City University London

BSc (Hons) Computer Science with Games Technology

Final Year Project Report

Academic Year 2019-20

Random Runner

A Randomly Generated Game Built with Unity

by

Sameer Ali

Project Supervisor

Dr. Christopher Child

Abstract

The aim of this project was to reproduce the success of well-established Endless Runner games such as Temple Run and Subway Surfers through the use of common features identified between them, by developing an engaging game that has high replay value, whilst also differentiating itself from them by providing a random level of difficulty as opposed to a gradual increase in difficulty.

This report explains how the Agile methodology was incorporated during the planning and development of the project in order to establish an organised and efficient plan that was acted on in order to produce a final build. The objective of the game is for the player to traverse the plane for as long as possible without colliding with the enemies that are generated in random positions on the plane. The primary focus of the project design was to implement the game world to appear as endless through the use of constant random generation of enemies as well as pick-ups that assist the player with manoeuvring and survivability.

This report will review the methods used in order to complete the project throughout its lifecycle including an evaluation of the methods and material used and also the results attained.

Contents

Chapter 1 - Introduction

The aim of the project was to create a game with the Unity engine that allowed for varied play-throughs via the use of constantly randomly generated objects. This document will outline the development process that was undertaken through design, implementation and evaluation of the project. This includes the methods used, literature review, outputs, results and conclusions.

1.1 Description of the Problem

The Temple Run series (Apple, 2020) [1], is an Endless Runner game that was released by Imangi Studios in 2011, which received over one billion downloads on mobile devices within three years of being released (Time, 2014) [2]. The game was considered as a giant success, resulting in a sequel being made as well as seeing strong competition from other developers in the mobile gaming industry. However, over recent years this popularity has fallen with less traction being seen for games within this genre (see Appendix A).

It was decided that the project would attempt to create a game within the genre by studying successful Endless Runner games to find the common themes between them and use this as a list of implementation features to work towards through development using the Unity engine (Unity, 2020) [3]. It was decided that the development of this game would attempt to differentiate a main feature that is present in these games, in being the gradual increase in the level of difficulty based on how far the Player traverses through the level, by focusing on the development of a completely random level of difficulty on each playthrough.

1.2 Project Objectives

The list below states the objectives of the project. The work was split into three iterations, each with a set of objectives to complete. The goal of the first iteration was to create a simple but playable game as a foundation for the following iterations. The second iteration focused on further expanding the first iteration by implementing the main functionality of the gameplay. The third and final iteration consisted of visual improvement goals that would be completed presuming there was enough time near the end of the project to do so.

1.2.1 First Iteration (Basic Functionality)

1.2.1.1 Plane Generation

1.2.1.2 Player Generation

1.2.1.3 Player Movement

1.2.1.4 Game Manager

1.2.1.5 Third-Person Camera

1.2.1.6 Enemy Generation

1.2.1.7 Collision Detection

1.2.2 Second Iteration (Main Functionality)

1.2.2.1 Plane Spawner

1.2.2.2 Object Spawner

1.2.2.3 Score

1.2.2.4 Object Models

1.2.2.5 Pick-ups

1.2.2.6 Destructor

1.2.2.7 Enemy Movement

1.2.3 Third Iteration (Final Enhancements)

1.2.3.1 User Interface

1.2.3.2 Sound

1.2.3.3 Camera Transition

1.2.3.4 Lighting

1.2.3.5 Skybox

1.3 Beneficiaries

The primary beneficiary was the author due to the challenge undertaken and experience acquired during the development of the project as well as allowing for the final build of the game to be used for a portfolio. The findings may also be submitted online to provide knowledge and encouragement for aspiring developers to consider during planning and development for their games. The final build was received positively through survey feedback, which may inspire a public release in future that would be beneficial for the mobile gaming community.

1.4 Assumptions

Due to not having previously used Unity before it was assumed after conducting research that it would be fairly straightforward to learn by following tutorials and making use of the resources available from the community to aid in developing the project. It was also assumed that this would be the most practical and effective way to meet the objectives that were set. There was prior limited experience with using other programming languages such as C++ and Java for the purpose of creating the functionality of a game for other modules and so, it was assumed that the transition of learning C# to write scripts for the functionality of the game that would be developed was possible.

