**Fish on the Run – Gaming App**

Final Report for CS39440 Major Project

Author: Dimitar Plamenov Velikov (dpv@aber.ac.uk)

Supervisor: Otara Akanyeti (ota1@aber.ac.uk)

27 March 2018

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree

In Computer Science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name: Dimitar Velikov

Date: 27/03/2018

**Consent to share this work**

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Dimitar Velikov

Date: 27/03/2018

**Contents**

[1 Introduction 5](#_Toc509929291)

[1.1 Motivation 6](#_Toc509929292)

[1.2 Background 6](#_Toc509929293)

[1.3 Game Description 6](#_Toc509929294)

[1.4 Design Requirements 7](#_Toc509929295)

[1.5 Teaching Objectives 7](#_Toc509929296)

[1.5.1 Preference of prey 7](#_Toc509929297)

[1.5.2 Energetic cost of swimming 8](#_Toc509929298)

[1.5.2 Size matters 8](#_Toc509929299)

[2 Preparation and Process 8](#_Toc509929300)

[2.1 Preparation 8](#_Toc509929301)

[2.1.1 Behind the scenes 8](#_Toc509929302)

[2.1.2 Game Engine editor selection 9](#_Toc509929303)

[2.1.2.1 Low-Level APIs 9](#_Toc509929304)

[2.1.2.2 High-Level APIs 9](#_Toc509929305)

[2.1.3 Performance 11](#_Toc509929306)

[2.1.3.1 Rendering 11](#_Toc509929307)

[2.1.3.2 Frames Per Second 11](#_Toc509929308)

[2.1.3.3 Movies vs Games 11](#_Toc509929309)

[2.1.3.5 CPU and GPU bound 11](#_Toc509929310)

[2.1.3.6 Achieving high performance with Unity 12](#_Toc509929311)

[https://unitytutorials.ca/misc/performance/writing-high-performance-c-scripts 12](#_Toc509929312)

[2.1.3.7 Intermediate Language to C++ (IL2CPP)) 12](#_Toc509929313)

[2.1.3.8 Goal 12](#_Toc509929314)

[2.2 Setting up the Environment 13](#_Toc509929315)

[2.2.1 3D Game Engine Creator - Unity 13](#_Toc509929316)

[2.2.2 C# IDE - Visual Studio 14](#_Toc509929317)

[2.2.3 Autodesk Maya 14](#_Toc509929318)

[2.2.4 Version Control - GitHub 14](#_Toc509929319)

[2.3 Process and Project Adaptation 14](#_Toc509929320)

[2.2.1 About Scrum 14](#_Toc509929321)

[2.2.2 Scrum Adaptation for the Project 15](#_Toc509929322)

[3 Design and Implementation 17](#_Toc509929323)

[3.1 Sprint 1 17](#_Toc509929324)

[3.1.1 Sprint Planning 17](#_Toc509929325)

[3.1.2 Design 17](#_Toc509929326)

[3.1.3 Implementation 17](#_Toc509929327)

[3.1.4 Review 17](#_Toc509929328)

[3.2 Sprint 2 17](#_Toc509929329)

[3.1.1 Sprint Planning 17](#_Toc509929330)

[3.1.2 Design 17](#_Toc509929331)

[3.1.3 Implementation 17](#_Toc509929332)

[3.1.4 Review 17](#_Toc509929333)

[3.3 Sprint 3 18](#_Toc509929334)

[3.1.1 Sprint Planning 18](#_Toc509929335)

[3.1.2 Design 18](#_Toc509929336)

[3.1.3 Implementation 18](#_Toc509929337)

[3.1.4 Review 18](#_Toc509929338)

[3.4 Sprint 4 18](#_Toc509929339)

[3.1.1 Sprint Planning 18](#_Toc509929340)

[3.1.2 Design 18](#_Toc509929341)

[3.1.3 Implementation 18](#_Toc509929342)

[3.1.4 Review 18](#_Toc509929343)

[3.5 Sprint 5 18](#_Toc509929344)

[3.1.1 Sprint Planning 18](#_Toc509929345)

[3.1.2 Design 18](#_Toc509929346)

[3.1.3 Implementation 18](#_Toc509929347)

[3.1.4 Review 18](#_Toc509929348)

[3.6 Sprint 6 19](#_Toc509929349)

[3.1.1 Sprint Planning 19](#_Toc509929350)

[3.1.2 Design 19](#_Toc509929351)

[3.1.3 Implementation 19](#_Toc509929352)

[3.1.4 Review 19](#_Toc509929353)

[3.7 Sprint 7 19](#_Toc509929354)

[3.1.1 Sprint Planning 19](#_Toc509929355)

[3.1.2 Design 19](#_Toc509929356)

[3.1.3 Implementation 19](#_Toc509929357)

[3.1.4 Review 19](#_Toc509929358)

[3.2 Design 19](#_Toc509929359)

[3.3 Implementation 19](#_Toc509929360)

[3.3.1 Unity’s all-in-one editor basic components 20](#_Toc509929361)

[3.3.2 Unity scripts execution order 21](#_Toc509929362)

[3.3.3 Game Implementation 23](#_Toc509929363)

[3.3.3.1 Main Character 23](#_Toc509929364)

[3.3.3.2 Procedurally generated world 24](#_Toc509929365)

[3.3.3.3 Collision Detection 26](#_Toc509929366)

[3.3.3.4 User Interface Elements 27](#_Toc509929367)

[3.3.3.5 Storing the Player data locally 27](#_Toc509929368)

[3.3.3.6 Mobile Input 28](#_Toc509929369)

[3.3.3.7 Change Scene 29](#_Toc509929370)

[3.3.3.8 Main menu scene 30](#_Toc509929371)

[3.3.3.9 Post-Processing Effects 31](#_Toc509929372)

[4 Testing 31](#_Toc509929373)

[4.1 Issues found while testing on a mobile device 31](#_Toc509929374)

[4.1.1 iOS Permissions 31](#_Toc509929375)

[4.1.2 Swipe Controls 32](#_Toc509929376)

[4.1.3 Post-Processing Stack 34](#_Toc509929377)

[4.1.4 The problem of tunnelling 35](#_Toc509929378)

[4.2 Security Testing 36](#_Toc509929379)

[5 Evaluation 37](#_Toc509929380)

[5.1 Survey 37](#_Toc509929381)

[5.2 Any changes based on the survey 37](#_Toc509929382)

[5.3 Critical Evaluation 37](#_Toc509929383)

[5.4 Summary 37](#_Toc509929384)

[6 Future work 37](#_Toc509929385)

[Accessibility – Colour-blind mode 37](#_Toc509929386)

[7 Bibliography 37](#_Toc509929387)

# 1 Introduction

This report will discuss the whole creation process of the “Fish on the Run” – Gaming App.

1.1 Motivation

Game development inspires me because it gives me the opportunity to express myself and to be innovative while creating virtual worlds. As most teenagers of today, I’ve been playing computer games since my early childhood. My desire to create my own game has greatly increased after taking the first steps in the game development after taking/with Aberystwyth University’s “Computer Graphics and Games” module. In this module, I obtained basic theoretical and practical knowledge of how real games are made and how they operate. This is why I have decided to extend my knowledge in this subject matter and to develop and document a non-commercial game with commercial-use quality as my dissertation.

1.2 Background

The idea for the plot of the game is a collaboration of the project supervisor’s fishing game project suggestion and my wish to develop a game similar to the other popular endless world games such as Temple Run and Subway Surfers.

Looking through the App Store (iOS) and Google Play Store (Android) I have found a variety of fishing games but none of them was similar to “Fish on the Run”. What makes this game unique from the others is the way the main character is swimming forward in an endless-world.

1.3 Game Description

The game brings satisfaction to its players by implementing the idea that the education and fun can be symbiotic. “Fish on the Run” has a competitive nature since the players are competing against each other while trying to achieve the highest possible score but is also an educational game since the players learn about marine life and the environment while playing.

