Computing Project

(Comp1108)

A 3D game bases on Unity Engine

Student Name:	**Vo Minh Nhut**
ID Number (FPT CMS):	**NHUTVMGC60222**
Student Registration # (Greenwich CMS):	**000901562**
Banner ID (Greenwich CMS):	**vn6073s**
Supervisor:	**Dr. Nguyen Van Sinh**

A dissertation submitted in partial fulfilment of the University of Greenwich's Bachelor of Science (Honours) in Computing

Abstract

Within the recent years, there has been a huge demand for entertaining. This has been a golden era for entertainment, especially game industry to develop. In fact, there has also been a lot of successful games on the market. However, some were not so successful as the rest. This might be due to the rush of productivity that was overwhelmed the interest in designing a high engaging game.

Prior to the introduction of gaming frameworks, games were mostly designed frame by frame over low-level programming languages such as C++ or Python. This was very time-consuming, and the game makers could not spend time in develop a highly engaging game scheme.

Game programming frameworks such as Unreal® Development Kit, Cry® Engine, Unity® Engine were introduced as an alternate solution for game developers. These game engines have its pre-built structure which would take responsibility for maintaining a smooth-running game and leave some free time for the game maker to design a better gameplay.

Within this project, it would analyse the current circumstances of the game industry and the trend which was shown that was more interested in cross-devices game playing and multiplayer interaction. After the consideration of the three above techniques, it was Unity® that was used to implement a prototype for this project.

Acknowledgment

I would like to say many thanks to **Dr. Nguyen Van Sinh** for practical recommendations that helped me a lot in comprehending and solving the issues which I ran into while working on this assignment.

Table of Contents

Abbreviations List

GPU:	Graphics Processing unit
API:	Application programming interface

1. Introduction

1.1. Overview and Motivation

Southeast Asia – A new game market

According to a research of Niko Partners – a leading organisation of marketing and customer research along with consulting service in Asia game industry, approximately 85 million players in Southern Asia spent $661 million in 2014 and the spending was expected to reach a 1.2 billion peak in 2017 (Leonal, 2014). Particularly, a research of Newzoo – another consulting company found that $233 million was a game industry's revenue of Vietnam during the year of 2013 (Newzoo, n.d.).

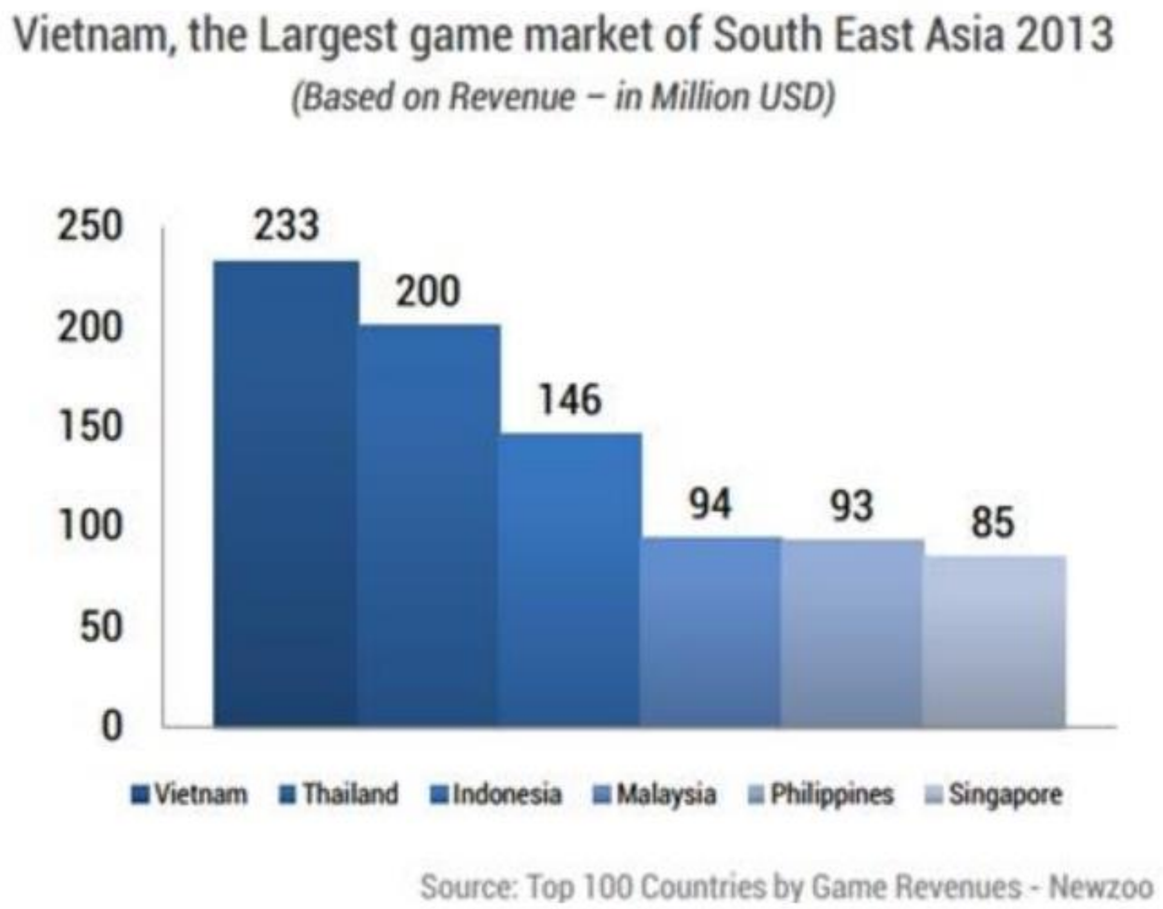


Figure 1: Game industry revenue of ASEAN countries in 2013
(Adapted from Newzoo Inc,2013)

Within the last five years, it has been witnessed that amongst the popular games, e-Sport and MMORPGs have a very remarkable growth (Gaudiosi, 2015). According to Newzoo (Newzoo, 2016), the anticipated revenue of eSports global market for 2016 is approximate $463,000 in which the SEA takes up to 40%. This expectation relied on the average growing rate of 43% per year (ibid), which is fairly spectacular. In terms of MMORPGs, as stated by the Gaudiosi (Gaudiosi, 2015), the final revenue for

MMORPGs is around $784,000 for Personal Computer(PC) Gaming and approximately $231,000 for Mobile Gaming. Additionally, it has been forecasted that 37% per year is a growth rate of MMORPGs in the SEA region throughout the year of 2018 (ibid). It could be seen that SEA region might be an ideally market to develop a new game.

Arcade games – A new classic

Arcade games once were fairly popular from the late 70s to the middle of the 90s (Wolf, 2008). Originally, arcade games were also called “coin-op” because it operated based on coins which were usually put in public places such as malls, playgrounds, clubs (ibid). It was fairly obvious that arcade games had become a golden egg of game industry (Kent, 2001). However, the technology evolution made a switch from 2D games to the more advanced 3D games and the arcade games eventually lost its place in the industry (ibid).

The decrease in popularity of arcade game was mainly because of the public interest but besides that, there were some other problems that led to the fall of this game genre (ibid). Firstly, there was a majority of the 90s arcade games that were programmed in 2D interface, so it was not able to recreate the 3D experience for the gamers (Heaven, 2015). Secondly, the games were hard-integrated into the game machines. Therefore, it could hardly keep up with the trend (ibid). Therefore, as a consequence, these problems led to the decline of arcade games in the late 90s and early 2000s (ibid).

However, within the recent years, there has been a considerably remarkable comeback of arcade games in the form of casual mobile games (ibid). Derived from arcade games, the new casual games such as Angry Bird, Fruit Ninja, Candy Crush Saga have always been in a very top of the rating chart since the day they were introduced (Chen and Leung, 2016).

According to Business Insider, Angry Birds itself brought back to its owner company approximately $96 million in revenue for 2011 then $200 million for 2012 (Shead, n.d.). Additionally, according to The Guardian’s statistic data, Candy Crush Saga has also brought back over $1.3 billion in 2014 alone (The Guardian, n.d.). It could be considered that the future of arcade game, especially casual game is brighter than ever before.

1.2. Problem Statement

First of all, it is necessary to consider the fact that most of the current popular games are free to play (Healy, 2016). However, the gamers will need to pay to win the games (ibid). In fact, most of the game industry revenue is the money that was paid on in-game merchandises (Gaudiosi, 2015). Consequently, despite its gameplay and graphic design, such games might lose its attraction and popularity since there would probably be some gamers who would find those games unfair as the price to win might be unaffordable (Caoili, n.d.). Therefore, there is a relatively high demand for a widely affordable.

Secondly, due to the decrease of interesting in those excellent games above, some gamers have found the classic non-combat games to be interesting (Chen and Leung, 2016). In fact, as mentioned in section 1.1, the annual revenue of Arcade games has been growing at a significant rate. These facts might imply a demand for a new trend of game development. However, it is needed to considerate that most of the current free arcade games are quite poorly or outdated designed (ibid). For instances, some spaceship attack games use the graphic design from the 90s which is somewhat familiar to all the user and also somewhat not interesting anymore (Stuart, 2014). Consequently, the problems of an up to date graphical design is another factor that needed to be taken into account.

Thirdly, currently in the market, there is an increasing competition between the platforms, this might lead to the unavailability of some trending games for some of the less focused platforms (Rickard, n.d.). As an example of this discrimination, there are times when some trending games that were made available to iOS before being provided to the rests (ibid). This might somehow reduce the heat of the game itself and also the potential revenue of the producer. Most importantly, it might restrict the reachability of the game to the potential loyal gamers and consequently the popularity and lifespan of the game itself (ibid). Therefore, it is necessary to develop a game that has the ability to run on many types of devices regardless the platform.

Furthermore, it is very expensive to develop an application, more specifically, a game. According to Erminesoft (Erminesoft, 2016), it took Imangi Studios a price up to $250,000 to make the famous Temple Run game. The development cost consists of hiring the developers, core programmers, graphical and AV designers and also the licenses for using the platform (ibid). As a result, it is relatively hard for game developers to create a game without a solid financial backup.

Additionally, it is a relatively common reality that the players are expected to invest into the games not only by tradable value objects but also an enormous amount of time to stay competitive in the game (Adachi and Willoughby, 2011). However, it has been claimed that killing time is among the principal purpose of games (Cummings and Vandewater, 2007). As described in their report, Cummings and Vandewater stated that there are a number of teenagers and adults has been spending time playing game for recreation (2007). Hence, regarding the customer benefits, it might be the best to combine those two concepts above and design a game that do not require a mass amount of time for a player to be classified as an advanced one, yet the game also need to have a reasonable gameplay time so that it can stay interesting to the players.

In another aspect, currently, there is a majority of the popular games that is full of violent scenes and that is no good for the public (ibid). According to Adachi and Willoughby (ibid), violent factors in video games might lead to a more frequent outburst of aggressive behaviours and possibly criminal actions. In such a busy lifetime, an aggressive behaviour is probably the last item in the wishing list of this society (ibid).

Therefore, a more generally appropriated game would be a decent alternative to the violent current ones.

1.3. Scope & goal

Within this project, the aim is to use an advanced game development technology to successfully develop a 3-D game that consists of the following properties:

1. The game must have a decent 3-D graphical design so that the gamers would properly feel interested in playing it.
2. The developed game should not contain or at least contains a very small factor of violent actions.
3. The game should be made free to the user, either to play or to win so that the only factor that distinguishes the gamers is their effort and skill.
4. The game should be available on 2 or more platforms so that the platform dependency would not prevent the game from becoming popular.
5. Finally, the cost of developing such a game should not be too high so that the game developer can be more freely in implementing and deploying their ideas of the next trendy game.

1.4. Structure of the report

Outline of the report

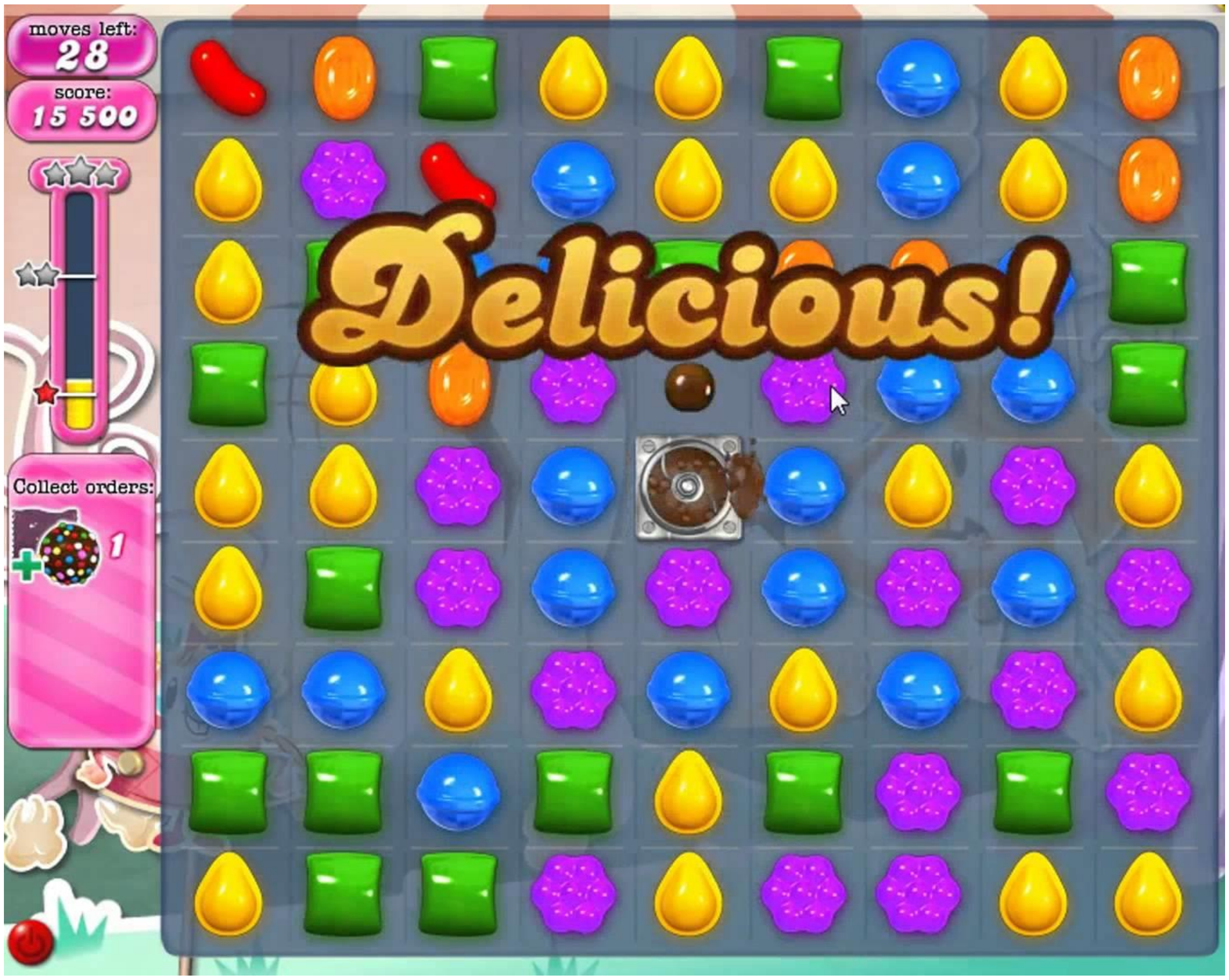
2. Literature review (requirement specification)

2.1. Existing games overview

There have been many successful game titles in the industry for the last few year, this section of the report tends to provide a deeper view into these successes regarding the design and the technique that was used.

2.1.1 Candy Crush Saga

Candy Crush Saga (CCS) is a mobile-based game which was developed by King Studio in 2012(Wikipedia, 2016). After a short period of time, the game went viral dues to its easy to understand yet challenging to conquer gameplay (ibid).



In terms of the design, within this game, there are puzzles in the form of the wrapped or unwrapped candies that imply different gainable score for each type of the candy(Wikipedia, 2016). According to Silverman, this type of design make it more engaging for the players as it differentiates clearly between different types of score objects(the candies) however, the whole set of objects remains in-sync so that it eliminate the confusion and hence improve the engagement of the game (2003). Regarding the context, Candy Crush has a very interesting level system so that it can almost literally last for a long time; it can mix a variety of obstacles and missions such as multiple thrusts to eliminate special candy or a high score to pass a level (Wikipedia, 2016).

