentScore and set its value to 0 to ensure that when the game starts the score starts at 0.

```
        3 references
15      private int _currentScore = 0;
```

The next thing I did was create a public void called IncrementScore where I added the coding needed to increase the currentScore by 1. Then I stored that score using PlayerPrefs so I can use it later. Then I recalled the currentScore stored in the PlayerPrefs and set it to the menuScore.

```
            0 references
17          void Start()
18      {
19          //recalls last _currentScore stored
20          menuScore.text = PlayerPrefs.GetInt("MenuScore", 0).ToString();
21      }
22
        1 reference
23      public void IncrementScore()
24      {
25          //takeing the _currentScore + 1
26          Score.text = "Score: " + _currentScore.ToString();
27          _currentScore++;
28
29          //Storing _currentScore
30          PlayerPrefs.SetInt("MenuScore", _currentScore);
31      }
32  }
```

Addpoints script

In this script I checked if the player was colliding with the platforms' 2D collider, if so, I give the command to access the ScoreKeeper script to increment the score. Then the last string of code is there so that the score cannot be added twice if the player touches a platform's 2D collider twice.

```
0 references
public class Addpoints : MonoBehaviour
{

    0 references
    private void OnTriggerEnter2D(Collider2D other)
    {
        //checking if there's a collision with player
        if(other.tag == "Player")
        {
            //a reference to the scorekeeper script
            ScoreKeeper scoreKeeper = GameObject.FindObjectOfType<ScoreKeeper>();

            //checking if script was found
            if(scoreKeeper != null)
            {
                //incrimentScore
                scoreKeeper.IncrementScore();
            }
            //unallowing score to double if player lands on the same platform twice
            Destroy(this.gameObject);
        }
    }
}

```

MainMenu script

In this script has two functions. The first one is the PlayGame function, which allows the player to load the next scene, which is the game scene. The other being the QuitGame function, which allows the player to quit the game and close the application. For the PlayGame function to work I had to first declare that I will be using the unity's Scene Management system, as can be seen at the top of the snippet below.