The game is based on a fish that swims upstream in a river. While swimming the fish needs to avoid floating objects such as other fish, poisonous ivy, and other obstacles. The fish will lose energy every second it swims. To gain energy the fish can prey on smaller fish or zooplanktons or rest behind large sedimentary rocks. For each fish size, there will be an optimum rock size. There will be also fishing rods flowing from the river’s surface. The fish has to collect coins and runes. The coins will contribute to the player’s high score and the runes will give him some advantage i.e. invulnerability, higher speed etc.

1.4 Design Requirements

The final product will be a 3D iOS and Android cross-platform mobile game. It will be developed using Unity creation engine for gaming software and C# programming language. The game is of commercial-use quality and is optimized for both platforms. The game runs at 60 frames per second with no crashing, lag or delay. The game contains multiple scenes including main menu, settings, and a list of all the fish with a short description for each one and a game scene. The game is fully operated by and responsive to tactile user input. The track is randomly generated using predefined paths. End of game state etc.

## 1.5 Teaching Objectives

“Fish on the Run” attempts to reflect the habits of underwater animals. The idea is for the player to gain:

* Tacit knowledge – Knowledge that is gained through experience (i.e. while playing the game). More information in sections 1.5.1, 1.5.2, 1.5.3.
* Explicit knowledge – Knowledge that is gained through reading information.

In order to get feedback about how the game reflects the player’s comprehension, the tacit knowledge is evaluated via survey in Section 5.

https://www.tlu.ee/~sirvir/Information%20and%20Knowledge%20Management/Key\_Concepts\_of\_IKM/tacit\_and\_explicit\_knowledge.html

### 1.5.1 Preference of prey

Like people some fish eat meat, others don’t. Unlike people, this is not a lifestyle choice for fish. There are three types:

* Omnivores – Eat both vegetables and meat.
* Carnivores – Meat eaters, require live foods.
* Herbivores – Eat plants, algae and fruits.

https://www.petmd.com/fish/nutrition/evr\_fi\_fish\_nutrition, https://www.thespruce.com/omnivore-herbivore-carnivore-1380916

The main character in the game is a shark. All sharks are carnivores. They eat fish and sea mammals. They even prey turtles and other sharks (they are cannibals).

http://www.kidzone.ws/sharks/facts4.htm

### 1.5.2 Energetic cost of swimming

All the species have energy and most of them swim at some point in their lifecycle. Swimming is a defining feature for water mammals as well as:

* Finding food.
* Escaping predators.
* Defending territory.
* Reproduction.

Types of swimming and muscle:

* Aerobic red muscles – Usually represent nearly 10% of the muscle mass. Those muscles are used for slow or steady swimming. They don’t cost much energy.
* Anaerobic white muscles – Usually over 50% of the body mass. They are used for high-speed swimming and bursts. This activity takes much more energy.

In those terms, while following the water flow the game character’s energy is slowly decreasing, because it is making aerobic movement. Cases, when the character is making any move to the left or to the right are considered to be brief burst movements that take more energy.

https://www.researchgate.net/publication/253253377\_The\_energetics\_of\_Fish\_Swimming

### 1.5.2 Size matters

All sharks are carnivores, but the size of their prey matters. For example, when they are preying on other sharks, they have to be smaller.

# 2 Preparation and Process

This section discusses the preparation for the project, including background reading and the analysis based on the problem and the preparation.

## 2.1 Preparation

The preparation phase of the project length is two weeks – from 29/01/2018 to 09/02/2018. It consists of studying about games in general, game development, tools, game loop. It also includes setting up the environment and implementing simple tasks in order to gain knowledge of how to operate with the new software.

### 2.1.1 Behind the scenes

A 3D game consists of different components:

* Game Objects – such as Models, Lightings, HUDs
* Game Engine – This is what makes the game interactive. Part of its functionality is to add, remove and move objects around the scene, detect collisions and to respond to them adequately. Depending on the framework used, all of it or most of it is written in script files.
* Scene – This is where all the Game Objects are located. Every object has a position on the scene that is determined by the value of its X, Y and Z axis.
* Camera – The camera is also a Game Object and has its position on the scene. It is used to display a view of the virtual world to the player.
* Renderer – The renderer is drawing the game objects that are located in the camera parameter (i.e. the objects that are seen by the current position and rotation of the camera).

### 2.1.2 Game Engine editor selection

This section compares several different Game Engine editors in order to select the best one to use during the project development and testing phase. Nowadays games are created using Game Development Platforms, Frameworks and APIs. These platforms are made to facilitate the game creation, as well as allowing smaller teams with a less financial capacity to be able to compete against larger software companies. These methods make it easier to make games interactive, as well as simplifies the implementation of Game Logic, Graphics, Audio, Networking, Artificial Intelligence and Algorithms.

#### 2.1.2.1 Low-Level APIs

Lower-level APIs such as OpenGL and Metal are designed to be more efficient than their higher-level counterparts (e.g. Apple’s SceneKit and Unity, please see section on High-Level APIs for more detail). They send instructions directly to the graphics processing unit (GPU). Lower-level APIs are not appropriate for beginners because they are very difficult. They demand a very strong theoretical knowledge which includes a fluent understanding how the GPU works.

<https://developer.apple.com/documentation/scenekit>

#### 2.1.2.2 High-Level APIs

Contrary to the Low-Level APIs, these types of APIs do not require such a strong theoretical knowledge and they do not have direct access to the hardware. This makes the game development for beginners easier, but less efficient to develop games with it.

https://developer.apple.com/scenekit/

##### Apple’s SceneKit

SceneKit is a high-level 3D graphics framework that can be used to create 3D games [2]. As a result of a short study it has been found out that the framework has several disadvantages:

* The available tutorials and sources that can be used to gain knowledge about the framework are limited.
* Another disadvantage is that the framework does not have graphical scene editor and the only way to track the visual progress on the project is to run an Apple Simulator via Xcode and execute the application [3,4].
* Requires Xcode software and an Apple device to execute it (i.e. MacBook).
* The application can be deployed only on Apple devices.

##### Unity

Unity is one of the world’s leading creation engines for gaming. It does come with all of the tools necessary to build a 3D game. Unity comes with huge advantages over previously listed platforms:

* All-in-one editor – displays the scene and all of its objects. Every object can be modified directly from the editor. The editor gives the opportunity to perform the basic operations such as movement, scaling and rotation, as well as modifying more specific game object parameters and attaching components such as collision detectors and scripts. The editor also has options for camera, rendering, world lighting, fog etc.
* Scripting in Unity – Is similar to SceneKit, the game object’s behaviour has to be described in code and stored in scripts. Those scripts can be written in either C# or Unity Script (JavaScript adapted for Unity).

[https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/]

* Portability – Every project can be rebuilt and deployed to more than 25 platforms. This means that the scope of the project will not only cover iOS devices, but it could also be built for VR, PC, Mac, etc. It has to be taken into account that the game character has to be controlled via the user’s input and the typical user input for the mobile devices are tactile. On the other hand, PC games input typically comes from the mouse and the keyboard. This means that simply switching the platform is not enough and there are some slight modifications that have to be adjusted for most of the platforms. https://unity3d.com/unity/features/multiplatform
* Consider writing for profiler and other debugging tools

Might write about Autodesk Maya here. Depends on whether I will have the time to create the object that I am planning to.

### 2.1.3 Performance

Performance tweaking is an essential part of the 3D Game Development and for the purpose of this project, it is examined in great detail.

#### 2.1.3.1 Rendering

All 3D Models are made up of triangles and squares. The more shapes you have the more detailed is the model. Respectively the more shapes the higher the rendering cost is.

The 3D world in games is called ‘a Scene’ and contains different models, lightings, shadows, shadings etc.

The rendering process can be defined as drawing these objects on the screen.

**Write more about rendering** https://unity3d.com/learn/tutorials/temas/performance-optimization/optimizing-graphics-rendering-unity-games

#### 2.1.3.2 Frames Per Second

In order for a game to be enjoyable, it should run smoothly with no lag or delay. This means that the drawings are repeated multiple times per second. Each drawing repetition is called a frame and a reasonable measure of performance is how many frames can be rendered per second.