2.1.1. Fruit Ninja

Fruit Ninja is a game title that was developed by Halfbrick game studio which based in Australia. This is a very successful video so that it has been developed into many versions for different devices from mobile to game console and especially, it was also developed to adapt to Microsoft Kinect® sensor so that players can play it using body gestures (Wikipedia, 2016).

In terms of the design, once again, it was the animated 3D objects design that has been used. Within the game, the mission is to slash through the fruity objects to earn points, the more you cut the more you would earn (ibid). However, the players also need to avoid the bombs (ibid).

2.2. Existing techniques

It has been a revolution for the industry to implement the game engines into the procedures of game making (Eberly, 2006). According to Ward (2008), utilising game engines benefits in many ways. First, it would enable the developers to focus on the details that make the game interesting instead of having to pay too much attention to the technical issues (ibid). Second, with the nature of a development framework, game engines consist of a wide range of reusable components that would make it easier to develop a game(ibid). Hence, in order to demonstrate the practicality of this assignment, during this project, a game engine would be used to implement a simple deliverable game that meets the requirements in the section above.

According to Wikipedia (“Game engine,” 2016), game engine is a framework which is optimised to aid the productivity of making game. It could be used to design a wide variety of game for almost any type of devices from mobiles to personal computers. A standard game engine would consist of several components such as the core programme, rendering engine, audio engine, physics engine (ibid). The main programme would contain the core logic of the game which usually is implemented by a collection of algorithms(ibid). Regarding the rendering engine, it would generate the graphical designs of the game using abstracted GPU unit through a range of pre-built libraries and APIs(ibid). Additionally, there is an audio engine which by its name would be responsible for the algorithms that are related to the game's sound(ibid). Last but not least, the most important part is the physic engine(ibid). It is crucial for a game to replicate the experience of the real world so that the customers can have a better understanding of the game concept (Kickmeier-Rust, 2012). The physic engine would be in charge of the interactions between the game models and objects (Wikipedia, 2016).

There are several robust and advanced game engines in the industry. However, within this dissertation, it would be Unreal, Unity, Cry Engine that get analysed and considered. Although each of them has their advantages and disadvantages, utilising either of them would probably result in a rise in productivity.

2.2.1. Unreal Development Kit (UDK)

Developed by Epic Games ®, Unreal Game Engine (or Unreal Development Kit) is one of the most popular high degree game making tools that have been using by developers all around the world(Wikipedia, 2016).

2.2.1.1. Advantages

It is a high degree game engine

Excellent art pipeline, advanced shader ability

Completely open source

Based on C++

2.2.1.2. Disadvantages

Less plugin complexity

Free only for Windows and Mac

2.2.2. Cry Engine

2.2.2.1. Advantages

A very advanced game engine with a lot of built-in plugins and component

2.2.2.2. Disadvantages

Difficult to master in comparison to UDK & Unity

Target only high-end machine (PCs, game consoles)

2.2.3. Unity-3D Engine

2.2.3.1. Advantages

Highly popular game engine

Balance between ease of use and power

Multiplatform structure

Multiple object sources importing

Enriched support from the community

Relatively free to use

2.2.3.2. Disadvantages

Challenging to use in teams

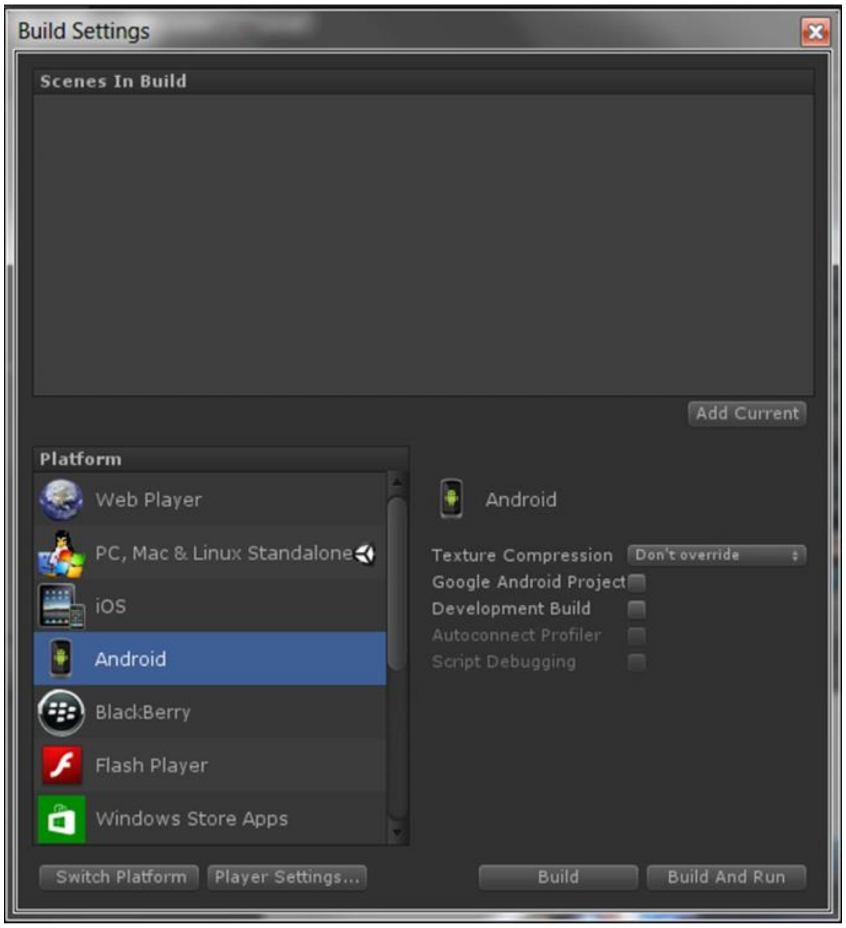
3. Methodology

3.1. Proposed method

According to the advantages and disadvantages of these above game engine, also compare against the requirements in Section 1, it has reached to a decision to use Unity Engine to implement a demonstration prototype for this project.

3.1.1. General Views of Unity Game Engine

Unity 3D was constructed for the purpose of 3D game making with great interactive context or real-time animated contents. Unity is one of the most favourite game making tool due to its flexibility in platform dependency. The editor can only run on either Mac or PC. However, the game can be export to run on Windows, Mac, iOS, Android and even game consoles.



Texture Compression: Don't override
Google Android Project
Development Build
Autoconnect Profiler
Script Debugging

In the year of 2009, within 4 years after releasing, Unity lied among the top five best game development tool (Wikipedia,2014). With a huge amount of supporting document and an active community, Unity remains one of the best engines for small and medium scale game development.

In another aspect, Unity also supports importing the pre-built objects from designing programmes such as 3DS-Max and Cinema4D so that it has the variety of customised game models.

With the Unity core component written by Joachim Ante in 2001, Unity Inc was released in 2005 at version 1.0. In the year of 2007, Unity 2.0 was released with the support for iOS devices in a small update later in 2008. In June of 2010, a new version of Unity that enables the compatibility for Android devices.

According to some independent research, Unity has been subscribed by 250.000 customers worldwide included Cartoon Network, Coca-Cola®, Disney, NASA, Microsoft and a vast number of indie developers, students.

3.1.2. Unity Features

Unity features an inherited integrated developing environment with graphical design editor, detailed object inspector and live preview (without having to build the application).

Multi-platform run-ability:

Standalone application on PCs and Macs

Web browser compatible

iOS, Android mobile application

Xbox® 360 console application

Built-in reusable and easy to maintain resource manager that would import any necessary resource into the project and keep track of the updates. It has a broad range of supported sources which varies from 3DS-Max, cloud services to Blender®.

Additionally, with version 4.0 and later, Unity also supports a new Asset Server which would act as a version management system for all the resources that would contribute in the progress of making the game.

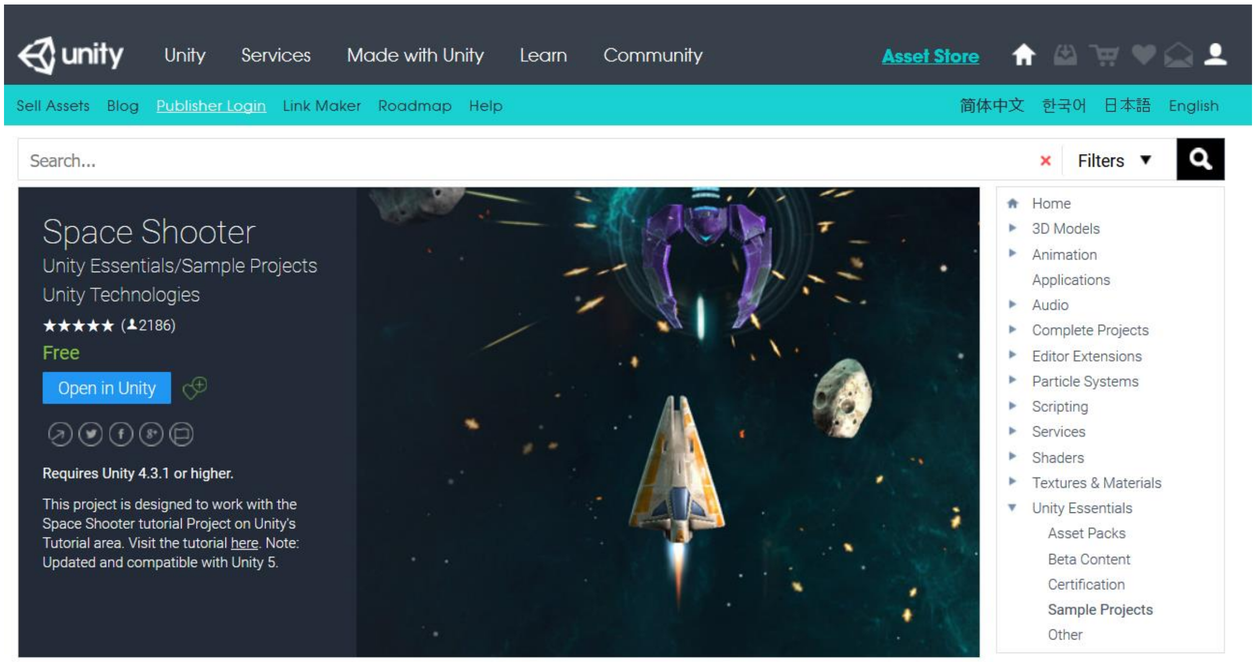
3.1.3. Unity's Components Explanations

3.1.2.1 Assets

According to Unity manual, Asset is a collection of pre-built resources that would be used in the game making process. The resource could be in the form of graphical models, pictures or sounds. When setting up the project, Unity would make a series of references to this collection to organise and manipulate the resources.

Figure 2: Unity Asset

With the support of an active community, Unity has its Asset Store which contains numerous pre-built models, soundtrack and pre-designed material that the developer could use at disposal.



Space Shooter
Unity Essentials/Sample Projects
Unity Technologies
★★★★★ (2186)
Free

Requires Unity 4.3.1 or higher.

This project is designed to work with the Space Shooter tutorial Project on Unity's Tutorial area. Visit the tutorial here. Note: Updated and compatible with Unity 5.

Figure 3: Unity Asset Store

3.1.2.2 Scenes

Within the game making process, Scene could be considered as a game screen which would be viewed directly from the player view. A Scene could be built and manipulated so that it can be switched to quickly and precisely during the gameplay.

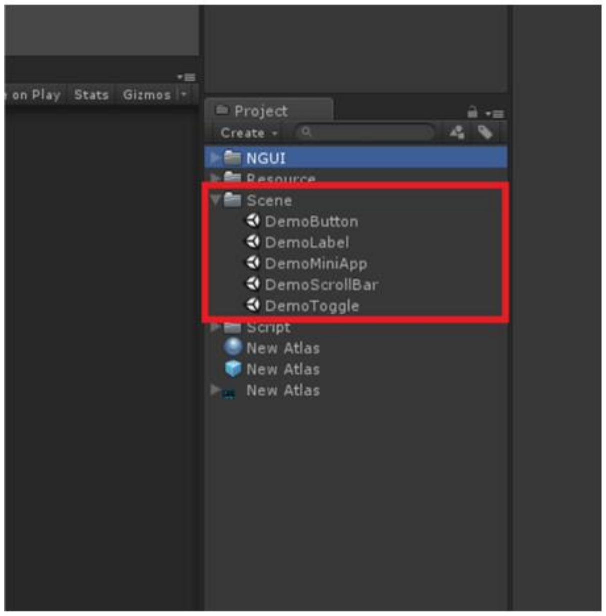
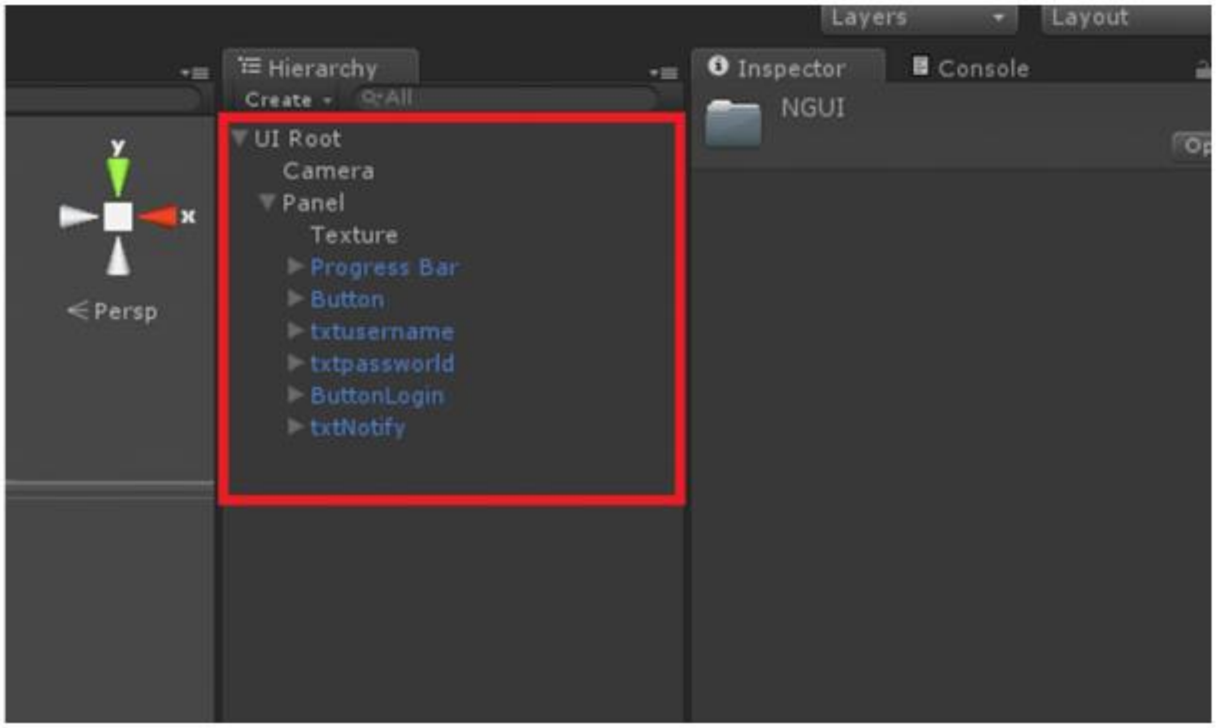


Figure 4: Scene

3.1.2.3 Game Objects

In order for a resource to be able to interact with a game scene, it has to be represented as a game object which would be considered as an object while the scene is the environment. Each and every game object in Unity consists of many components. However, they have to have a component which is called “Transform”. Transform component manipulates the rotation, position and scaling of the object.



3.1.2.4 Game Objects' Components

As mentioned above, a game object can consist of a wide range of components. These components would help define the properties of a game object and therefore, they are also attached to the object. These characteristics usually include the lighting, camera, physical effects, scripting.

3.1.2.5 Scripts

Has been considered as a most important part in the process of making game using Unity, Scripting is the backbone of the game which would link the game objects together and define a mechanism by which they behave accordingly to. Unity script could be written in C#, JavaScript or Boo(a derived language of Python). Due to the fact that C# has been a very popular programming language in the community as well as the educational environment, C# is also a powerful high-level language that has a flexibility and robustness, within this project, it would be used as the main scripting language.

In order to be recognised by the game engine, scripting files must inherit from the MonoBehavior class.