Another assumption made was that assets could be found for the project in the UAS (Unity Asset Store) that would be free to use for any aspect of the game design that fit the needs of the objectives in order to provide an immersive experience for the user. Assets were required for 3D models, materials and sounds.

Lastly, an assumption was made that users would enjoy playing a game that provided constant random generation as a main proponent of the gameplay experience.

Chapter 2 - Output Summary

2.1 Unity Project Folder

File Type: .zip file directory

File Size: 47.9MB

The Unity project is located in a .zip file named “RANDOM RUNNER_Data.zip” with “Assets/” being the root directory of the project. The root directory acts as a central location containing code and assets.

C# Script files can be found in “Assets/RANDOM RUNNER/Scripts” and “Assets/Animation/Scripts”, the former being for gameplay Scripts and the latter being for User Interface Scripts. There is a total of 37 C# Script files containing 1536 lines of code (including comments), where 402 lines (including comments) where written by the author and the rest are re-used from the sources indicated in the Results and References sections. Imported assets can also be found in the “Assets/” directory which were also indicated in the Results and References sections.

The end user for the project folder would be anyone who is interested to know how the game “RANDOM RUNNER” was implemented using Unity, as well as to play the game for testing or recreational purposes. The code can be freely modified and the assets can be changed as unused assets were not removed from the project folder. The game can be built and run using the ‘build’ setting within Unity to create a new executable file. The project was built using Unity version 2018.4.2.22f1 (64 bit).

2.2 Windows Executable

File Type: .exe file

File Size: 635KB

The executable file can be run on a windows computer by an end user that would be interested in playing the game for testing and/or recreational purposes. The executable file was created using the Unity build function, which can be run by opening it.

Chapter 3 - Literature Review

3.1 Unity Engine

With regards to game development, a game engine is used by developers to build a game. There are two well established software engines to choose from that are free and easily accessible, being the Unity game engine and the Unreal game engine. Although both engines support the C# programming language, the main difference is that the Unreal engine is natively built using C++, which results in it also being the designated language to use for development in order for it to be effectively utilized. C# can be used in Unreal however, knowledge of C++ would still be required as the engine only uses this as an interface and so, it was likely that issues would occur if it was attempted to be used. After reading an article regarding which engine to use, it was decided that the project would be created through the Unity engine in order to meet the objectives of the project, particularly when the article directly mentioned Unity as being "originally designed to run on devices like consoles and phones" (Creative Bloq Staff. 2019) [4].

The documentation provided on the Unity website was then reviewed, which explains how it is used for game development through the use of "GameObjects, Components and Variables" (Unity, 2020) [5]. They are stated to be the "three fundamental building blocks" for gameplay and interactivity aspects within Unity that make it a unique environment to work with.

Unity provides various Components that do not require coding to function, such as the application of scaling objects. This was considered to be highly beneficial for implementation, as well as reassuring as opposed to using the Unreal engine where the same article mentioned that "you need a large and specialist team that's dedicated to different parts of the process" when attempting to create similar functionality, which was not possible.

3.2 Programming Language

As a result of deciding on using the Unity Engine for implementing my project, the programming language that would be used had to be reviewed. Unity supports programming through the use of scripts written in C#. After having reviewed the Scripting Overview section in the documentation provided by Unity on using classes, functions, and methods through C#, the author was reassured that the language would be suitable due to it having "some similarities to Java or C++", which were languages that the researcher had previous experience with. It also states that C# is a "managed language" meaning that the issue of 'memory leaks' would not be prevalent as this is managed within the environment, which would be of a greater concern for game development with C++. C# code is written in Unity via 'Scripts' that are attached to Components, which have properties made by Unity that can be stored in 'GameObjects' to affect their behaviour (Unity, 2020) [6]. Due to a lack of experience with Unity's scripting, the documentation provided by Unity was constantly reviewed. The tutorials available on the Unity website and YouTube, as well as the posts on the Unity Forum where frequently accessed in order to provide confidence in using this language.

3.3 Common Themes of Successful Endless Runner Games

In the thesis "Game Design Patterns In Endless Mobile Minigames", multiple common themes and patterns of games in the endless runner genre have been identified (David Cao, 2015) [7]. From the findings these patterns can be separated by importance by either being essential or not a necessity. Some examples of common essential themes for this genre of game is the use of an endless level, random level generation and obstacles. From the designs and patterns reviewed and compared it was assumed that these themes must be included and explored in order to re-create a successful game in this genre. It was then decided from these common themes found that in order to differentiate the game that would be developed based on the successful examples used in the thesis, an alteration would be made to adjust an essential pattern in these games, this being a change to the gradual difficulty increase in order to create a different challenge for the user.