An important thing to consider while measuring the performance is that different devices have different hardware specifications and some devices may perform better than others. This is why the PC games have minimum hardware requirements. The performance that could be achieved with mobile games is worse since the hardware of mobile devices is not as powerful as those in personal computers.

#### 2.1.3.3 Movies vs Games

Rendering for video games is a complicated process and very different from movies, which the general gaming public is not aware of. Movies are pre-rendered, some of them have much more computationally heavy graphics and require very powerful hardware systems in order to complete that rendering in a reasonable amount of time. Those hardware systems are called rendering farms.

On the other hand, video games use the hardware based on the device to render their graphics. Therefore, the graphics are calculated by the time they are executed, so they have to be simpler and more efficient, which makes their capabilities limited.

#### 2.1.3.5 CPU and GPU bound

Central Processing Unit

#### 2.1.3.6 Intermediate Language to C++ (IL2CPP))

<https://blogs.unity3d.com/2015/05/06/an-introduction-to-ilcpp-internals/>

C++ is an object-oriented language which outperforms most or all of the OOP languages.

Unity is usually described as a C# game engine, but it is actually written in C++ and It is written in that language because it is much faster than its alternatives.

C# is easier and is much more difficult to do something wrong because it is a managed language. A managed language is a language that has a virtual machine (VM also called Garbage Collector - GC) which deallocates the allocated variables in memory and makes the application memory safe. This is why most of the Unity Scripts are written in C#.

When a game is written in C#, the code gets compiled in the game into IL (intermediate language). Since Unity supports 28 platforms and every game has to be able to support each of them, the game is converted from IL to C++. It is then sent to the platform’s compiler and compiled to the platform’s native language. As an example, for iOS applications such as this project, Xcode will recompile the game from C++ to ARM assembly language.

#### 2.1.3.7 Goal

Apparently, the more FPS (Frames per Second) the smoother the motion will appear. There is a myth in gaming industry according to which 60 frames per second are the optimal frame rate and the human eye is not capable of seeing more than that. There are different reaction tests that prove the human eye can react to visual signs in less than a millisecond, which corresponds to 1000 FPS.

On the other hand, even if the hardware is capable of rendering 200 FPS, the frames the player is going to be able to see are upper-bounded by the monitor/display refreshing rate measured with Hz (60Hz = 60 FPS). Nowadays most of the PC monitors have a refreshing rate of 60Hz. There are high-end gaming monitors which are capable of producing 144Hz.

Despite that, the refreshing rate of flagship high-end mobile phones such as iPhone X and Samsung Galaxy S8 is 60Hz. There are probably very rare cases where mobile devices have a refresh rate higher than that. In the best case, in the final release version of the ‘Fish on the Run’ game has to run in 60FPS. [6]

## 2.2 Setting up the Environment

### 2.2.1 3D Game Engine Creator - Unity

Unity offers three products:

https://store.unity.com/?\_ga=2.60661989.970581740.1519081095-102683856.1512945710

* **Unity Personal** – A completely free software, coming with basic functionality and is considered to be appropriate for beginners and students. One disadvantage for this version is that at the start of every application, there is a “Created by Unity” start screen image, which for a game with a purpose wider than educational is not very professional.

**Insert a screenshot**

* **Unity Plus** – Can be acquired for a paid subscription with a minimal length of one year (35 USD per month). Appropriate for serious developers. Comes with more features such as multiplayer, real-time performance reporting and others.
* **Unity Pro** – Even greater flexibility and feature extension in return for 125$ per month. Appropriate for studios and professional developers.

It has been considered that Unity Personal software and the feature pack that it comes with completely suits the needs of this project.

Setting up the Unity environment consisted of:

* Create a Unity account.
* Download the software.
* Install the software.
* Login to the account.
* Create a new project.

### 2.2.2 C# IDE - Visual Studio

Visual Studio Community Edition is the default script editor for Unity since version 2018.1. It is a free IDE and is used for creating desktop, web and mobile applications.

### 2.2.3 Autodesk Maya

Autodesk Maya is a professional software for creating sophisticated 3D graphics and animations and has been acquired for free with a three-year student license.

https://www.autodesk.co.uk/products/maya/overview

Setting up the Autodesk Maya environment consisted of:

* Registering on the Autodesk website.
* Verifying the current status as a student.
* Acquiring free license.
* Downloading and installing the software.
* Registering within the software with a unique key generated from the website.

### 2.2.4 Version Control - GitHub

Setting up a new repository and allocating its path.

## 2.3 Process and Project Adaptation

http://www.scrumguides.org/scrum-guide.html

It has been considered that this project will be managed by the Scrum process. Scrum is a framework for developing and delivering products.

### 2.2.1 About Scrum

##### The Scrum team has three roles:

* Product Owner – Defining the Product Backlog features and ordering them to best achieve goals and items.
* Development Team – Professionals whose task is to deliver working features at the end of each sprint.
* Scrum Master – A person that helps the team understand the Scrum theory, rules, values and guide them.

##### Scrum process has several events:

* Sprint – A time box of 1 – 4 weeks. The goal is to accomplish a task or a set of tasks that are considered during the Sprint Planning.
* Sprint Planning – The entire product team is considering the amount of work to be assigned for the next sprint.
* Daily Scrum – A daily 15-minute meeting for the Development Team. Optimizes the team work and the team performance. Includes discussion about the current sprint progress and issues.
* Sprint Review –

##### Three Artifacts:

* Product Backlog – A list of all the features that are known to be needed in the product. Typically, the initial backlog is incomplete.
* Sprint Backlog – The selected features from the Product Backlog that are going to be implemented during the sprint.
* Increment – The sum of all the Product Backlog completed features marked as “Done”. Used to track the progress of the project.

### 2.2.2 Scrum Adaptation for the Project

##### Team:

Since the project is a collaboration of mine and the project supervisor’s ideas, we are both product owners. I am also a scrum master and a developer (a single person development team).

##### Events:

The sprint length is considered to be one week. Every sprint will be documented in **Section 3** and **Section 4** (testing section). It consists of:

* **Sprint Planning** – Sprint Planning event is held once a week. Considers the features to be implemented during the following sprint. Also includes the creation of an up-front design diagram.
* **Implementation -** ds
* **Testing** - Unity 3D Game Engine Creator does not provide the trivial way of writing Object-Oriented Code. This is one of the reasons why the Unit Testing has been considered irrelevant for the project. The feature testing for this project consists of observation of the application behaviour during runtime on both Unity Editor and a Mobile Device as well as having a console output of the different features and events.
* **Sprint Review** – A meeting between me and the project supervisor to discuss the amount work done through the sprint and receive a feedback.

Both Sprint Planning and Sprint Review are conducted at the same time– every Friday in one-hour time slot. The meeting starts with the Sprint Review and ends with the planning of the next week Sprint.

There is no Daily Scrum. There is a weekly Scrum instead – a two hour meeting every Tuesday with a group of 5-10 students where each of the students is talking about its project and receiving some useful feedback, ideas and suggestions.

##### Artifacts:

The user stories are written using the free online tool <https://trello.com/>. The tool also gives the opportunity to give each story a particular colour or a set of colours. This is how the difficulty of each story is indicated. The web tool also provides different groups that can store stories. For example, initially all the stories are under group “To Do”. During a sprint the selected set of stories are moved to group “Doing”. At the end of every sprint, the completed stories from “Doing” are moved to “Done”.

# 3 Design and Implementation

This is the main phase of the project. It starts at 12/02/2018 and ends at 23/04/2018. This time includes a 3-week Easter break from 26/03/2018 to 16/04/2018. This means that this phase took 7 weeks and for each of those weeks was scheduled a different sprint.

## 3.1 Sprint 1

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.2 Sprint 2

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.3 Sprint 3

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.4 Sprint 4

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.5 Sprint 5

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.6 Sprint 6

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.7 Sprint 7

### 3.1.1 Sprint Planning

### 3.1.2 Design

### 3.1.3 Implementation

### 3.1.4 Review

## 3.2 Design

It is difficult to make a design for a game build in Unity because the game consists of multiple components, typically located on the game scene, interacting with each other according to their description written in C# script files. The components can be looked as features. In those terms, a Unity project is very flexible and extensible. In any phase of the project, any feature (component) can be modified, extended or removed without any sequences or the necessity of refactoring.