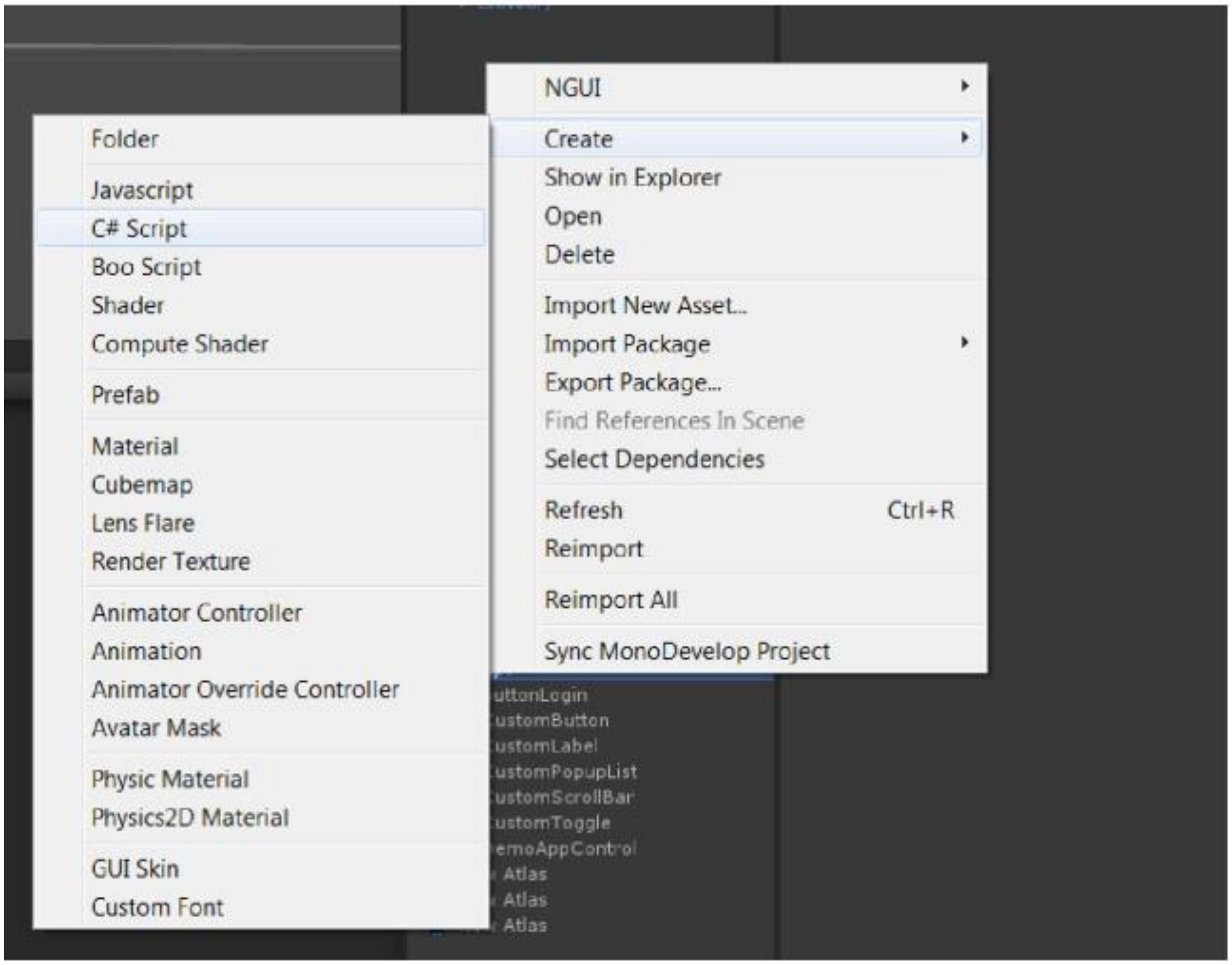
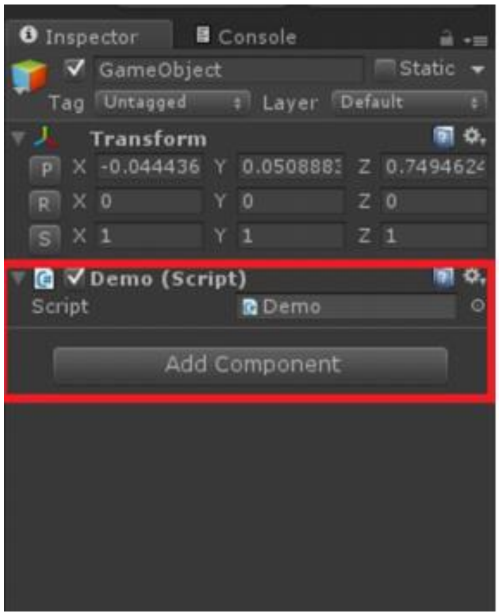


Figure 5:Scripting file



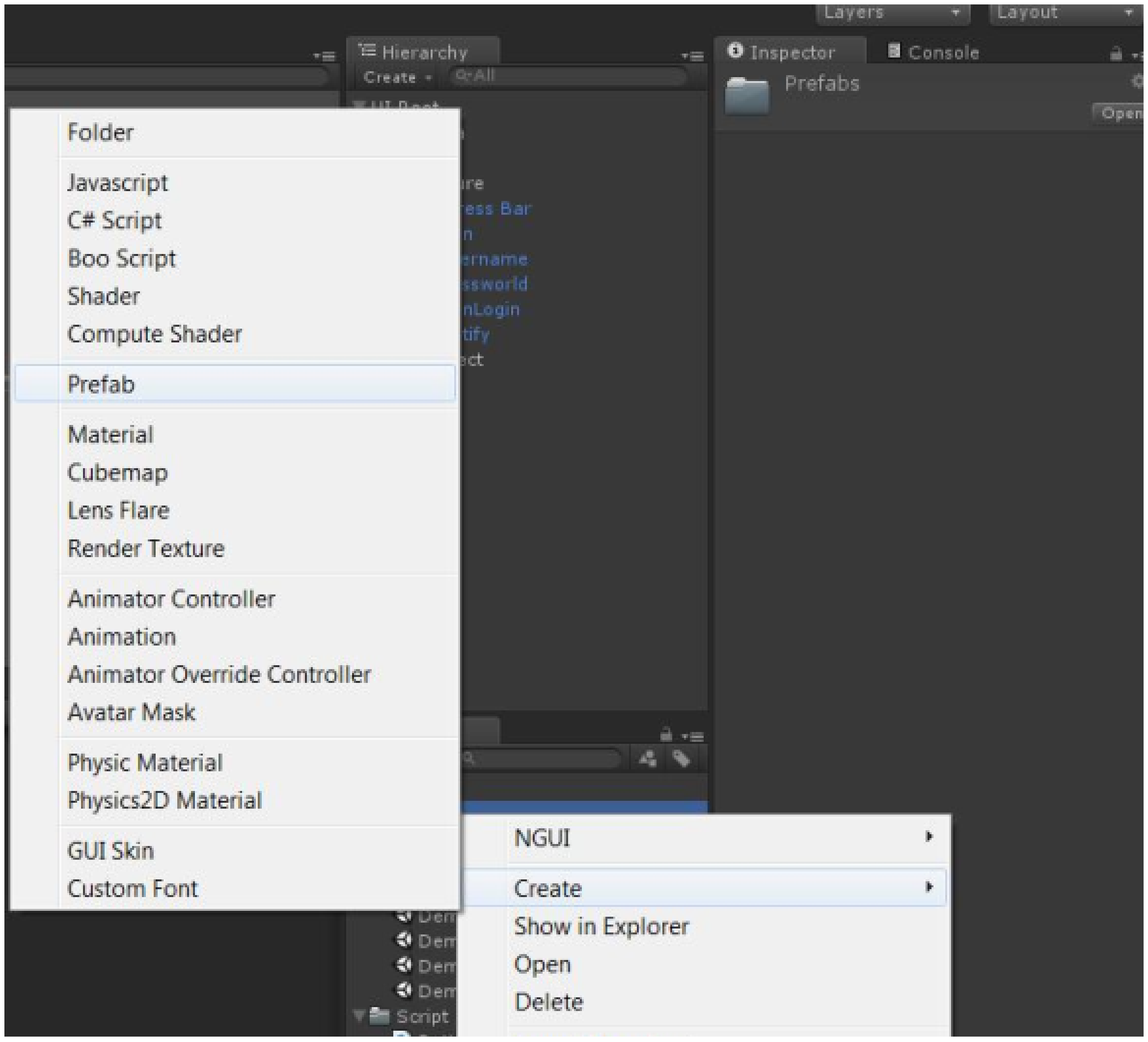
GameObject Static
Tag Untagged Layer Default
P X -0.044436 Y 0.0508883 Z 0.7494624
R X 0 Y 0 Z 0
S X 1 Y 1 Z 1
Script Demo

Additionally, for a script to be executable, it has to be attached to the game object as a component.

3.1.2.6 Prefabs

Prefab is a packaged resources pack that enable the developer to reuse the whole set of game objects along with their component configurations at the crucial moment without having to set them up again. In another word, prefab is a container in which the game objects and models are encapsulated.

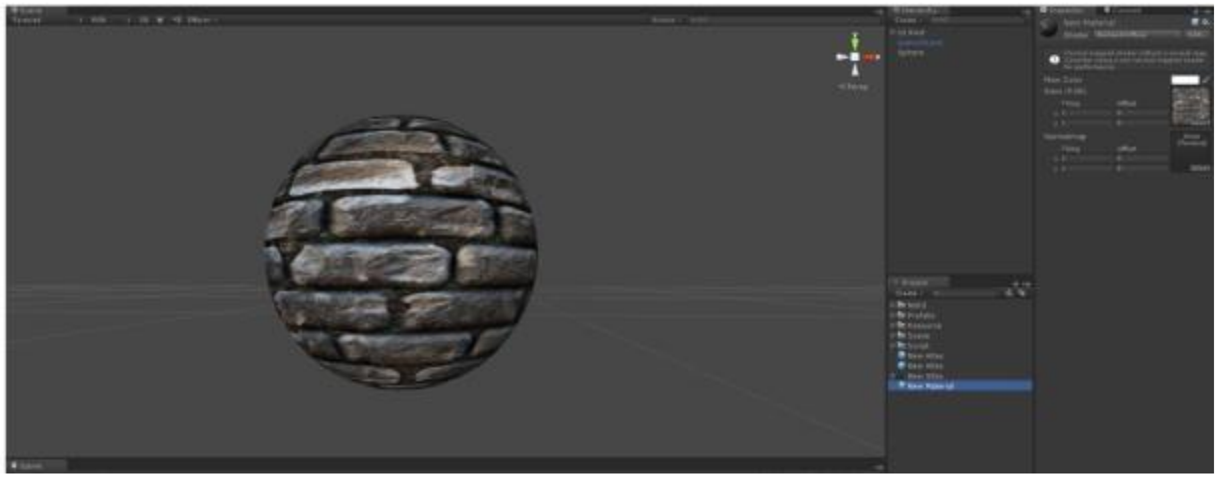
In order to create a prefab in Unity, it is possible to drag and drop a game object from the hierarchy section into the asset or right click to the object and select “Create Prefab”.



3.1.2.7 Materials & Shader

In order to render the game objects in Unity, it needs a cooperation of a few components such as Material, Shader and Texture.

The material consists of a collection of the used textures, colours on which a surface should be rendered. On another aspect, Shader could be considered as a control scripting that would determine how the illustrations of the objects should be rendered.



3.2. System design

3.2.1. Game Flow

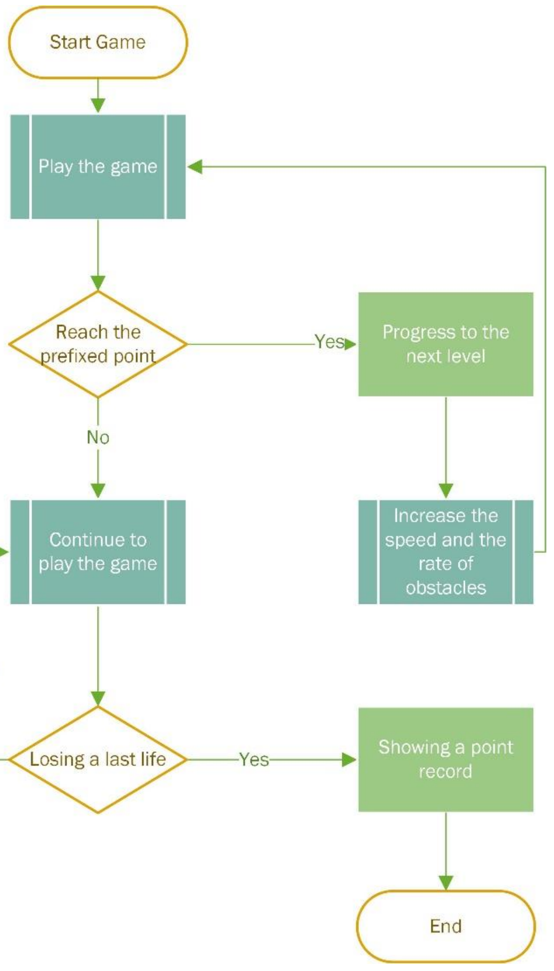
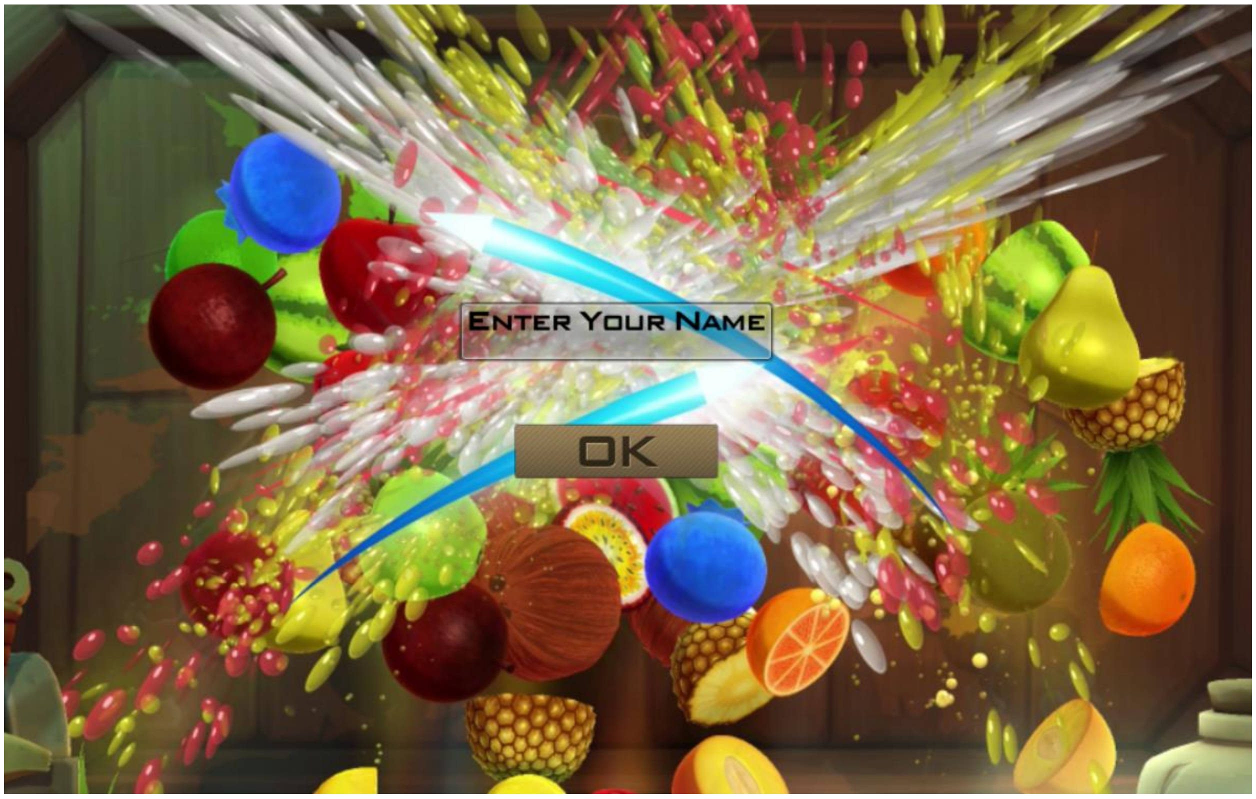