3.4 Software Engineering Methodology

At the beginning stages of the project multiple methodologies were reviewed that could be applied to game development with regards to the scenario and feasibility for the time scales set and working preferences. It quickly became apparent that the most suitable route would be to use the Agile approach due to there being a lack of experience with the software that would be used as well as with implementing some of the objectives. As a result, this could have led to changes in requirements throughout the development and so, an Agile approach would best meet this criterion as it allows for quick action to be made throughout the development cycle when changes need to be made. Agile development focuses on iterative development, which was used to determine when certain objectives would be met for the project plan. From this methodology it was decided to use XP (Extreme Programming) as a main proponent for the structure due to the focus on providing continuous feedback and testing at each stage of development so that the project would be optimised to behave as intended and avoid making mistakes along the way to compensate for a lack of experience. LSD (Lean Software Development) was also undertaken to remove the need for paperwork to be written as in order to reduce time spent planning outside of the project.

Chapter 4 - Method

4.1 Software Engineering Methodology

The implementation of this project required aspects of Agile methodology to be incorporated in order for it to function successfully. Unity uses an ECS (Entity Component System) (Unity, 2020) [5], which results in each GameObject's functionality being based off a sum of the Component's that are attached to it. Therefore, the implementation of features and functionality was tailored to work through this model in order to best use the engine's performance and ability. This framework allowed for design or code changes to be easily adjusted if needed, which is a proprietary characteristic under the Agile methodology. It was decided to fundamentally base this on the concept of splitting the development of the project into three iterations.

The first iteration focused on developing the minimum functionality required in order for the game to be considered playable by allowing the Player GameObject traverse through the game whilst having to manoeuvre past the Enemy GameObject's to provide the initial challenge and objective of the game. The second iteration focused on the development of adding additional features and complexity to the 'level design' through the use of pick-ups, scoring, models etc., which would result in a unique experience on each playthrough and was closer to the expected functionality for the final game design. The third iteration focused on adding functionality that would make the user experience smoother, convenient and aesthetically pleasing through the use of adding sound, lighting and UI (User Interface) implementation. However, this iteration was not fundamental for the game to function as intended and so, less time was initially allocated in order to complete this.

Instead of producing use case tests to conduct at the end of each iteration, constant tests were conducted throughout the implementation phase whenever a feature was implemented, although this was not documented. This was decided to ensure that a significant amount of errors would not be present at the end of each build that prevented the code from being compiled and effectively stopping the game from running, as well as taking up large amounts of time to understand and rectify, which would become extremely difficult to track and resolve when following the ECS structure.

An initial work plan was devised to track the development progress (Appendix A) however, due to the severity of Covid-19, the project deadline was postponed by two weeks. This resulted in the deadlines set being adjusted to review the implementation of the final build and for further implementation of features to be considered.

OneDrive was used as a backup system and version control each time significant progress was implemented as well as for each completed iteration by saving the project to a zip folder that was then uploaded to OneDrive. The decision to use OneDrive was made due to Unity projects being inconvenient to configure for use with version control tools.

A number of assets from external sources were used in the implementation which were imported from the UAS as mentioned in the Results and References sections. Image files were used for various textures and GUI elements within the game. 3D models were used to represent all GameObjects in the gameplay scene. Sound files were used for in-game audio. As all the assets were imported from the UAS there were no issues that would require them to be edited using third party software.

4.2 Tools

Multiple tools were used to create and develop the technical side of the functionality as well as the visual side of the project through Scripts, Assets and UML diagrams.

4.2.1 Unity Game Engine

Unity is a free engine used widely to develop games at any level. Full documentation is readily available on the Unity website as well as having a large online community through the easily accessible tutorials, forums and assets available for use. After conducting research, it was decided that Unity was the most suitable engine to use with regards to the specification requirements of the project, and the UI was found to be straightforward to use for implementation.