## 3.3 Implementation

This section will start with a short introduction of the Unity all-in-one editor, its script execution order and will cover the actual game implementation.

### 3.3.1 Unity’s all-in-one editor basic components

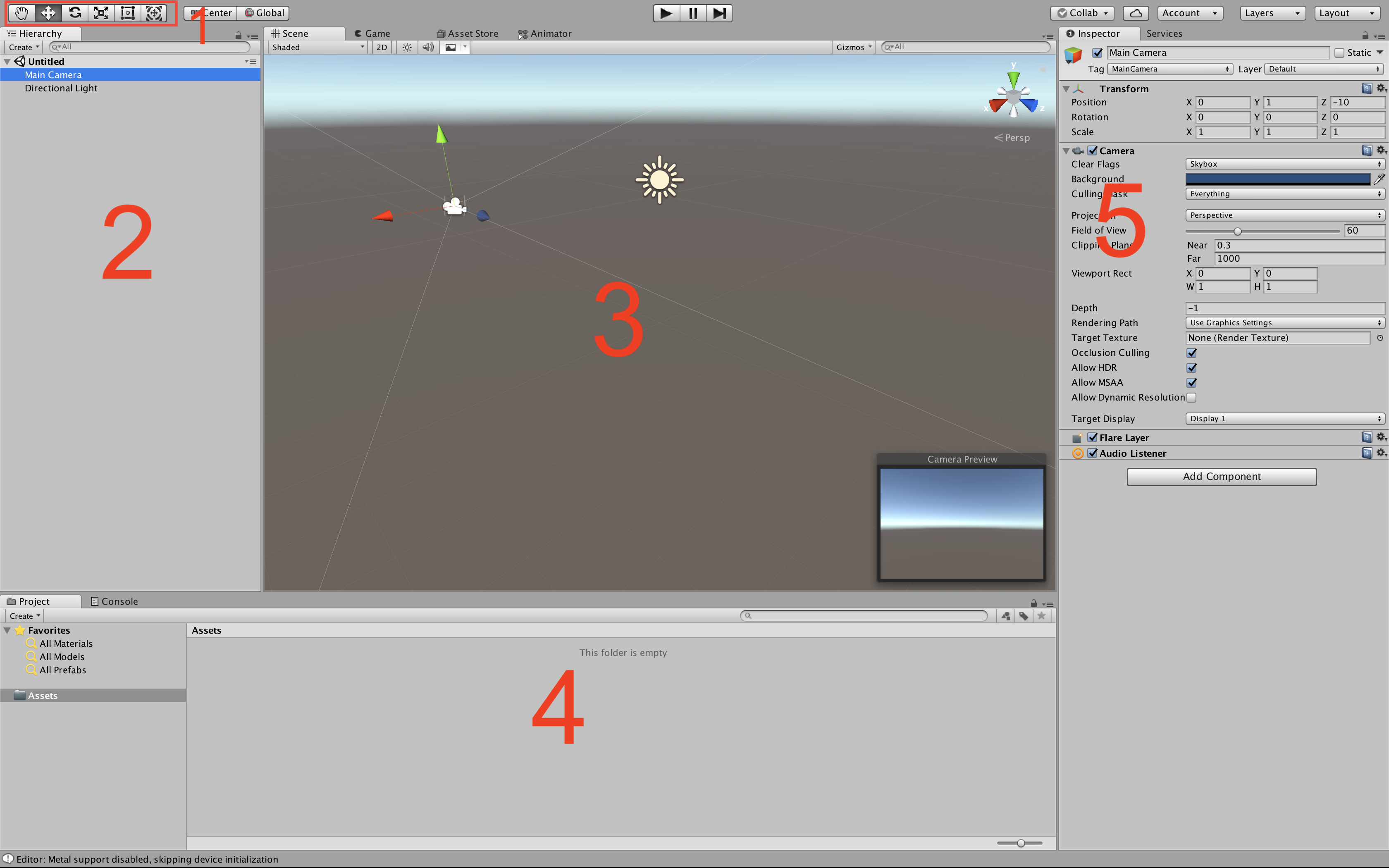


Image 3.3.1

***Image 3.3.1*** is an example of an empty Unity project.

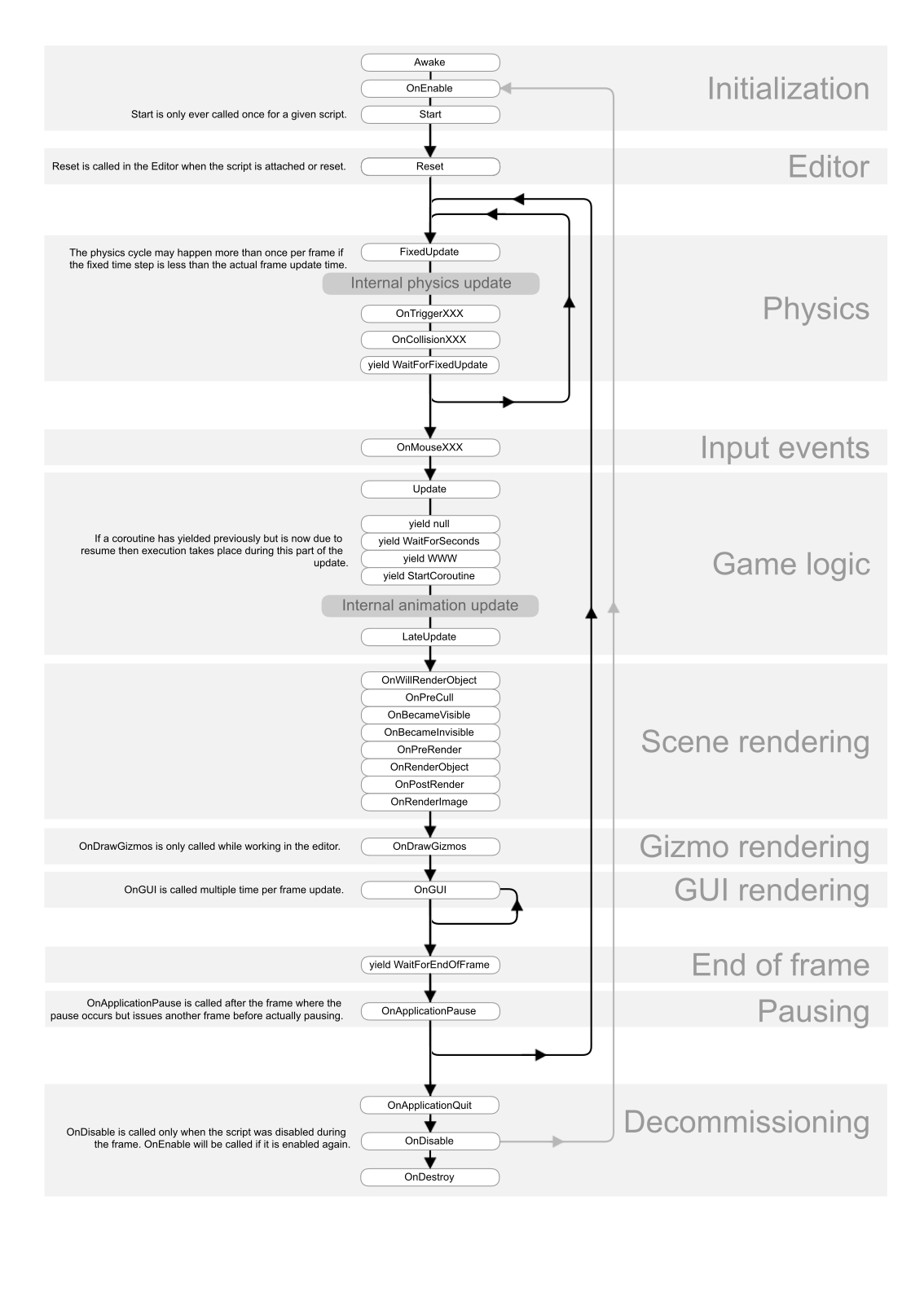
* 1 – Tools:
  + – Move the current view around the scene and rotate.
  + – Move Game Objects around the scene.
  + – Rotate Game Objects around their X, Y or Z axis.
  + – Scale Game Objects.
* 2 – **List of all of the Game Objects**. After an object is selected it can be duplicated, removed, renamed, moved, scaled, translated. Object selection provides additional operations described in bullet point 5.
* 3 – **Scene editor**. A scene is populated with Game Objects. Displaying objects during the editing time is one of the Unity’s advantages.
* 4 – This section is containing two tabs.
  + **File explorer** - All of the project assets such as models, textures, prefabs, audio, video, text file, script file or any kind of Game Object is accessible in the file explorer.
  + **Console** – Displays all of the application logs such as errors, warnings, notifications including any debugging log invoked in script files using “*Debug.log(“”);”* function.
* 5 –**Object inspector**. It has easy access to every Game Object property. It can be used to activate or deactivate the selected object, scale, translate or rotate it, change its properties or add, remove, activate or deactivate its components.