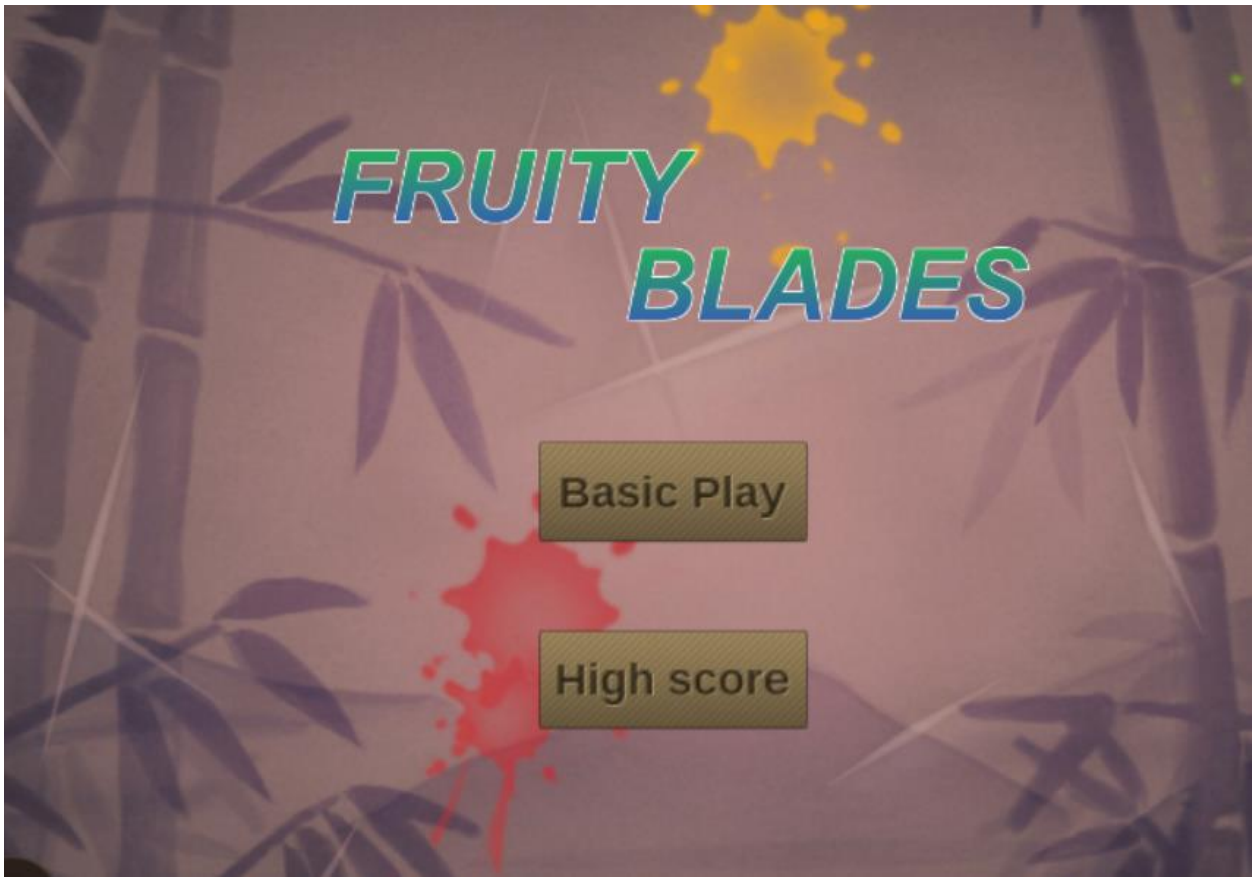
Figure 6: Game Flow Diagram

4. GUI design

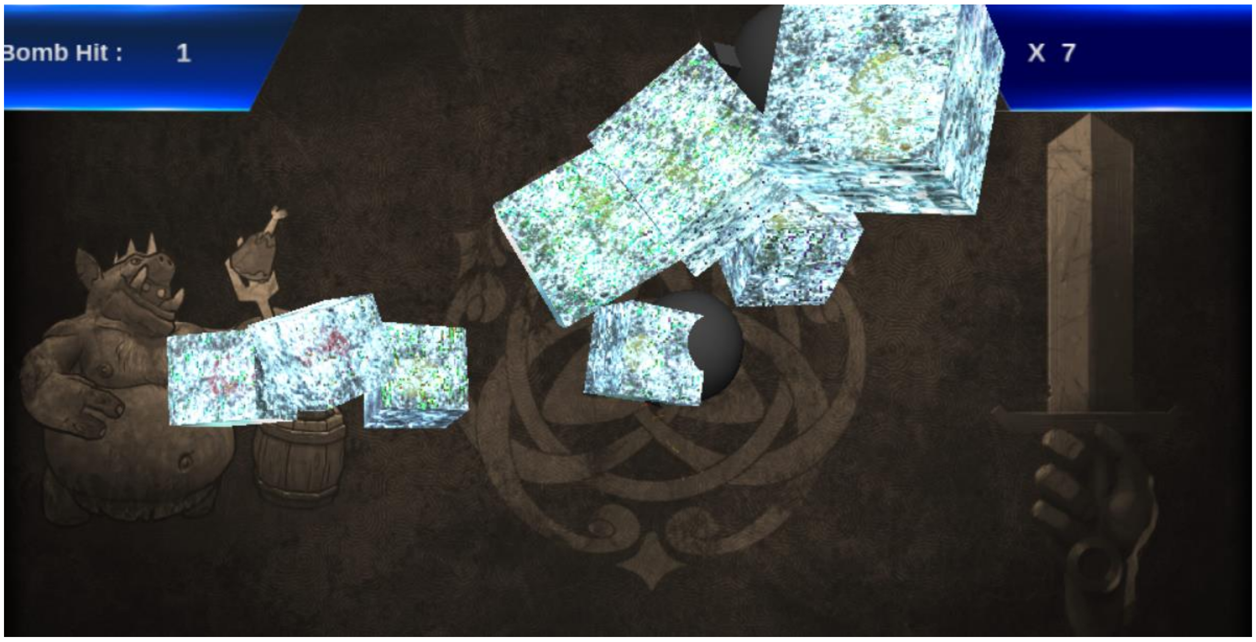
4.1. Screens Designs

4.1.1. Welcome Screen

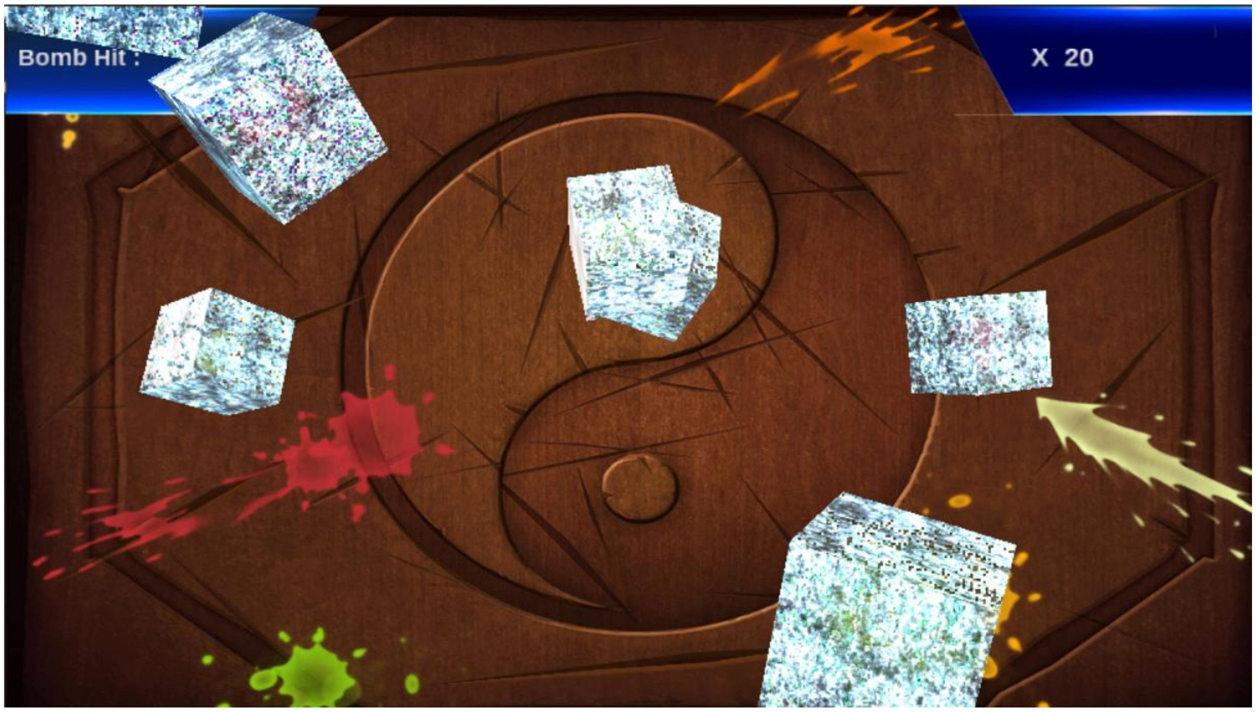




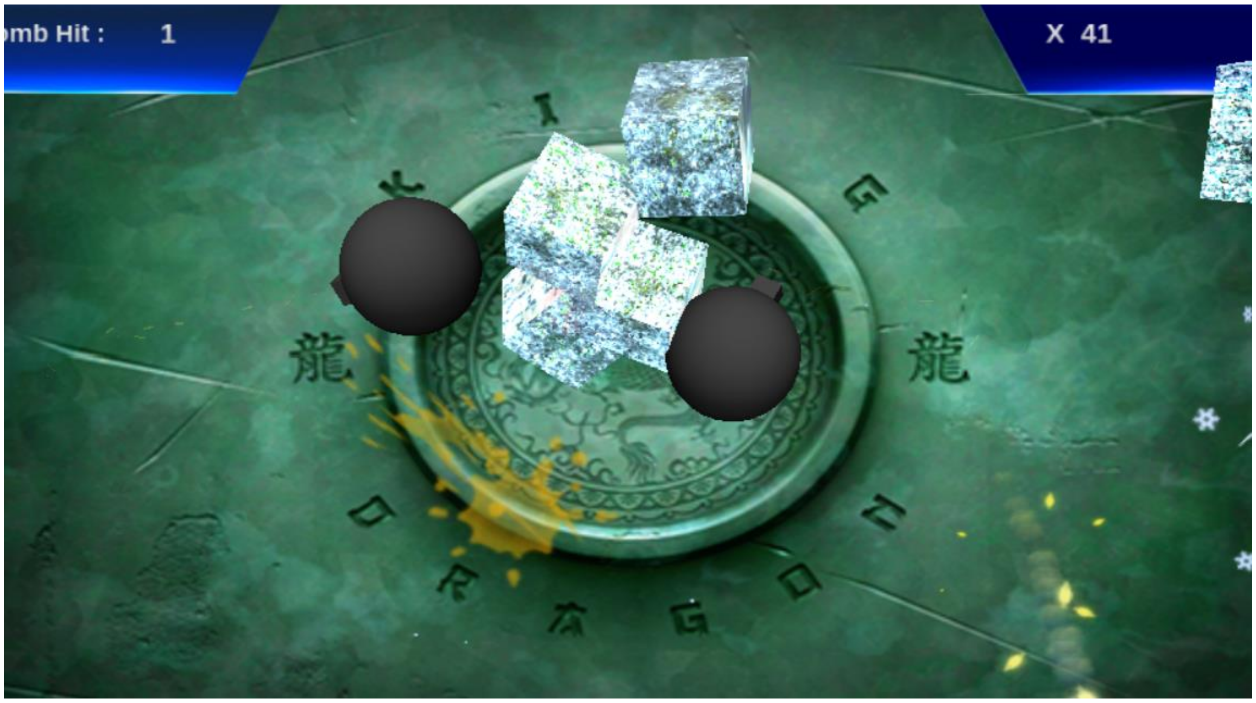
4.1.2. Level 1 Screen



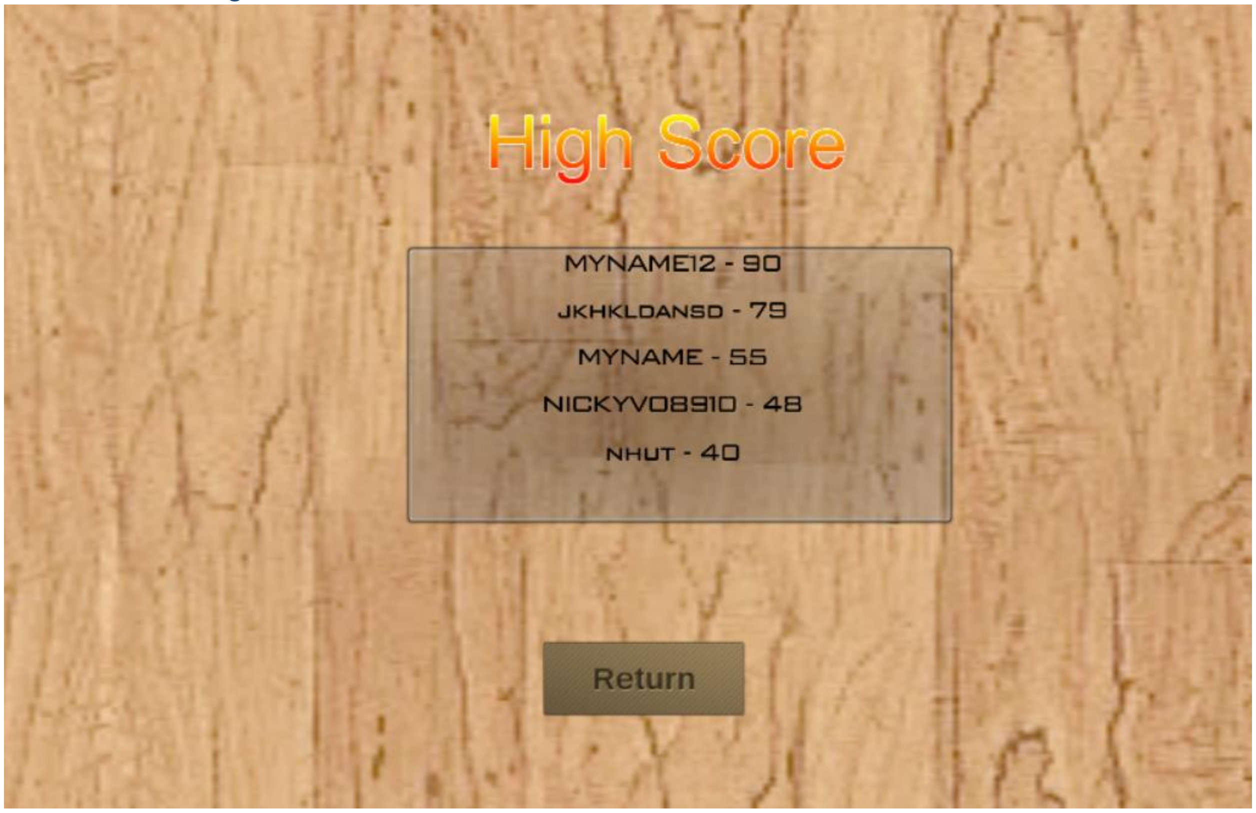
4.1.3. Level 2 Screen



4.1.4. Level 3 Screen



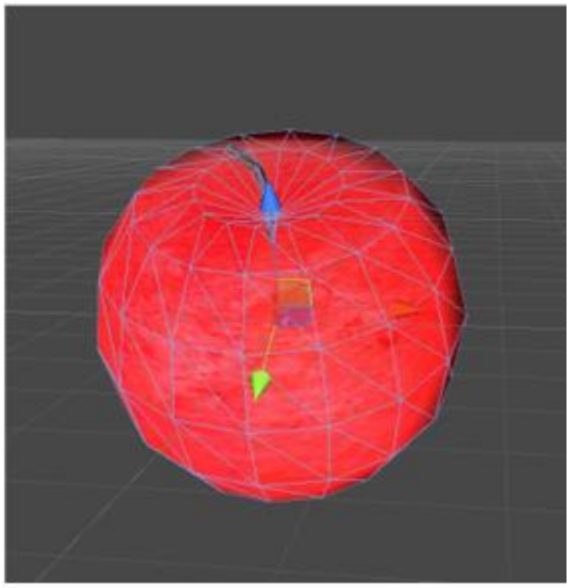
4.1.5. High score Screen



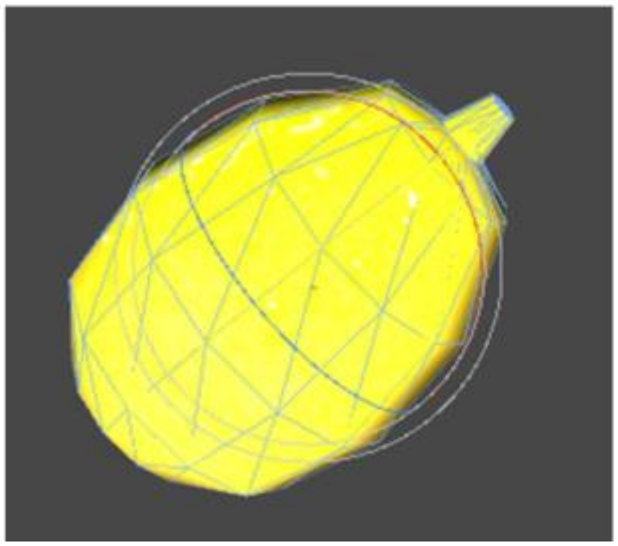
MYNAME12 - 90
JKHKLDANSD - 79
MYNAME - 55
NICKYVO8910 - 48
NHUT - 40