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

0 references
public class MainMenu : MonoBehaviour
{
    0 references
    public void PlayGame()
    {
        //Loads next scene in game manager
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    0 references
    public void QuitGame()
    {
        Debug.Log("Quit!");
        //Closes application
        Application.Quit();
    }
}
```

The game making process

The first part of the process of making this game was searching for inspiration for our game. Once we found a game that we both liked to replicate we started the planning process where we planned out what adjustments we would make to this game to make it our own and what we would need to do so. This included things such as the device specifications and keys needed for the movement, a flowchart of the gameplay, the pseudocode needed for the game, a breakdown of the game play, the game objective, and the game assets, which we had created ourselves.

Then we created the unity project and linked it to GitHub. Then we started by adding and setting up a few game assets, which where the frog, which is the player and the game background. After that we gave the character some basic jump movements by adding a script that gives it a short hop wherever the right or left arrows are pressed. Then we added a background script to the background that makes it go down by a specific amount whenever the right or left arrow is pressed, to complete the illusion for jumping.

Then we created two simple animations. One of them was an idle animation for when the game starts and an idle animation for when the character stops jumping. But then we realised that we added animations too early, so we removed them. Then we imported the rest of the game assets, which were the platforms, into the game and adjusted them accordingly.

Then we started searching and testing out different methods of how we could make the platforms spawn automatically in a random order. When doing so we both found two different methods but they both did not work as well as they should. Then we decided to combine the two methods and it worked much better but later we realised that this method still was not the best therefore we removed this and tried out a different method.

We than created a “playerObject” which is now known as “frog”. This “GameObject” includes a “Boxcollider2d” component and a “Rigidbody2d” component. We than remade the “Player” script to make our “playerObject” s jumping movement more accurate. We then realised that the camera was not following our “player” object, so we added a script to the main camera for it to follow the “player” whenever its jumping upwards.

Once the player movement was done, we then generated the platform levels and aligned the platforms spawn positions with the jumping force value of the frog for it to land in the middle of the platform. Once the platforms have been generated and the frog was jumping on each platform successfully, we started to add distinctive characteristics to our platforms and used a randomiser to generate the platform variations randomly.

After the game was done, we added a scoring system to the game, where the player gets a point every time the frog jumps and lands on a platform. Once we managed to add the score at the top left of the game page we decided to move on to the next step and make a main menu, for which a script was made and attached to. This main menu includes a play button, which allows the player to play again, the game round’s total score, a quit button, and some simple game instructions at the bottom. The only problem with the main menu was that the total game score in it does not match the one in the game since it always adds in an extra point.

When it came to adding in the splash screen, we decided to make an animated one instead. This was made completely in Aftereffects using the current assets we had along with the game name we had created. After the animation was done, we searched, chose, and added in free usable audio into this animation. Once the animation was done, we exported it and imported it into our unity project where we inserted it into its own scene called Intro animation and placed it before the main menu and the game scene. Then we added a script called Wait to the camera of the scene. What this script does is it waits for an amount of time given, which in our case was the duration of the loading scene animation, and after it is done waiting it loads the next scene.

After that we set up our game's settings such as our company name, our game name, and added in our game icon. Then we built our game and ran it and started testing our game.

Link to our GitHub: https://github.com/LunarChild101/AssignmentProject

Testing

From the results obtained from the below chart we know that our game is playable and mostly in a good condition. However, there are still a few things which should be improved within this game.

In our game there are two major issues which really need to be fixed. The first main issue is that sometimes a row of white (cloud) platforms appears and does not allow the player any chance of progressing within the game no matter how careful the player is. Then the other main issue is that when a platform's timer runs out the player simply falls to the platform below it instead of dying. Therefore, giving the player an extra chance to continue the game when the game was supposed to end.

The rest of the game's issues are minor and are not going to affect the player's experience that much. The first minor issue is the accuracy of the frog's jump since it does not always jump at the centre and sometimes it hits a platform's collider. Another minor issue is that once the game ends the game loading screen animation loads right before the main menu when it is only supposed to appear when you start up the game only. Then the last minor issue is that the score obtained during the last game played always shows with an extra point in the main menu.

Functionality	Expected Output	Actual Output	Status
Pressing the Play button loads the game scene and starts the game	Starts the game	Starts the game	
Pressing the left arrow key makes the character jump to the left.	Character jumps to the left	Character jumps to the left	
Pressing the right arrow key makes the character jump to the right.	Character jumps to the right	Character jumps to the right	
The accuracy of the frog	Where the frog lands	The frog, for the most part lands in the centre of the platforms. The frog sometimes collides with the collider of the platform above it, most commonly with the ghost platform.	

Jumping up a platform adds 1 to score points.	Score + 1	Score + 1	
When the character jumps on the stone platform it never gets destroyed.	Stone platform doesn't get destroyed	Stone platform doesn't get destroyed	
When the character jumps on the grass platform it gets destroyed after 5 seconds.	Grass platform gets destroyed after 3 seconds.	Grass gets destroyed after 3 seconds	
When the character jumps on the broken grass platform it gets destroyed after 1 seconds.	Broken grass platform gets destroyed after 1 second.	Broken grass is destroyed after 1 second	
The blue platform turns on and off every 2 seconds making it appear and disappear.	Blue platform appears and disappears every 2 seconds	Blue platform appears and disappears every 2 seconds	
When character jumps on blue platform when it is on during that 2 seconds, the character can land on it and jump up to the next platform.	Player lands and jumps to the next platform	Player lands and jumps to the next platform	
In every row of platforms there should be at least 1 usable platform	Every row contains at least 1 usable platform	True but sometimes a line of white platforms (clouds) appears.	

When character jumps on the white platform the game ends immediately.	Game ends immediately	Game ends immediately	
Once the game is over the game loads back to the MainMenu	Loads MainMenu	Loads game loading screen animation, then main menu	
The final score of the game should be showing on the MainMenu.	Shows score achieved	Shows the score achieve +1 extra point	
When quit button is pressed the application closes.	Application closes	Application closes	
After Loading Screen animation ends the game loads	The game loads	The game loads	
After any of the timed platforms' time runs out the game ends immediately	Game ends immediately	Player falls to below platform where the game can continue	