### 3.3.2 Unity scripts execution order

Unity scripts are event functions that are executed in predetermined order as a script executes. https://docs.unity3d.com/Manual/ExecutionOrder.html

For the purpose of this project only three types of the listed functions will be used:

* **Start** – Initialization function. Called only once per script. Executed after Awake.
* **Update** – Called once per frame.

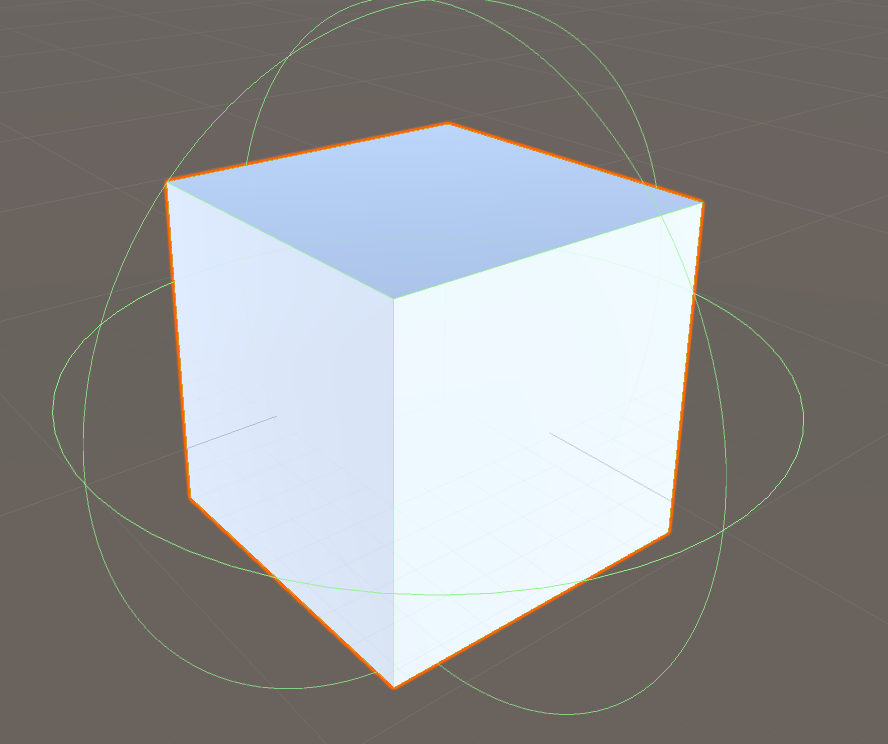


### 3.3.3 Game Implementation

This section will describe the implementation of the project with a discussion about each of the game components.

#### 3.3.3.1 Main Character

The implementation phase starts with the main character, its basic movement and actions.

***Image 3.3.3.1*** is a cube that will be used as a player object. The green lines around the object represent a “Character Controller” in the form of a sphere. This is a simple type of collision detector, such as a “Box Collider” with the difference that the used collision detector is able to change its position. The sphere collision detection is the most efficient kind of detectors. [add reference to CS3240 lectures here] This is because the detection check is performed every frame, and the calculation for a sphere collision is the simplest possible calculation. It only needs the centre of the Sphere and its radius.

**Image 3.3.2**

A new script is created and attached as a component of the character. The script’s purpose is to make the character move. A reference to the Character Controller (collision detector) is obtained in order to control the position of the character on the scene.

Different scene components can be reached using:

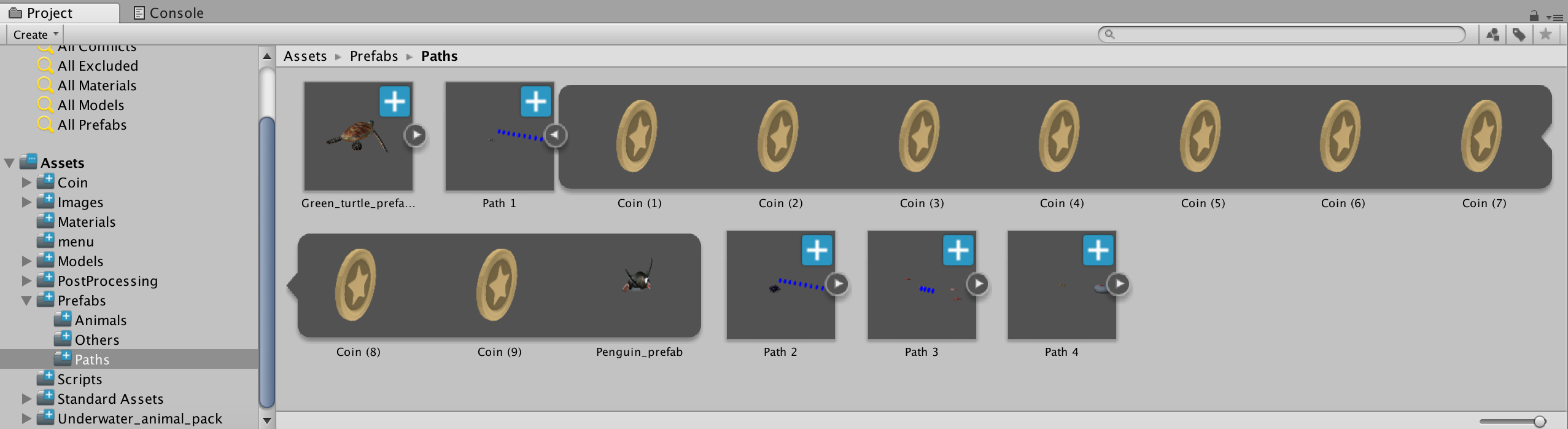
* “GetComponent<Name of Component>()” - If the desired component is attached to the same Game Object as the current script is attached to.
* “GameObject.Find(“Name of the Object”).GetComponent<ComponentName>()” – If the desired component is not attached to the Game Object, the current script is attached to.

#### 3.3.3.2 Procedurally generated world

##### This feature of the game is controlled by “ProcedurallyGeneratedWorld” script. It works on the principle of spawning predefined prefabs accessed by indexing the linked to them array with a random integer.

**What is a prefab?**

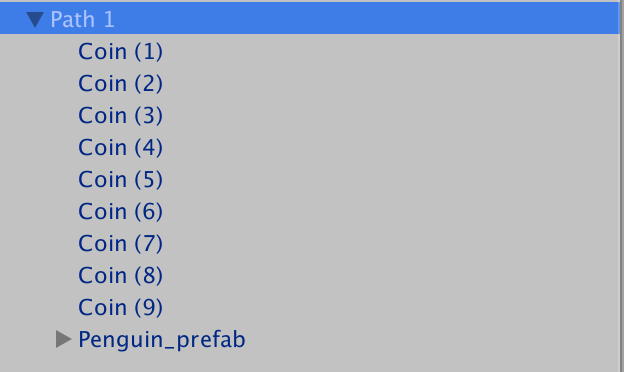
A prefab is usually a set of objects that are already tweaked with the developer’s preferences, properties and components. The prefab is stored as an asset file. It can be called and instantiated any time before the game execution and during runtime.