4.2. In-Game Objects

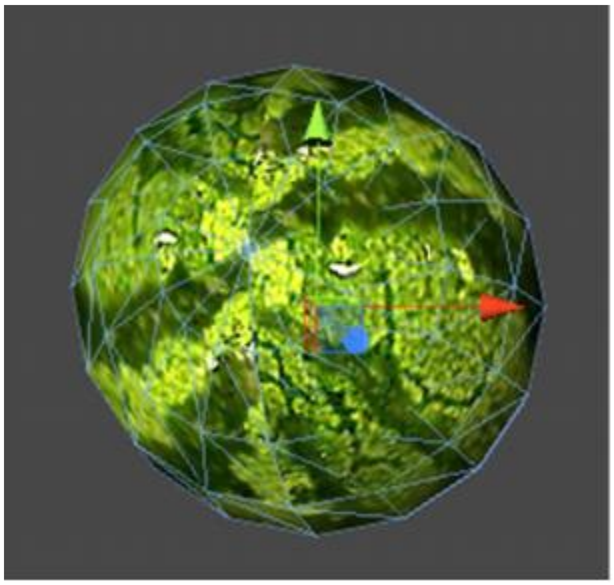
4.2.1. Apple



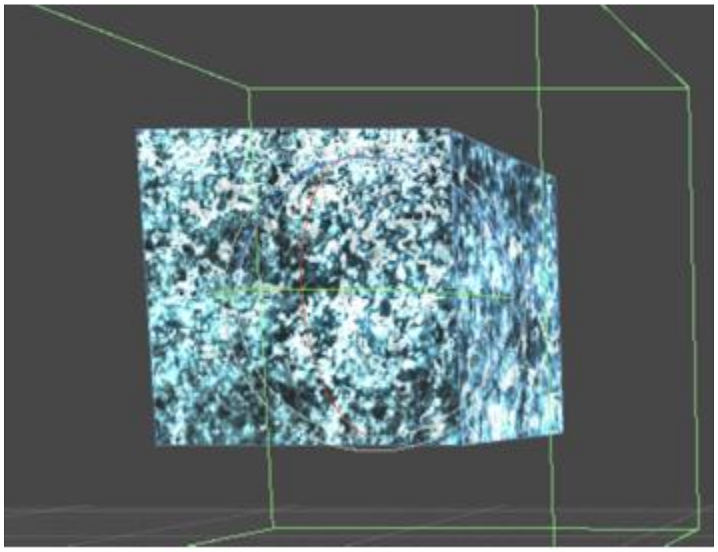
4.2.2. Lemon



4.2.3. Watermelon



4.2.4. Ice Cube



5. Implementation

5.1. Unity API Overview

5.1.1. Unity API Components

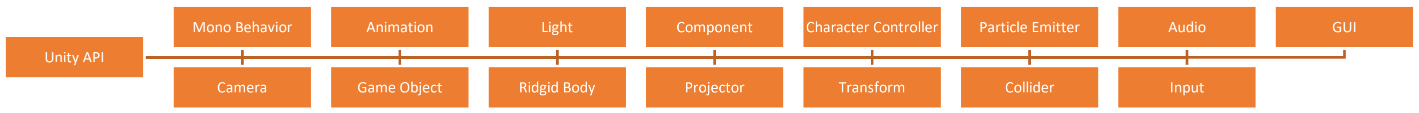


Figure 7: Unity API Diagram (Adapted from Unity (n.d))

- **GameObject:** This is considered as the parent class of all other classes in Unity, each and every scripting needs to be attached to a game object.

- **Component:** This is a class which acts as a container for others object to being attached to a Game-Object class.
- **Transform:** This would assist in determining the location, rotation and the scaling of a rendered object in the scene.
- **Input:** This class would handle all the inputs of the application which ranges from touch gesture, mouse clicks and the sensors.
- **Camera:** The camera class would be responsible for controlling the camera (the output) of the programme.
- **Light:** Along with the camera, this is another component that would help visualising the game objects to the users. This element is of particular importance because prior to being illuminated by the “Light”; everything tends to be dark and therefore, invisible.
- **Projector:** This component takes responsibility for projecting the texture onto the object surface.
- **Particle Emitter:** This particular component would aid in creating the particle effects.
- **Audio:** This would manipulate the audio component of the application.
- **Animation:** The “Animation” would act as a controller for the movements(animated) of the objects.
- **Rigidbody:** This function would replicate the physical characteristics of the object, giving it a body on which the physical forces and laws can interact.
- **Character Controller:** This would manipulate the character.
- **Collider:** This is responsible for the collision of the game objects.
- **GUI:** This particular module is specialised in creating the User Interface.

5.1.1.1. Mono Behaviour

This could be considered as the backbone of the scripting scheme in Unity, each and every programming script has to be an extent of this particular class.

5.1.2. Unity App Life Circle

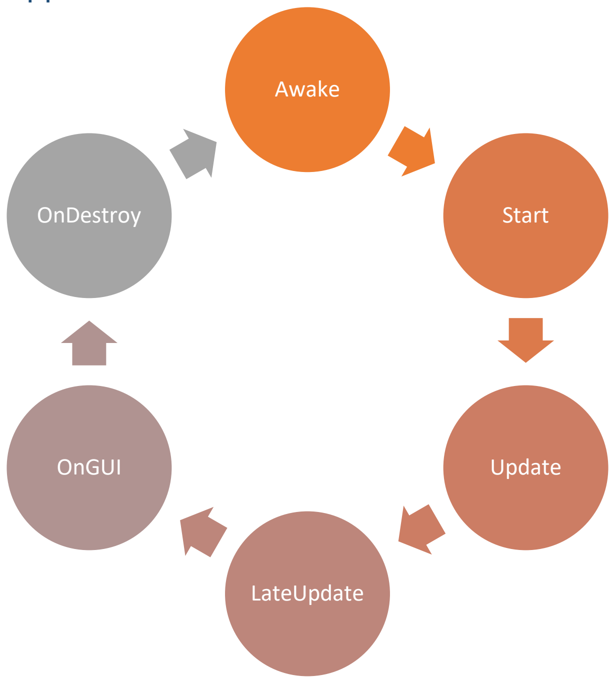


Figure 8: Script Life Circle (Adapted from Unity (n.d))

A Unity Object would traverse through a series of events such as:

- Awake: This would happen when the object has just been initialised, and the script has just been loaded.
- Start: This would be the next state if none of the script components disabled.
- Update: The is the most common state that has been utilised in the scripting process.

This event would be called on each frame load of the application.

- LateUpdate: This usually comes after the Update() event, however, this particular event is optional, it does not have to run all the time. Hence it would be useful for some abrupt change in the application.
- OnGui: This state would be called at every frame for the purpose of constructing a smooth and interactive user interface.
- OnDestroy: This would be the end of the life circle as it would be called when the script is eliminated from the memory.

-Above are the general view about Unity Framework and how it works in the game making process. In the next section, it would be the issues that occurred in the process and the proposed solution for them.

5.2. Implementation Issues & Solutions

5.2.1. Selection Handling

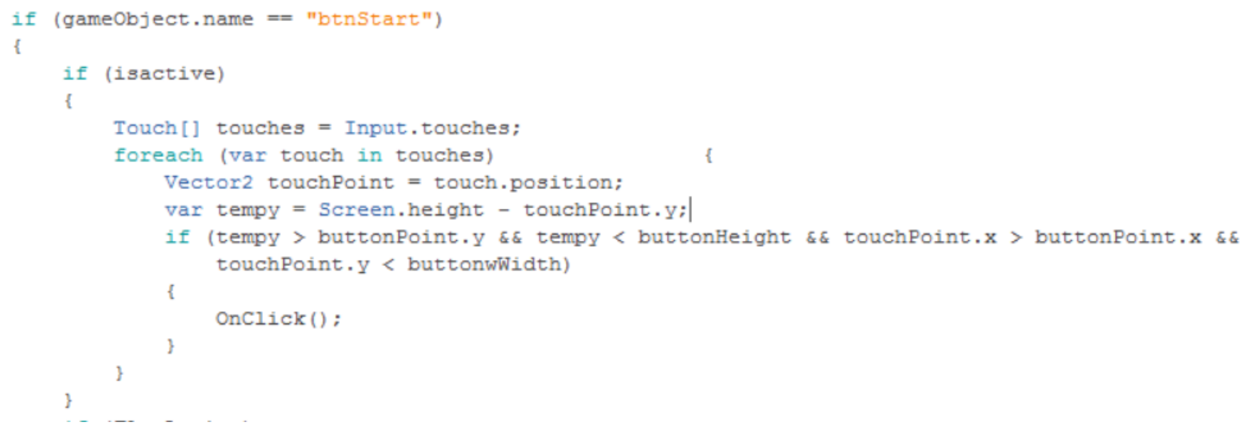
The first necessary thing to do in order to have the touch/ selection gesture recognised by the system is to register an EventDelegate for the GameObject.

public UIButton ButtonStart;

EventDelegate.Add(ButtonStart.onClick,()=>{ StartGame();});

When the delegate has been set, whenever there is a click on that button, the StartGame method would also be called.

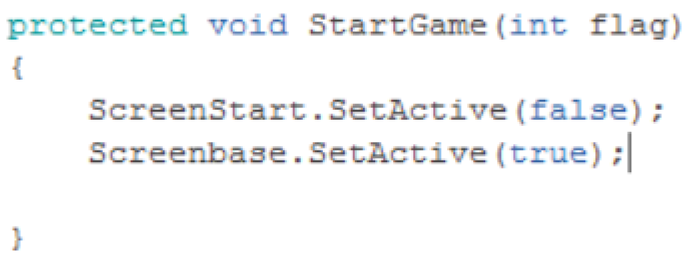
However, it would be different on mobile devices since it does not have a mouse cursor. Consequently, another method of determining the selection must be used for touch-enabled devices. At this moment, the touch gesture would be registered when the touch is located within the coordinates of the button(GameObject).



```
if (gameObject.name == "btnStart")
{
    if (isactive)
    {
        Touch[] touches = Input.touches;
        foreach (var touch in touches)              {
            Vector2 touchPoint = touch.position;
            var tempy = Screen.height - touchPoint.y;
            if (tempy > buttonPoint.y && tempy < buttonHeight && touchPoint.x > buttonPoint.x &&
                touchPoint.y < buttonwWidth)
            {
                OnClick();
            }
        }
    }
```

5.2.2. Scenes Managing

As mentioned in 4.1, this game is made of many scenes. Therefore, it is crucial to make sure the scenes can be switched correctly during the play. Within Unity, “isActive” is a property of which determined whether the scenes (as a GameObject) to be visible or not. Additionally, along with such property, there is a method to control its value as

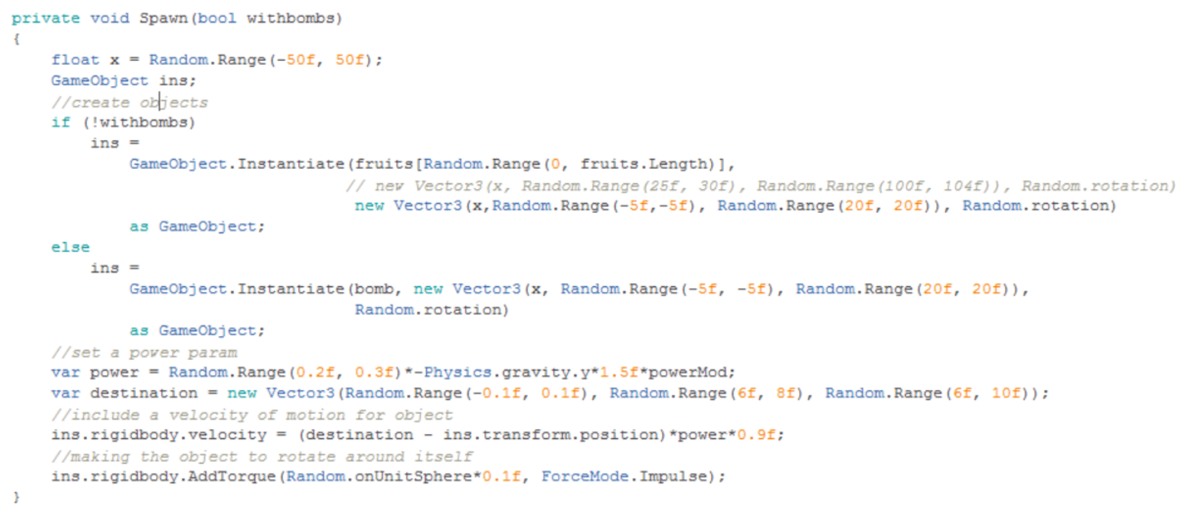


```
protected void StartGame(int flag)
{
    ScreenStart.SetActive(false);
    Screenbase.SetActive(true);


}
```

following.

5.2.3. Fruit Dispenser

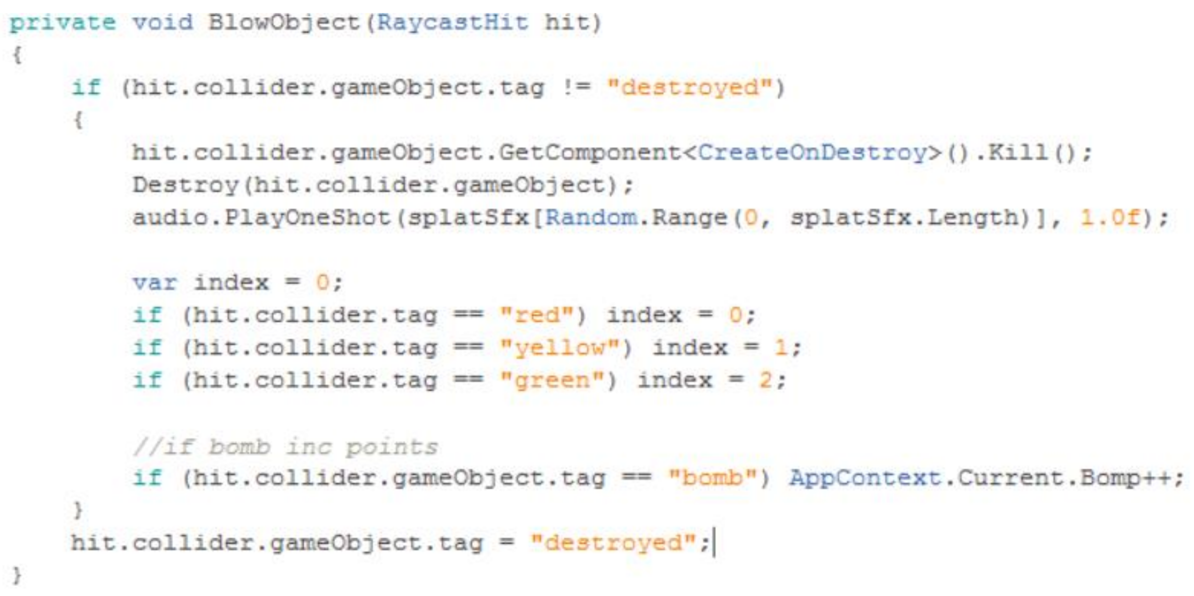
During the game, the fruits and the bombs (GameObjects) would be thrown onto the screen (toward the player). The following code snippet would first create a game object and then add in a velocity so that the objects can move through the 3D space.



```
private void Spawn(bool withbombs)
{
    float x = Random.Range(-50f, 50f);
    GameObject ins;
    //create objects
    if (!withbombs)
        ins =
            GameObject.Instantiate(fruits[Random.Range(0, fruits.Length)],
                                      // new Vector3(x, Random.Range(25f, 30f), Random.Range(100f, 104f)), Random.rotation)
                                       new Vector3(x,Random.Range(-5f,-5f), Random.Range(20f, 20f)), Random.rotation)
            as GameObject;
    else
        ins =
            GameObject.Instantiate(bomb, new Vector3(x, Random.Range(-5f, -5f), Random.Range(20f, 20f)),
                                       Random.rotation)
            as GameObject;
    //set a power param
    var power = Random.Range(0.2f, 0.3f)*-Physics.gravity.y*1.5f*powerMod;
    var destination = new Vector3(Random.Range(-0.1f, 0.1f), Random.Range(6f, 8f), Random.Range(6f, 10f));
    //include a velocity of motion for object
    ins.rigidbody.velocity = (destination - ins.transform.position)*power*0.9f;
    //making the object to rotate around itself
    ins.rigidbody.AddTorque(Random.onUnitSphere*0.1f, ForceMode.Impulse);
}
```