**Image 3.3.3.2**

**Image 3.3.3.2** shows a folder where the Path prefabs are stored. “Path 1” is expanded and shows each of its children which in the current case are 9 “Coin” objects and a “Penguin\_prefab”.

Usually, each prefab has a parent object and every other object is related to the parent object by position size, rotation etc. For example, if the parent object is scaled, all of its children will be scaled or if the parent object is disabled, all of its children will disappear. **Image 3.3.3.3.** gives an example of an expanded prefab named Path 1. The prefab is currently loaded into the scene and is a subject of modification. The indented objects are its children.



**Image 3.3.3.3**



**Image 3.3.3.4**

The “ProcedurallyGeneratedWorld” script contains:

Six class members:

* GameObject[] pathPrefabs; - This is an array and is used as a link to the prefabs. It is initialized in the Start function, but its size and type of elements are determined in the Unity editor – **Image 3.3.3.4**
* Transform player; - This is a reference to the position of the player object and is used in the Update() function in order to check whether is time to spawn a new prefab.
* float spawnZlocation = 0.0f; - Keeps a reference to the Z position of the last spawned prefab. Updated at each prefab spawning. Used to determine the position of the next prefab.
* const float pathLength = 4.85f; - This is the length of each prefab. Every prefab has equal length.
* const int pathCount = 5; - This is the number of the initially spawned prefabs. This number is maintained during the gameplay and is not changed.
* List<GameObject> renderedPaths; - Keeps a reference to all the paths that are currently to the scene. Used for easy deleting the paths that are currently behind the player.

Five functions:

* void Start() – Used for initialization.
* void Update() – Checks the current position of the player and calls “SpawnTile()” and “DeleteTile()” functions.
* void SpawnTile(int prefabIndex = -1) – Spawns a prefab in front of the player. Makes sure the world is “endless”.
* void DeleteTile() – Removes the first element of the renderedPaths list. This is a path that is no longer seen by the player (i.e. its current position is behind the player) and is removed for efficiency purpose.
* int RandomInt() – Returns a random integer. The random integer is used to determine which element of the pathPrefabs will be loaded to the scene.

#### 3.3.3.3 Collision Detection

Collision detection is occurring when two objects try to occupy the same space at the same time.

There are two main types of collision detection:

[https://blackboard.aber.ac.uk/bbcswebdav/pid-1110322-dt-content-rid-1759508\_1/courses/CS32420\_AB0\_2017-18/CS32420\_8\_collision1.pdf]

* Continuous (priori) – detects the collision before the actual collision. It consists of an algorithm which predicts the trajectories of the objects and is more computationally expensive, because of its complexity.
* Discrete (posteriori) – detects the collision after the objects have collided. Simpler and more efficient but may cause a “Tunnelling effect[[1]](#footnote-1).”

Types of colliders: [unity reference]

* Collider – There are different colliders with various shapes and properties, but the most commonly used collider for this assignment is the Box Collider. Box collider is a primitive collider and is used as an indication of the intersection of two objects. There are more complex colliders such as rigid body and mesh colliders which may detect more complex collisions, but they are typically used when more complex detections are required (i.e. shooting games – determine whether the player is shot in the head, body or at the hand etc.)
* Trigger - Unlike the typical colliders, when the trigger is activated the same detection is happening but there is no visual sequence i.e. the movement trajectory of both objects is not changed.

The most efficient colliders are primitive colliders, especially the sphere collider since on every frame the performed calculation only takes the centre of the collider and its radius. The box collider is not the same because the radius is not consistent, and this cost additional background calculations. In order to be efficient and to decrease the number of the calculations per frame as much as possible, I implemented this feature using primitive colliders such as Boxes and Spheres along with Discrete collision detection. The use of more complex colliders such as rigid body can lead to higher precision, but the purpose of this game does not require that precision.

#### 3.3.3.4 User Interface Elements

User interface elements are the two-dimensional elements displayed on the screen during the game such as energy bar and current score.

Canvas [https://docs.unity3d.com/Manual/UICanvas.html]

Unity requires all the UI elements to be a child of a Canvas. Unity automatically creates a Canvas Game Object when a UI element is created. A Canvas can hold more than one UI element and renders its elements the same order they appear in the hierarchy.

##### Energy bar

The player’s energy is slightly decreased on each frame as well as on every movement to the left or to the right and while praying fish. It is increased when resting behind a rock.

The energy is displayed on the game scene as an energy bar. The energy bar is represented as a Unity UI build-in slider. The slider contains two parts:

* A background with a fixed position.
* A Slider which is displayed on the top of the background.

The slider has a minimum, maximum and current value and can be accessed and modified through “slider.value”. The player energy is controlled by a script, which uses an “Update()” function in order to update the slider.value on every frame.

##### Score

During the gameplay, the player has a score. The score is slightly increased every second and significantly increased when a coin or a fish object is being collected.

The Score is stored in the same canvas as the Energy bar. It is represented as a Unity build-in UI Text element. The text is accessed the same way as the slider is accessed and updated the same way as the slider. The difference is that the Text elements accept string values and the player’s score is a float variable. “(int)score.toString()” has been used in order to get rid of the floating point and to convert the int to a string, so it could be set as a value to the Text object.

#### 3.3.3.5 Storing the Player data locally

Storing the number of the player’s collected coins and the high score is done by serializing the “PlayerData” class. By serializing, is meant converting the public instance variables of the selected class to JSON format using JSON Utility: “{"highScore":0,"coins":0}”. The output of the conversion is stored as a String variable and the string content is saved in a .txt file using Unity’s system input/output library.

Saving data to a file is a relatively simple process - It takes one function call to File.WriteAllText(), which takes two String parameters – file path and content. The file path is determined by calling “Application.DataPath”, which returns different string results depending on the platform. For the iOS devices it returns the path to the application/Data folder and for the Android devices, it returns the path directly to the APK[[2]](#footnote-2).

[https://docs.unity3d.com/ScriptReference/Application-dataPath.html].

In addition to the application path, a file name and file extension are included at the end of the string using +”/save.txt”. On the other hand, there is a directory character separator which may be different for different operating systems i.e. ‘\’ on Windows and ‘/’ on MacOS. In order to generate a valid file path for different platforms, Unity provides “Path.Combine()” function which takes two strings ‘path1 and path2’. In that case, the “FilePath” variable is determined by “FilePath = Path.Combine(Application.DataPath, "save.txt");”, which excludes the use of any back or forward slashes.

The “Load()” function is working the other way around, it uses the same file path variable as in the save function, reads the “save.txt” file and then assigns its content to a string variable. The string variable is passed to the JsonUtility which de-serializes and overrides the public instance variables of “PlayerData” class.

#### 3.3.3.6 Mobile Input

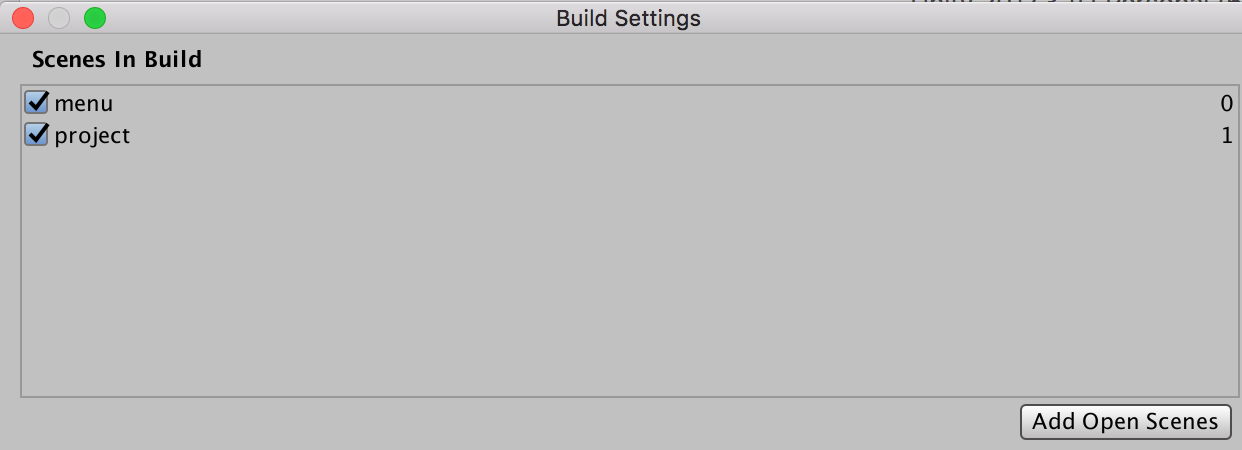
Since the game is developed to run on mobile devices, it is operated by a tactile input. Unlike the “Menu” scenes where the user is mostly interacting with buttons and does not require any specific code implementation, this is not the case for the “Game” scene. Similarly, to other endless world mobile games such as “Temple Run” and “Subway Surfers”, “Fish on the Run” game character will be moved by finger swipe screen gestures. The character has three positions – left, middle and right. In order to implement the swipe gestures:

1. Create “Mobile Input” script;
2. Detect for touch input from the user on each frame using the update function. If there is any input store it into an array.
3. On each frame -check if the input array Is empty. If It is not empty - there is a touch input.
4. There might be multiple swipes, but only the first swipe is considered.
5. Calculate the swipe direction - subtract the start position of the finger from the position when the finger Is removed from the screen. Both starting and ending positions are stored in Vector2 objects, keeping x and y coordinates. This means that if x > y, then the swipe is on the horizontal axis i.e. left or right. Otherwise, the swipe is vertical which corresponds to up or down movement.
6. There is a comparison whether the absolute value of x is bigger than the absolute value of y. Absolute value means that even If x is a negative number it is looked as a positive. This is because a left swipe would always have a negative x value, and, in that case, x will always be less than y. The game ignores vertical swipes and only proceeds if x > y (horizontal swipe).
7. Once the horizontal swipe is detected, there is a check whether x has a positive (left swipe) or a negative (right swipe) value. After the swipe direction is detected, an instance variable with data type Boolean and name corresponding to the direction is set to true (i.e. “swipeRight = true”). Those Boolean variables are set to false at the beginning of each frame.
8. “Mobile Input” class has a public static instance variable of type “Mobile Input”. This means that an instance of that class can be easily accessed everywhere in the other classes.
9. “Character” script is responsible for the player movement. On every frame its update function is using the public static “MobileInput” variable in order to check whether is a left or right swipe and move the character:

“if (MobileInput.Instance.SwipeLeft)”

During the implementation of the movement, there were some issues with following the game path and keeping a constant position. For example, the left lane corresponds to player position x = -3, mid lane x = 0 and right lane x = 3. For more information please read section 4….

#### 3.3.3.7 Change Scene

When a new scene is created and included in the build settings of the project it gets its own unique number. The Changing scene is an action that is typically invoked by clicking the UI element “Button”. 

Scene numbers - >

The clicked “Button” has the number hardcoded in it. When the button is clicked, the scene number is passed to a function with a body:

SceneManager.LoadScene(sceneToChangeTo);

Where “sceneToChangeTo” is an Integer that represents the unique number of the scene.

#### 3.3.3.8 Main menu scene

The main menu is the default scene loaded when the game is launched. The scene contains:

* Background image;
* UI Text element with the name of the game;
* UI Score element displaying the high score;
* UI Button element to start the game;
* UI Button element to ..

##### Background Image

It has been considered that the background image is going to be a. gif animation which occurred to be unsupported by Unity. It took several steps to get around this limitation:

* Split the .gif animation into multiple images using online tool. [https://ezgif.com/split]
* Create a script that has a reference to the background image and array keeping reference to every image with their/its right order.
* Create an infinite loop that changes the background image every 7 frames achieving the animation effect.

Fixing positions to the screen – different devices

Persisting Player’s Data

Local save and protection

Player prefs - simplest, hackable - text document

#### 3.3.3.9 Post-Processing Effects

Unlike “Unreal Engine” [ https://www.unrealengine.com/en-US/what-is-unreal-engine-4], Unity does not have built-in graphics effects but does offer a “Post-Processing Stack” asset that could be acquired for free from the Unity Asset Store.

Four of the post-processing effects were initially applied:

* Antialiasing – Aliasing is an effect where lines of the objects have a “staircase” appearance and appear on a low-resolution screen or displays. The antialiasing effect gives a smoother appearance to the models.

[ https://docs.unity3d.com/Manual/PostProcessing-Antialiasing.html]

* Fog – Hides the clipping of the farthest scene objects.

[ https://docs.unity3d.com/Manual/PostProcessing-Fog.html]

* Depth of Field [https://docs.unity3d.com/Manual/PostProcessing-DepthOfField.html] Specifies the distance of the objects that the camera is going to be able to focus.
* Colour Grading – Alters the colour of the final image (similar to applying filters to image i.e. Instagram filters). The applied filter is a Filmic Tone mapper. It brings a more filmic look to the scene which is achieved by altering the contrast, colour hue and saturation.

[https://docs.unity3d.com/Manual/PostProcessing-Stack.html]

Some of those effects were removed during the testing phase of the game (please read section 4…).

4 Testing  
https://docs.unity3d.com/Manual/iphone-iOS-Optimization.html

## 4.1 Issues found while testing on a mobile device

### 4.1.1 iOS Permissions

The implementation of the local player data storage, documented in Section 3.3.3.5 worked properly on the tests in the unity all-in-one editor, but that was not the case when this feature was tested on an iOS device.

iOS refused to give permissions to read or write a file and this resulted in a permission error in the Xcode debug console. Screenshot

In order to find a solution to that, it has been found out that “Application.dataPath” variable returns the path to the game folder and could be used in editors but is not appropriate for build games.

Alternatively to the “Application.dataPath” variable there is a “Application.persistentDataPath” variable which returns a path that is used to store data during runtime. That path is different for every platform. After a test on both Unity all-in-one editor and on an iOS device the replacement of “dataPath” with “persistentDataPath” variable worked properly.

https://answers.unity.com/questions/1181652/difference-between-applicationpersistantdatapath-v.html

### 4.1.2 Swipe Controls

##### How is the character movement implemented?

The game character is moving through x, y and z-axis with “characterController.Move(Vector3 \* Time.deltaTime)” function call on every frame:

* Character controller is the character’s collider.
* “Vector3” is a Unity structure that stores a 3d position on the game scene. In this case it stores the position that has to be added to the current one.
* “Time.deltaTime” returns the time it takes to execute the last frame and its value updated each frame. This variable is very useful in order to make run equally on different devices with different performance. For example, if a high-end flagship smartphone runs the game on 60 frames per second, this means that one frame is rendered in 1000ms / 60 fps ≈ 16.6ms per frame. Of course, not every frame is rendered in exactly 16.6ms because some frames may require the execution of more or less functions and operations which affects the time respectively positive or negative. Typically, the allocated memory in languages such as Java and C# are handled by a garbage collector (GC). When the GC is invoked the CPU, activity rises to 100% in order to perform its operations as fast as possible, which on the other hand affects the FPS negatively. In those terms if we consider two devices:

1. 30 FPS – 1 frame takes ≈ 33.33ms
2. 60 FPS – 1 frame takes ≈ 16.66ms

Since Unity measures Time.deltaTime in seconds and 1 second = 1000 milliseconds, the Time.deltaTime returns values such as 0.033 for device 1) and 0.016 for device 2).

Multiplying Vector3 \* 0.033 (device 1) will return a value two times higher than multiplying it with 0.016 (device 2), which means that the character for 1) will be moved twice as much as the character in 2). This is used because 1) will execute two times more frames per second and those multiplications are what makes the game runs equally on different devices.