5.2.4. Collision Handling

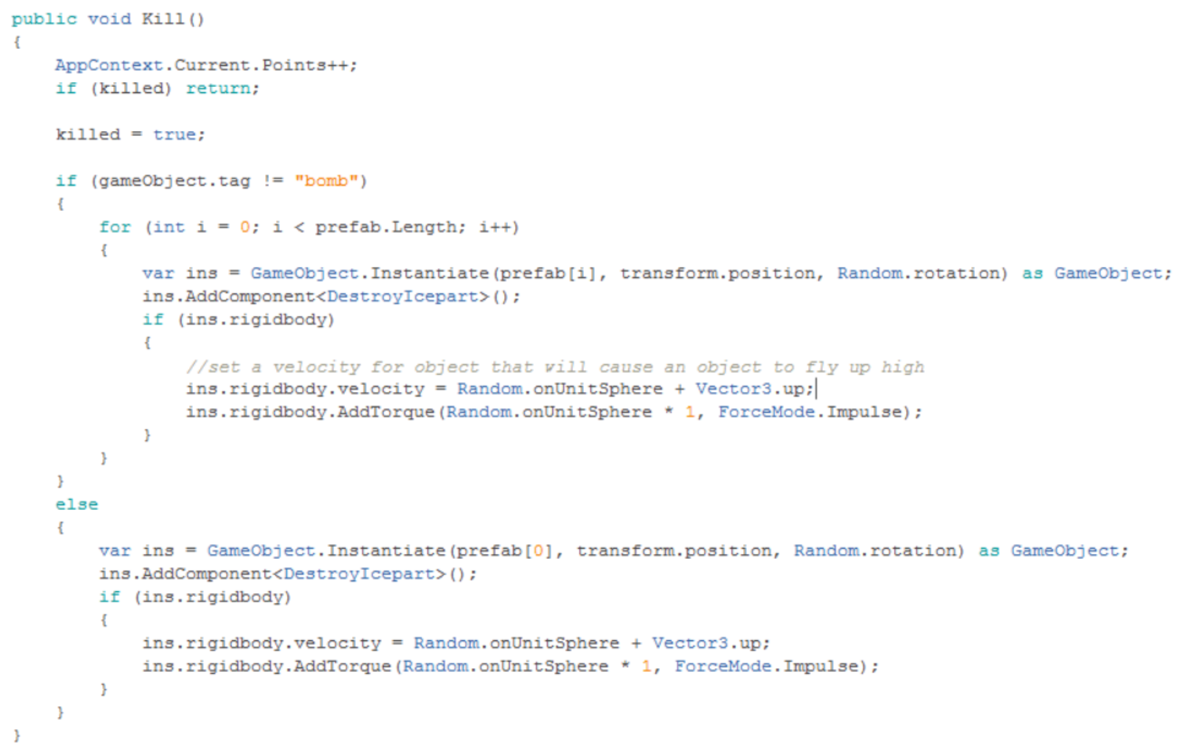
In order to have the game functional, it is crucial to handle the collisions between GameObjects in the 3D space. In terms of the ice cubes, each of them is covered by a Box Collider so whenever there is a collision of any object to the collider area, an OnCollision event would be thrown. More specifically, within this game, the OnCollision event would call the "Kill" method to detonate an ice cube and add the point to the total one. However, if it was a bomb that had been hit, it would also be counted.



```
private void BlowObject(RaycastHit hit)
{
    if (hit.collider.gameObject.tag != "destroyed")
    {
        hit.collider.gameObject.GetComponent<CreateOnDestroy>().Kill();
        Destroy(hit.collider.gameObject);
        audio.PlayOneShot(splatSfx[Random.Range(0, splatSfx.Length)], 1.0f);

        var index = 0;
        if (hit.collider.tag == "red") index = 0;
        if (hit.collider.tag == "yellow") index = 1;
        if (hit.collider.tag == "green") index = 2;

        //if bomb inc points
        if (hit.collider.gameObject.tag == "bomb") AppContext.Current.Bomp++;
    }
    hit.collider.gameObject.tag = "destroyed";
}
```

5.2.5. Explosion Effects

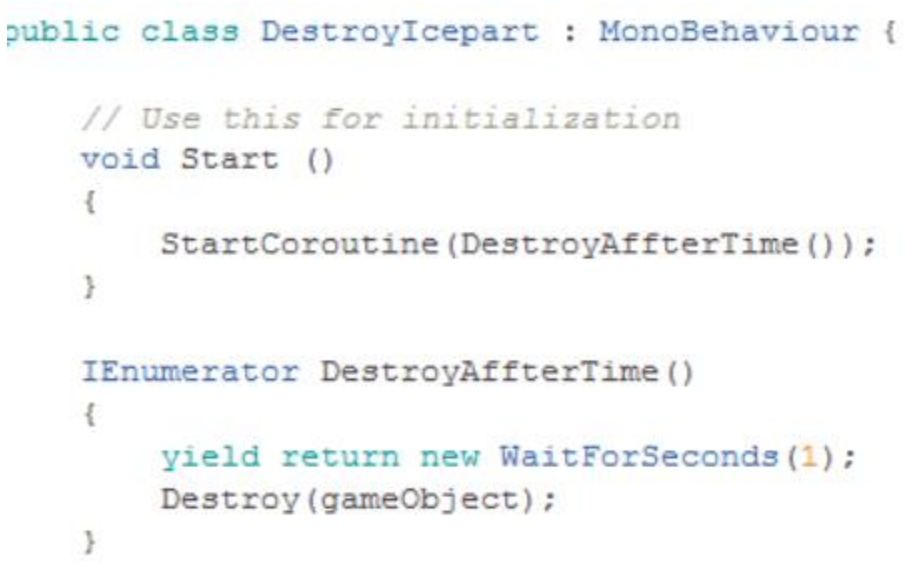
Occasionally, as the GameObject being hit, that object should act accordingly (tearing apart). The following code snippet would focus on dealing with such a problem. More specifically, the hit object would be torn into pieces and fly up for a distance as it already had a momentum at the collision moment, later on, the pieces would also be disposed.



```
public void Kill()
{
    AppContext.Current.Points++;
    if (killed) return;

    killed = true;

    if (gameObject.tag != "bomb")
    {
        for (int i = 0; i < prefab.Length; i++)
        {
            var ins = GameObject.Instantiate(prefab[i], transform.position, Random.rotation) as GameObject;
            ins.AddComponent<DestroyIcepart>();
            if (ins.rigidbody)
            {
                //set a velocity for object that will cause an object to fly up high
                ins.rigidbody.velocity = Random.onUnitSphere + Vector3.up;
                ins.rigidbody.AddTorque(Random.onUnitSphere * 1, ForceMode.Impulse);
            }
        }
    }
    else
    {
        var ins = GameObject.Instantiate(prefab[0], transform.position, Random.rotation) as GameObject;
        ins.AddComponent<DestroyIcepart>();
        if (ins.rigidbody)
        {
            ins.rigidbody.velocity = Random.onUnitSphere + Vector3.up;
            ins.rigidbody.AddTorque(Random.onUnitSphere * 1, ForceMode.Impulse);
        }
    }
}
```

5.2.6. Left over parts Disposing

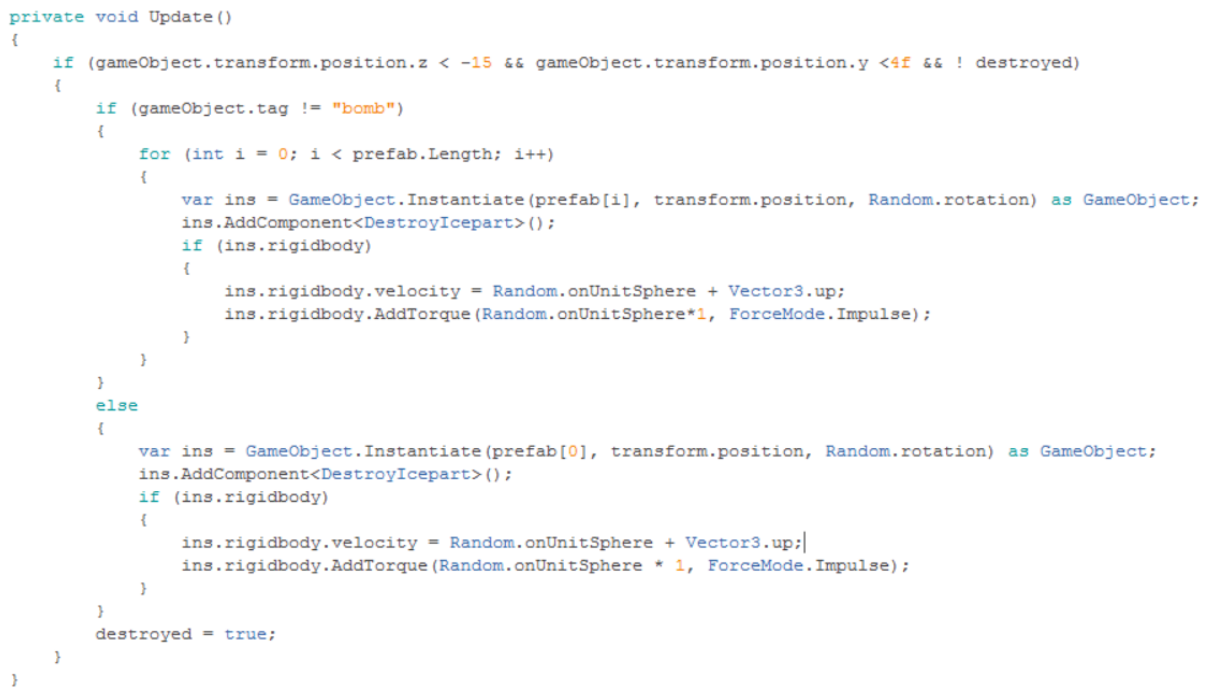
As mentioned in the previous section, the left over pieces of the hit objects should probably be disposed. There is a number of reason for this issue to become one of the most important ones within the game. First of all, every part in the game would be treated as GameObjects which has its own allocated memory, computing resources, etc. Therefore, having too many of them could lead to a negative impact on the overall performances of the system. Second, without being killed, those leftover parts would be flying around the scene and blocking the gamer view. Hence, it would somehow reduce the user experience. The following code would be dedicated to handle this type of problem.



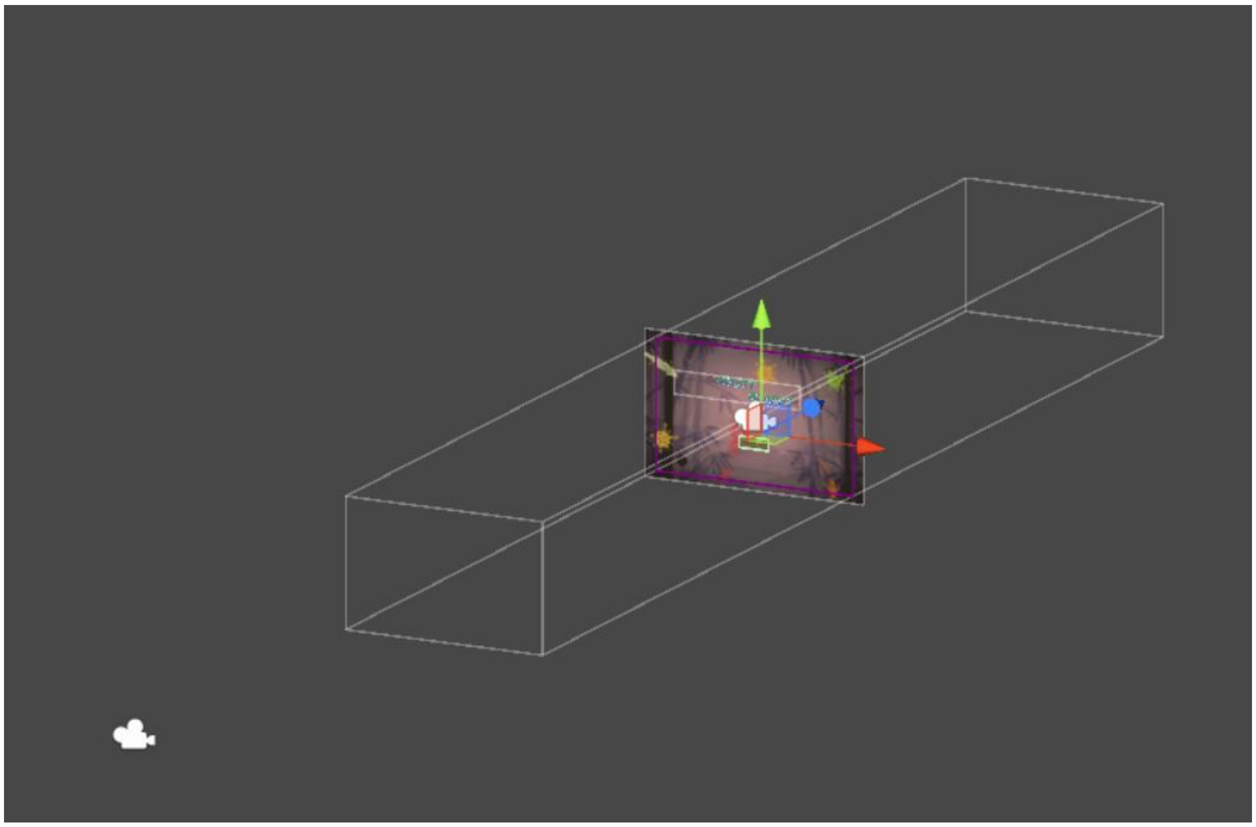
```
public class DestroyIcepart : MonoBehaviour {

    // Use this for initialization
    void Start ()
    {
        StartCoroutine(DestroyAffterTime());
    }

    IEnumerator DestroyAffterTime()
    {
        yield return new WaitForSeconds(1);
        Destroy(gameObject);
    }
```

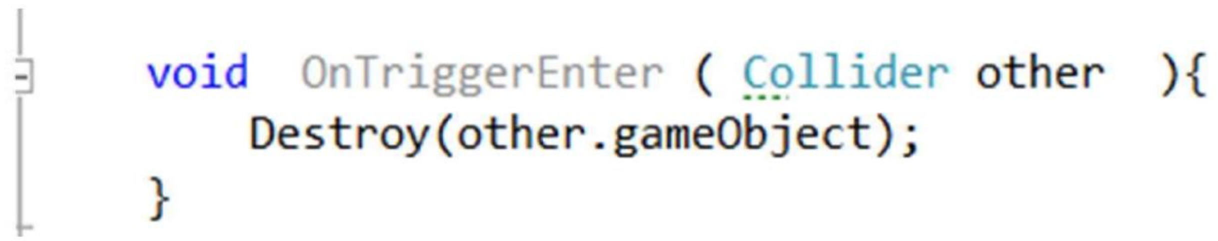
5.2.7. Invisible and Redundant Objects Handling

Due to the fact that the fruits and the bombs would be dispensed at random positions, not all of them fell into the camera angle and the user's view. One more times, the concerns that mentioned in section 5.2.6 has been raised. Consequently, it would probably the best to have the redundant objects which moved too close to the camera or outside of the camera view to be destroyed. The proposed solution is to have some reference planes and the camera itself to limit the volume in which the objects are allowed to be existed. In case any of the object touches these boundaries, it would be killed.



```
private void Update()
{
    if (gameObject.transform.position.z < -15 && gameObject.transform.position.y <4f && ! destroyed)
    {
        if (gameObject.tag != "bomb")
        {
            for (int i = 0; i < prefab.Length; i++)
            {
                var ins = GameObject.Instantiate(prefab[i], transform.position, Random.rotation) as GameObject;
                ins.AddComponent<DestroyIcepart>();
                if (ins.rigidbody)
                {
                    ins.rigidbody.velocity = Random.onUnitSphere + Vector3.up;
                    ins.rigidbody.AddTorque(Random.onUnitSphere*1, ForceMode.Impulse);
                }
            }
        }
        else
        {
            var ins = GameObject.Instantiate(prefab[0], transform.position, Random.rotation) as GameObject;
            ins.AddComponent<DestroyIcepart>();
            if (ins.rigidbody)
            {
                ins.rigidbody.velocity = Random.onUnitSphere + Vector3.up;
                ins.rigidbody.AddTorque(Random.onUnitSphere * 1, ForceMode.Impulse);
            }
        }
        destroyed = true;
    }
}
```





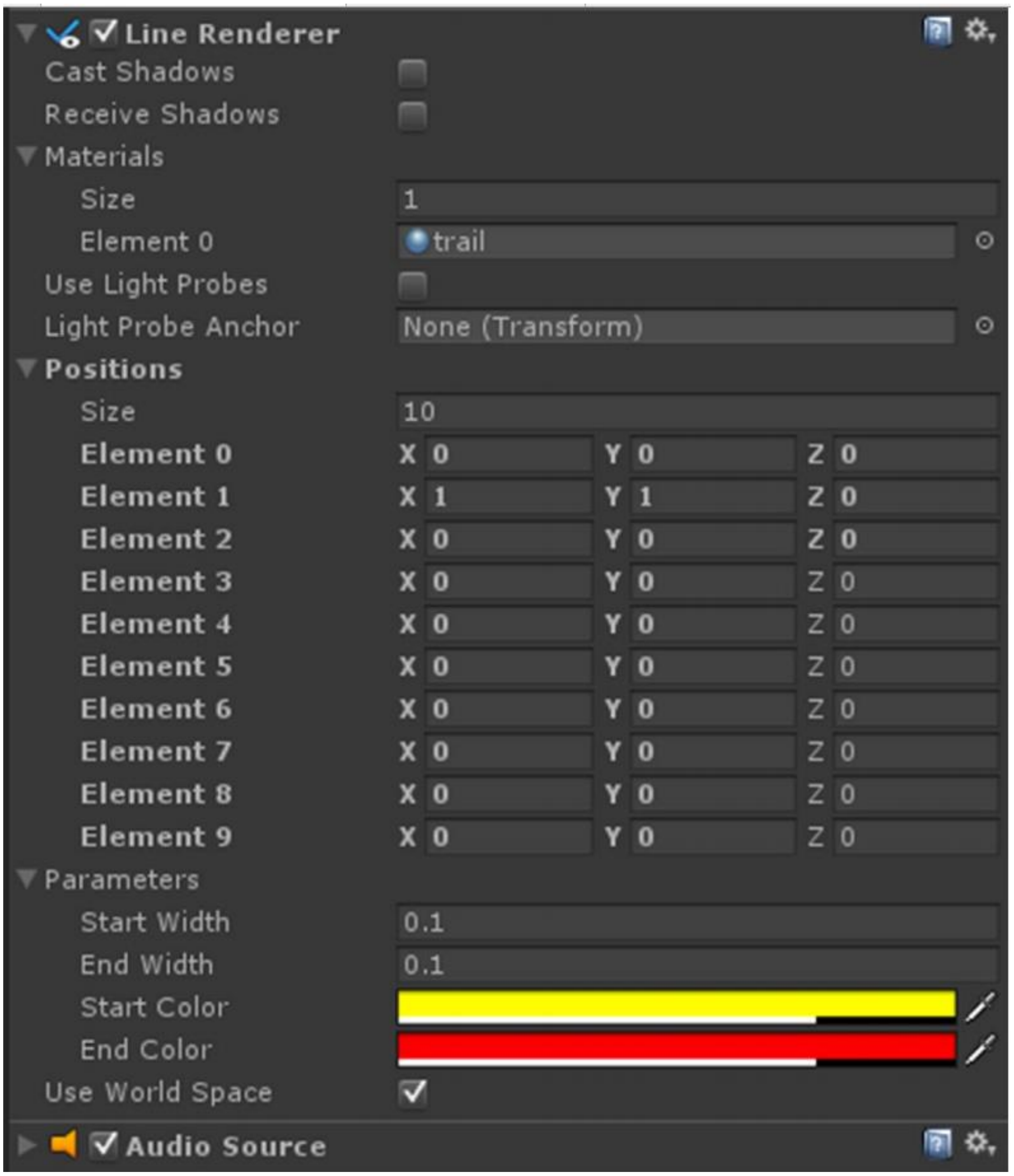
```
void  OnTriggerEnter ( Collider other  ){
    Destroy(other.gameObject);
}
```

5.2.8. Multi-touches Handling

As this project has been aiming for a multi-platform game, therefore, it is crucial to ensure that the game works well on both PC and mobile devices. Whilst being played on PCs, the cutting trails would be determined by the mouse movements. Every time the gamer clicks on the screen, the coordinates would be recorded and then in the LateUpdate event, a “LineRenderer” would be drew with those points. However, it would be slightly different on mobile devices as instead of using a single mouse cursor, the users tend to use 2 or more fingers to play the game. Therefore, multi touches recognition should be another factor that needed consideration in terms of enhancing user experiences.

For this particular situation, the proposed solution is to use the cutting trail as a Prefab that would be used to mark the touches. Specifically, each of them would have a property called fingerID.

At this moment, a new List would be created to store the touches' information accordingly.

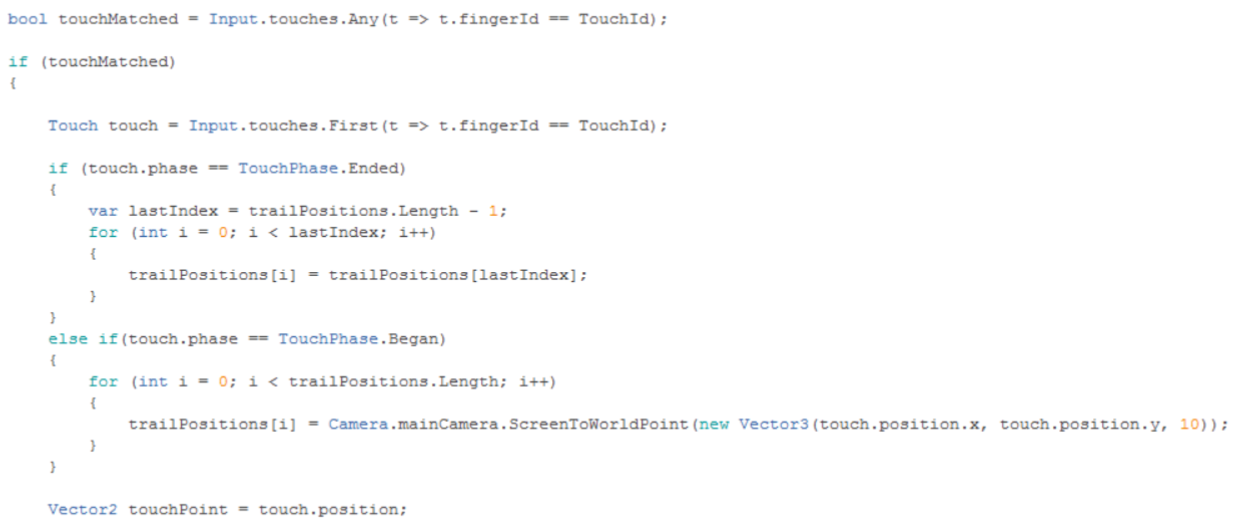


Line Renderer ✓

Cast Shadows	☐
Receive Shadows	☐
Materials	
Size	1
Element 0	trail
Use Light Probes	☐
Light Probe Anchor	None (Transform)
Positions	
Size	10
Element 0	X 0 Y 0 Z 0
Element 1	X 1 Y 1 Z 0
Element 2	X 0 Y 0 Z 0
Element 3	X 0 Y 0 Z 0
Element 4	X 0 Y 0 Z 0
Element 5	X 0 Y 0 Z 0
Element 6	X 0 Y 0 Z 0
Element 7	X 0 Y 0 Z 0
Element 8	X 0 Y 0 Z 0
Element 9	X 0 Y 0 Z 0
Parameters	
Start Width	0.1
End Width	0.1
Start Color	🟨
End Color	🟥
Use World Space	✓

Audio Source ✓

Figure 9: Trail Prefab Object



```
bool touchMatched = Input.touches.Any(t => t.fingerId == TouchId);

if (touchMatched)
{

    Touch touch = Input.touches.First(t => t.fingerId == TouchId);

    if (touch.phase == TouchPhase.Ended)
    {
        var lastIndex = trailPositions.Length - 1;
        for (int i = 0; i < lastIndex; i++)
        {
            trailPositions[i] = trailPositions[lastIndex];
        }
    }
    else if(touch.phase == TouchPhase.Began)
    {
        for (int i = 0; i < trailPositions.Length; i++)
        {
            trailPositions[i] = Camera.mainCamera.ScreenToWorldPoint(new Vector3(touch.position.x, touch.position.y, 10));
        }
    }

    Vector2 touchPoint = touch.position;
```

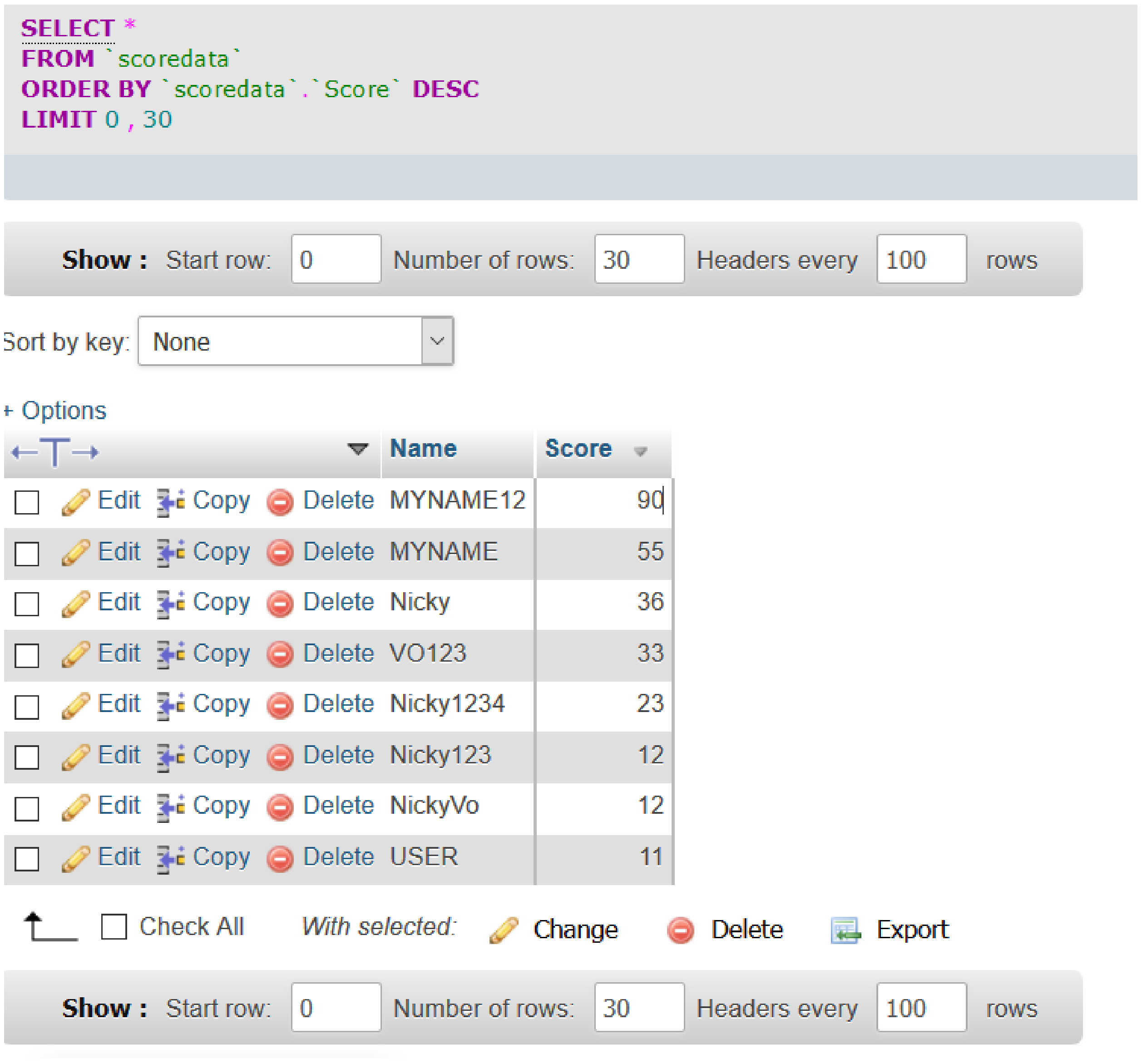
Figure 10: Creating a touch trail

5.2.9. High score Handling

In order to have the game run across the platforms, it is necessary to store the data platform-independent as well. At this moment the proposed solution would be using the web-based service dues to its versatility.

5.2.9.1 Web Service

In the external web server, a MySQL database would be used to store the players' names and their scores.



```
SELECT *
FROM `scoredata`
ORDER BY `scoredata`.`Score` DESC
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows
Sort by key: None
+ Options

	Name	Score
Edit Copy Delete	MYNAME12	90
Edit Copy Delete	MYNAME	55
Edit Copy Delete	Nicky	36
Edit Copy Delete	VO123	33
Edit Copy Delete	Nicky1234	23
Edit Copy Delete	Nicky123	12
Edit Copy Delete	NickyVo	12
Edit Copy Delete	USER	11

Check All
With selected:
Change
Delete
Export
Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Additionally, this web service should also have some functions such as:

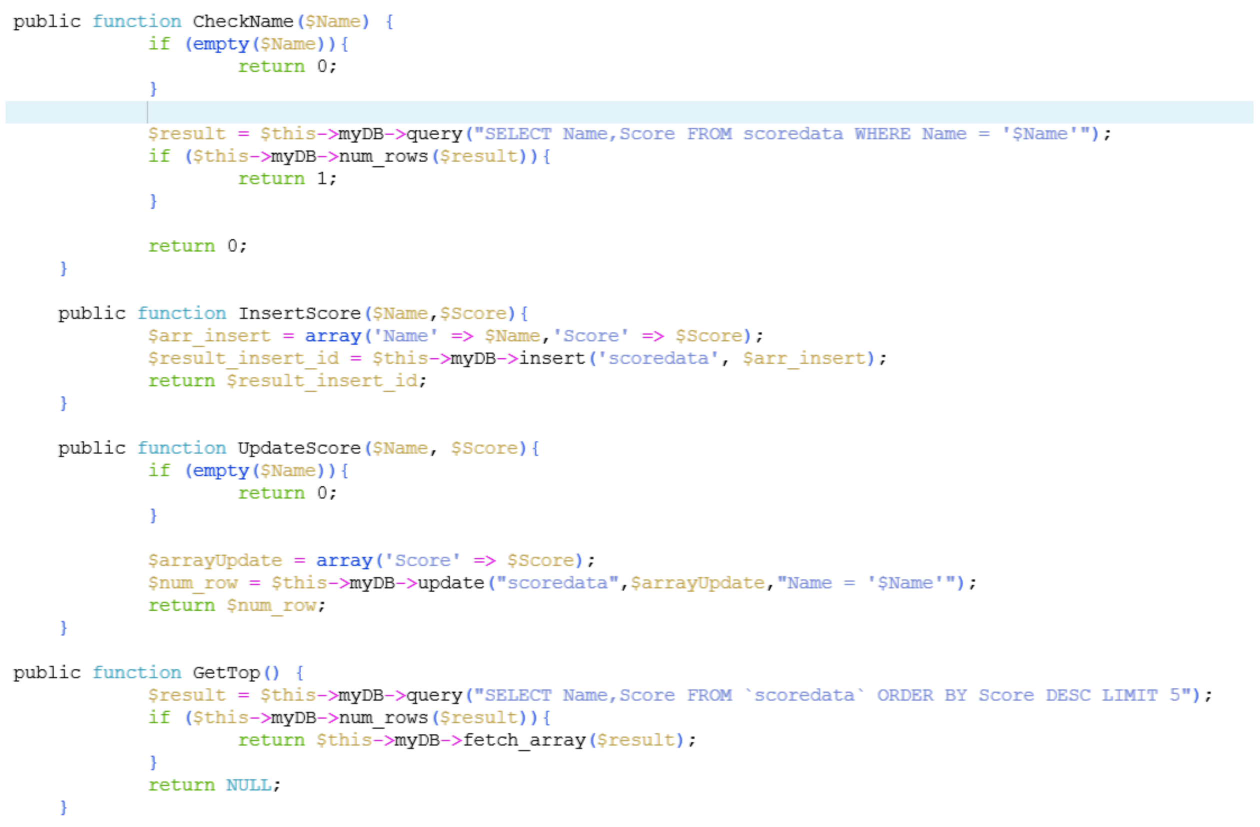
-Checking the existing name

-Insert a new record with the given name and score

-Update an existing record with the new score

-Export a list of top players

These requirements would be fulfilled by the following code snippet:

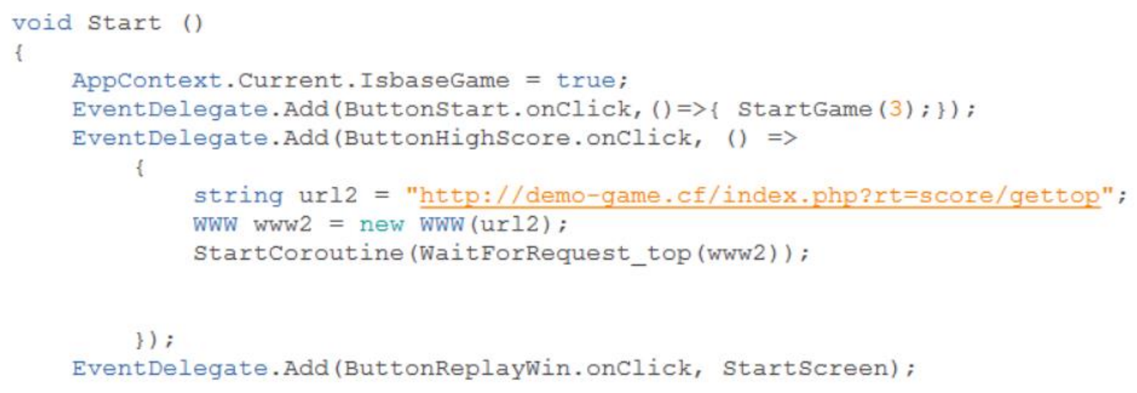