##### How is the “Time\*deltaTime” multiplication causing an issue?

Similarly, to other endless world games such as “Temple Run” and “Subway Surfers”, “Fish on the Run” game is considered to have three horizontal positions – left, middle and right. All the game objects (such as collectable items, obstacles and other fishes) positions are also fixed:

* Left lane x = -3;
* Middle lane x = 0;
* Right lane x = 3;

This is why the main character movement should be fixed into those 3 positions. Starting from the middle lane x = 0, the character can be moved to the left or to the right by swipe gestures. Vector3 position contains three types of coordinates that determine what will be the player’s next position:

* x – left or right movement. 0 unless the detection of a swipe gesture.
* y – up and down movement. Always 0.
* z – forward and backward movement. Fixed by a variable that is slightly increasing every frame until the reach of a limit.

Movement attempts:

1. In the first attempt to move the character via swipe control I modified the Vector3.x value to -3 for a left swipe and respectively to 3 for a right swipe. As a result of that the character movement occurred to be barely noticeable. In that attempt the Time.deltaTime multiplication was not considered – 3 \* 0.016 = 0.048.
2. In the second attempt the Vector3.x value was set to 187.5, since 187.5\*0.016 = 3. As a result of that the character movement was correct, but in some of the swipes, it was easy to notice that the character was moving more or less than expected. Attempt 2 failed because as already explained in section 4.1.2 some frames take more than 16ms and some take less than 16ms.
3. In this attempt the movement invoked by tactile gestures was separated from the constant forward movement. In that case characterController.Move function is called in two cases:
   * + 1. Forward movement – Every frame “playerController.Move(moveVector \* Time.deltaTime);” With moveVector x = 0, y = 0, z = speed.
       2. If a swipe gesture is detected - “playerController.Move(new Vector3(-3,0,0));” Notice that in this case there is no Time.deltaTime multiplication.

In cases when a swipe gesture is detected case 1 and 2 are executed in one frame.

##### How colliders affect the sideways movement?

As explained in Section 3.3.3.3., the colliders can be:

* Non - triggered – Detects the collision and shows it.
* Triggered – Detects the collision but does not show it (i.e. objects can go through each other).

Typically, when the player collides with an object such as a fish or a coin a function that detects the collisions and disables that object (fish, coin). Using non – triggered collider makes the player character move slightly after the collision is detected and before the object is disabled. In a continuous game those slight movements are very noticeable and can affect the left, middle and right positioning of the character. The solution of this is to mark all the colliders as triggers except those that lead to death.

### 4.1.3 Post-Processing Stack

As explained in Section …, four filters are applied to the graphics of the game:

* Antialiasing
* Fog
* Depth of Field
* Colour Grading

##### Depth of Field unsupported issue

All of them worked properly during the tests on the Unity editor, but that was not the case with the iOS device tests.

The Depth of Field filter resulted in an error in the Xcode debug console, saying that this is an unsupported feature for the iOS.

##### Fog performance decrease issue

The Fog works only with the camera renderer set to deferred (with forward renderer as default). The deferred renderer is much heavier than the default one. There wasn’t any performance difference noticed while testing in the Unity editor, but that was not the case with the iOS device tests.

Interesting facts are that when the game is executed in the Unity editor it is executed as written in C# language and it is not optimized. When the game is executed on an iOS device (iPhone in this case), it is much more optimised and is compiled to CPP or ARM Assembly language.[<https://blogs.unity3d.com/2015/05/06/an-introduction-to-ilcpp-internals/>]

Even though the game was optimised it was apparent that the game is running very slow. After running it through Xcode it has been noticed that the frames per second were decreased from 60 to 16.

##### Profiler screenshots

##### As a result of the filters testing:

* The unsupported Depth of Field filter was deactivated.
* The Fog filter was deactivated.
* The renderer was set back to its default setting – forward.
* The default Unity fog was activated.

After applying the changes and repeating the Xcode testing procedure, the value of the game FPS on an iOS device was 60.

### 4.1.4 The problem of tunnelling

For every frame, the positions of the objects on the scene are calculated and checked for collisions. If one or more objects are moving very fast, they might go through each other. In that case the collision will remain undetected. This is called tunnelling effect. Bb reference

The tunnelling effect is noticed during testing the game on a mobile device. In a case when the character is collecting coins and is moving too fast, some coins remained uncollected. This means that the collision between the character collider and the uncollected coin collider is not detected which is an example of tunnelling effect. In order to fix that the size of the coin collider was increased.

Unity iOS allows you to change the frequency with which your application will try to execute its rendering loop, which is set to 30 frames per second by default. You can lower this number to save battery power but of course, this saving will come at the expense of frame updates. Conversely, you can increase the framerate to give the rendering priority over other activities such as touch input and accelerometer processing. You will need to experiment with your choice of framerate to determine how it affects gameplay in your case.

<https://www.youtube.com/watch?v=owZneI02YOU>

deferred rendering vs usual rendering

explain that fps counter from unity standard assets takes lots of resources and show it from the profiler.

Unit testing

Profiler

4.1. Testing within the Unity All-in-one editor

4.2. Testing on an iOS device

4.2.1. Uploading the project to the iOS device

4.2.1.1. Acquiring iOS Developer certificates

4.2.1.2. From Unity to Xcode

4.2.1.3. From Xcode to iPhone

4.2.2. Testing

Optimizing – occlusion culling

Player settings optimisation

Turning off code checking crashed the game on ios

forward 60fps vs differed rendering 16fps

Attaching water plane and background plane from the player and reattaching them to the camera because when scaling the player (while eating) the water plane and background plane are rescaled as well, and this breaks the game.

## 4.2 Security Testing

It is difficult to find security vulnerabilities for a game software project that is not even connected to the Internet. Typically, most of the software vulnerabilities does not only come from their network connections, but from their databases. As explained in Section 3.3.3.5., the user data is stored on local files using JSON Utility. The local files are simple text files and are human readable. In order to increase the number of coins the player needs only to open the file and modify its values. All the Android versions and iOS 11.0.0 + support File Explorer applications which allows the player to open and modify this file. Such a modification cannot do any harm, it would just make the game easier for the player.

To prevent such attacks the JSON Utility has been changed for Binary. This makes the player data file non-human readable and much more difficult to modify.

Add screenshots for hacking

Add examples for the JSON formatted files and binary formatted files

# 5 Evaluation

## 5.1 Survey

## 5.2 Any changes based on the survey

## 5.3 Critical Evaluation

## 5.4 Summary

Learned how to work with:

* Unity 3D
* Visual Studio
* GitHub
* Trello

Learned the basics of C# programming language. Learned about 3D games.

# 6 Future work

## Accessibility – Colour-blind mode

Networking??

https://forum.unity.com/threads/il2cpp-code-generation-options.367074/

# 7 Bibliography

[1] U. Technologies, "Unity - Manual: Prefabs", *Docs.unity3d.com*, 2018. [Online]. Available: <https://docs.unity3d.com/Manual/Prefabs.html>. [Accessed: 15- Feb- 2018].

At the end I will add a short description below each reference.

[2] "SceneKit - Apple Developer", Developer.apple.com, 2018. [Online]. Available: <https://developer.apple.com/scenekit/> [Accessed: 15- Feb- 2018].

[3] "About Simulator", Developer.apple.com, 2018. [Online]. Available: <https://developer.apple.com/library/content/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html>. [Accessed: 15- Feb- 2018].

[4] "Xcode - Apple Developer", Developer.apple.com, 2018. [Online]. Available: <https://developer.apple.com/xcode/>. [Accessed: 15- Feb- 2018].

[5] "Unity - Products", Unity, 2018. [Online]. Available: <https://unity3d.com/unity>. [Accessed: 15- Feb- 2018].

[6] "Displays", Developer.apple.com, 2018. [Online]. Available: [https://developer.apple.com/library/content/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html - //apple\_ref/doc/uid/TP40013599-CH108-SW1](https://developer.apple.com/library/content/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html#//apple_ref/doc/uid/TP40013599-CH108-SW1). [Accessed: 15- Feb- 2018].

**IEEE Citation Style**

1. Tunneling effect – When object is moving very fast and goes through another object between two frames, the collision remains undetected (i.e. a bullet goes through a wall). [↑](#footnote-ref-1)
2. Android Application Package Kit is the package format used by the Android OS for storing and distributing applications. https://www.androidpit.com/android-for-beginners-what-is-an-apk-file [↑](#footnote-ref-2)