```
public function CheckName($Name) {
            if (empty($Name)){
                    return 0;
            }

            $result = $this->myDB->query("SELECT Name,Score FROM scoredata WHERE Name = '$Name'");
            if ($this->myDB->num_rows($result)){
                    return 1;
            }

            return 0;
    }

    public function InsertScore($Name,$Score){
            $arr_insert = array('Name' => $Name,'Score' => $Score);
            $result_insert_id = $this->myDB->insert('scoredata', $arr_insert);
            return $result_insert_id;
    }

    public function UpdateScore($Name, $Score){
            if (empty($Name)){
                    return 0;
            }

            $arrayUpdate = array('Score' => $Score);
            $num_row = $this->myDB->update("scoredata",$arrayUpdate,"Name = '$Name'");
            return $num_row;
    }

public function GetTop() {
            $result = $this->myDB->query("SELECT Name,Score FROM `scoredata` ORDER BY Score DESC LIMIT 5");
            if ($this->myDB->num_rows($result)){
                    return $this->myDB->fetch_array($result);
            }
            return NULL;
    }
```

5.2.9.2 In App Handling

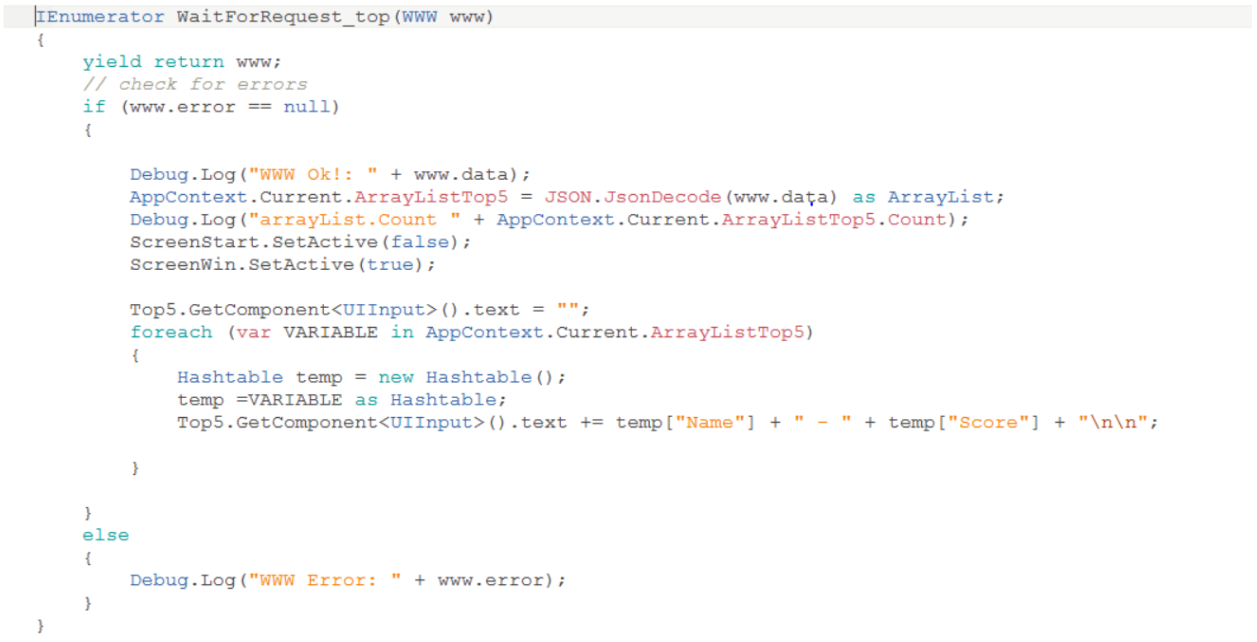
On another aspect, in order to have the ranking result visible to the players, some of the following methods would be applied:

- As the "Highscores" button is clicked, a function would be called to the above web service to obtain the current top 5. Also, in this method, in order to have the processes run properly, a method StartCoroutine would be used to tell the system to wait until this sub-process to be done before doing the next steps.



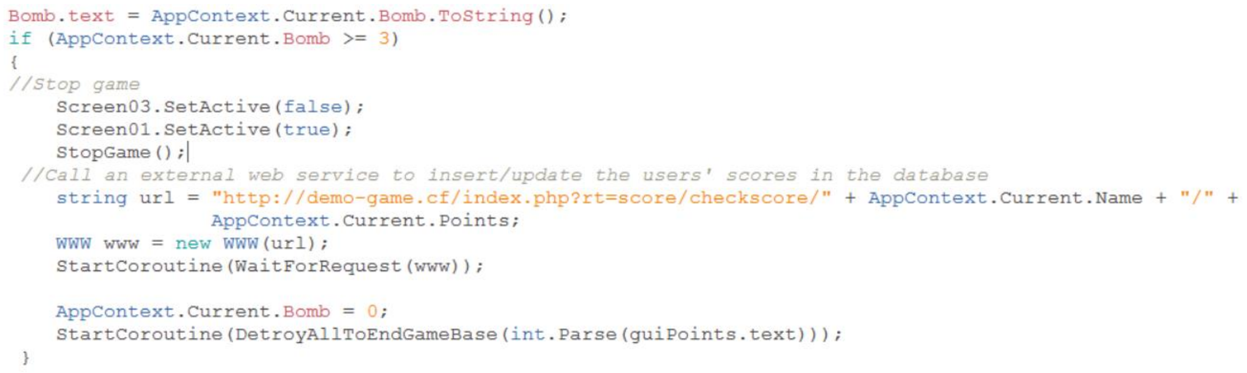
```
void Start ()
{
     AppContext.Current.IsbaseGame = true;
     EventDelegate.Add(ButtonStart.onClick,()=>{ StartGame(3);});
     EventDelegate.Add(ButtonHighScore.onClick, () =>
          {
               string url2 = "http://demo-game.cf/index.php?rt=score/gettop";
               WWW www2 = new WWW(url2);
               StartCoroutine(WaitForRequest_top(www2));


          });
     EventDelegate.Add(ButtonReplayWin.onClick, StartScreen);
```

Another issue is that when the output of the web service tends to be in the form of JSON. Therefore, at this moment is a solution for handling the JSON data.

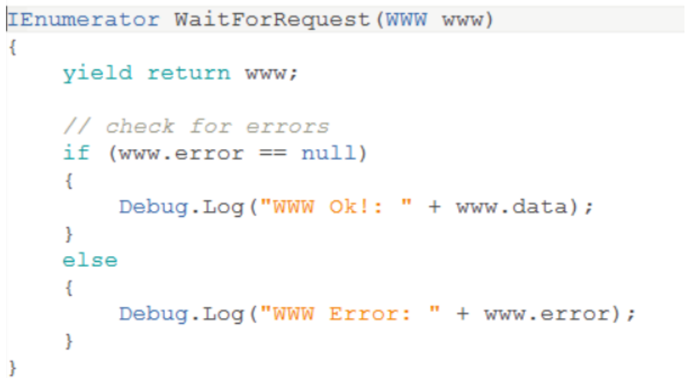


```
IEnumerator WaitForRequest_top(WWW www)
{
    yield return www;
    // check for errors
    if (www.error == null)
    {

        Debug.Log("WWW Ok!: " + www.data);
        AppContext.Current.ArrayListTop5 = JSON.JsonDecode(www.data) as ArrayList;
        Debug.Log("arrayList.Count " + AppContext.Current.ArrayListTop5.Count);
        ScreenStart.SetActive(false);
        ScreenWin.SetActive(true);

        Top5.GetComponent<UIInput>().text = "";
        foreach (var VARIABLE in AppContext.Current.ArrayListTop5)
        {
            Hashtable temp = new Hashtable();
            temp =VARIABLE as Hashtable;
            Top5.GetComponent<UIInput>().text += temp["Name"] + " - " + temp["Score"] + "\n\n";

        }

    }
    else
    {
        Debug.Log("WWW Error: " + www.error);
    }
}
```

As mentioned in section 3, after the third bomb was hit, the game would be over. Therefore, it is suitable to upload the username and its score to the web at such a moment.



```
Bomb.text = AppContext.Current.Bomb.ToString();
if (AppContext.Current.Bomb >= 3)
{
//Stop game
    Screen03.SetActive(false);
    Screen01.SetActive(true);
    StopGame();
 //Call an external web service to insert/update the users' scores in the database
    string url = "http://demo-game.cf/index.php?rt=score/checkscore/" + AppContext.Current.Name + "/" +
                AppContext.Current.Points;
    WWW www = new WWW(url);
    StartCoroutine(WaitForRequest(www));

    AppContext.Current.Bomb = 0;
    StartCoroutine(DetroyAllToEndGameBase(int.Parse(guiPoints.text)));
 }
```



```
IEnumerator WaitForRequest(WWW www)
{
    yield return www;

    // check for errors
    if (www.error == null)
    {
        Debug.Log("WWW Ok!: " + www.data);
    }
    else
    {
        Debug.Log("WWW Error: " + www.error);
    }
}
```

6. Results and Evaluation

6.1. Results

By applying the researched knowledge about unity into the actual work, it was successful to develop a small 3D game bases on the Unity Framework. This game can be run on MacOS®, Windows® and mobile devices. However, due to the lack of facility, it was not possible to test on Mac and iOS devices. Hence, only the Windows version and the Android version of the application were tested.

This game also meets most of the requirements that were proposed in the sections above.

6.2. Discussion & Evaluation

After the testing process, there are a few optimisable functions that should be considered to implement in the next version of the application:

Function Visibility

It was discussed that sometimes the game is a lack of navigation information, for instance, the user can only tell the level in which they are currently playing by distinguishing the differences between the background. It should be more visible to the users. Therefore, in the next implementation, it should be consider putting the level indicator in the GUI interface.

Sign in & sign out

It has come to the point that one or more user could have to share the same device while using this application, however, it has also been found that the application has no option for managing the users' id such as login and logout. Hence, the user can only log out be closing the application, and this might lead to some inconvenience. As a result, it is highly recommended to implement an access management system in the next version of the game.

Optimise database

Additionally, due to the fact that this game was created as a prototype demonstration for the practicality of using Unity to implement a 3D game, the database in which the users' information would be stored was not designed carefully. The database has only two entities which are Name and Score. This could be problematic when deploying the programme on a mass scale; this is because there are many people whose names are identical and name should not be used as a primary key. Although it was a solution in the prototype that if the name is identical, it would be overwritten but it is unprofessional to do so in the release version of the application.

7. Conclusion and future works

7.1. Conclusion

In conclusion, it has been demonstrated that Unity Framework could be used for implementing an interactive 3D game in a small to medium scale project with ease of understanding, throughout workflow and excellent maintainability along with flexibility.

7.2. Future works

As mentioned in the section above, because this implementation was just a demonstration for the practicality of the method, therefore, there are a few more thing that could be developed in the future in addition to the ones that was discussed in the discussion part.

It would be a right way to optimise the algorithms so that the game could run in a more efficient way.

Another thing is that the server-client communication should be consider researching and implementing so that it would increase the interactivity and communicative of the game.

Additionally, not all of the machine has the high-end configuration, hence, it might be worth trying to optimise the 3D effects so that it would be possible for the low-end machine to run the game smoothly.

References

Adachi, P.J.C., Willoughby, T., 2011. The effect of violent video games on aggression: Is it more than just the violence? Aggress. Violent Behav. 16, 55–62. doi:10.1016/j.avb.2010.12.002

Candy Crush Saga, 2016. Wikipedia.

Caoili, E., n.d. Facebook's most popular games continue to lose users [WWW Document]. URL http://www.gamasutra.com/view/news/169408/Facebooks_most_popular_games_continue_to_lose_users.php (accessed 8.11.16).

Chen, C., Leung, L., 2016. Are you addicted to Candy Crush Saga? An exploratory study linking psychological factors to mobile social game addiction. Telemat. Inform. 33, 1155–1166. doi:10.1016/j.tele.2015.11.005

Cummings, H.M., Vandewater, E.A., 2007. Relation of Adolescent Video Game Play to Time Spent in Other Activities. Arch. Pediatr. Adolesc. Med. 161, 684–689. doi:10.1001/archpedi.161.7.684

Eberly, D.H., 2006. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. CRC Press.

Erminesoft, 2016. How Much Does it Cost to Develop a Game With Unity 3D? Erminesoft.

Fruit Ninja, 2016. Wikipedia.

Game engine, 2016. Wikipedia.

Gaudiosi, J., 2015. In search of growth, video game companies hungrily eye Southeast Asia. Fortune.

Healy, S., 2016. Free to play, Pay to win. TechFlow.

Heaven, D., 2015. Return of the arcade. New Sci. 225, 20–21. doi:10.1016/S0262-4079(15)60069-0

Kent, S.L., 2001. The Ultimate History of Video Games: From Pong to Pokémon and Beyond : the Story Behind the Craze that Touched Our Lives and Changed the World. Prima Pub.

Kickmeier-Rust, M.D., 2012. An Alien's Guide to Multi-Adaptive Educational Computer Games. Informing Science.

Newzoo, 2016. Global Esports Market Report: Revenues to Jump to $463M in 2016 as US Leads the Way | Newzoo [WWW Document]. URL https://newzoo.com/insights/articles/global-esports-market-report-revenues-to-jump-to-463-million-in-2016-as-us-leads-the-way/ (accessed 8.10.16).

Newzoo, n.d. Top Countries by Game Revenues. Newzoo.

Rickard, C., n.d. Choosing the right mobile app for your project: Native vs cross-platform vs hybrid [WWW Document]. Inoutput Softw. Dev. Web Mob. Appl. Dev. Melb. URL http://inoutput.io/articles/development/choosing-the-right-mobile-app-for-your-project-native-vs-cross-platform-vs-hybrid (accessed 8.11.16).

Shead, S., n.d. The Angry Birds Movie just took $150 million at the box office and gave Rovio a vital lifeline in the process [WWW Document]. URL http://uk.businessinsider.com/the-angry-birds-movie-has-made-150-million-worldwide-2016-5 (accessed 8.4.16).

Stuart, K., 2014. The 30 greatest video games that time forgot. The Guardian.

Unreal Engine, 2016. . Wikipedia.

Ward, J., 2008. What is a Game Engine? [WWW Document]. Game Career Guide. URL http://www.gamecareerguide.com/features/529/what_is_a_game_.php

Wolf, M.J.P., 2008. The Video Game Explosion: A History from PONG to Playstation and Beyond. ABC-CLIO.

Appendix

Link to the Unity Package: https://1drv.ms/u/s!AmMi6Dwx9-F9-Cxg-4KGYuORx7KV

Link to the Android (6.0 or later) apk file: https://1drv.ms/u/s!AmMi6Dwx9-F9-C1a7mE9UI9iZR7M