4.2.2 MonoDevelop

Although Unity now uses Visual Studio as its default IDE (Integrated Development Environment), when the project was started Unity 2017 was already installed on the author’s computer, which used MonoDevelop as the default IDE (Unity, 2017) [8]. MonoDevelop was used by the author as no issues occurred as a result of using it. It was also supported by the Unity engine built-in debugger tool and so, was convenient to use.

4.2.3 Unity Asset Store

Multiple 3D assets were used that were provided from the UAS, as they would have to be compatible with the engine in order to be used in the project. This meant there would not be a worry regarding copyright issues. It also provided great convenience as this saved a great deal of time with having to experiment with assets that would normally be either created with a third-party tool or found online. All the assets used were free to download, which can be seen in Appendix D.

4.2.4 OneDrive
OneDrive is an online storage service provided by Microsoft that allows for a significant amount of data (up to 5GB) to be freely stored for safekeeping, sharing and later use. This was used for storing backup versions of the project into separate folders in case of emergency or if necessary, to access from another computer.

4.2.5 Survey Monkey
Survey Monkey is an online platform that can be freely used to create online surveys that can be shared easily for anonymous feedback (SurveyMonkey, 2020) [9]. This website was used to evaluate whether the objectives of the project were achieved through the feedback received from participants that watched a demo of the final build gameplay. The answers that would be collected focused on using an ordinal scale, providing the participant with a choice of what rating they would give to the related implementation. Once all the participants completed the surveys, the results would be analysed to review the most common ratings received of the game and its main features. As the survey was answered anonymously and did not require any details from the participants, a consent and/or participant information sheet was not required.

4.3 Iteration 1 (Basic Functionality)
For the first iteration the functionality was expected to be basic in order for the game to be run in a playable state. It was decided that a series of video tutorials would be used during the implementation for this iteration due to them focusing on designing a simple runner game (Brackeys, 2017) [10]. However, this would only be used for a starting point as this series looked at implementing manually designed levels instead of being endless.

4.3.1 Plane Generation
The first feature that was planned on being implemented was the Plane GameObject that the Player would be able to traverse as well as for Enemies to be placed on. As the level would be endless, the Plane would need to be extended/generated constantly in order for the never-ending affect to occur so that the gameplay would be prevented from ending whilst the Player was active.

4.3.2 Player Generation
For the first iteration the Player was planned to be implemented as a 3D cube GameObject that would be manually placed on the initial plane for movement and collision testing. This functionality would be implemented through the use of attaching Scripts to it.

4.3.3 Player Movement
As the aim of the game was to traverse as far as possible, the Player would be made to constantly move forwards, without the user having any control over this throughout every playthrough. Therefore, this would also result in the user not being able to adjust their experience to accommodate for a slower play-style, meaning that they would have to improve their awareness and reflexes to improve their scores. Thus, the implementation would focus on only allowing the Player to move either left or right.

4.3.4 Third-Person Camera
The Camera would need to follow the Player GameObject at all times otherwise it would move out of the user's view, making it impossible for them to know where to move in order to traverse through the level. It was also decided that a third-person camera would be used rather than first-person as this was more suitable for the user to manoeuvre the Player around Enemies appropriately.

4.3.5 Enemy Generation
For the first iteration the enemy was planned to be implemented as a 3D cube GameObject that would be manually placed on the plane for collision testing. This GameObject would then be stored as a Prefab that would be used in the second iteration when implementing the Spawner.

4.3.6 Collision Detection
In order for basic functionality to be tested, it was fundamental that collisions between GameObject's could be detected by the engine, which would then result in an event occurring. For this to happen, the GameObject's would have to have their own Box Collider Component that would trigger a Script to run when another Box Collider interacted with it to create a corresponding event. In this iteration the events that would occur were the Plane colliding with the Player and Enemy to prevent them from falling through it, and the Player colliding with the Enemy to end the game.
(See Figure 1 in Appendix D)

4.4 Iteration 2 (Main Functionality)

The second iteration objectives aimed to introduce visuals to the gameplay through animations as well as further technical functionality that would vary the gameplay experience on each playthrough. The implemented features required the functionality produced in the first iteration in order to behave as expected.

4.4.1 Random Plane Generation

This objective was planned to be implemented by writing a Script to randomly generate additional plane GameObjects that would either be placed along the x-axis or y-axis. This would result in the user having to adapt to a changing environment for a more difficult experience.

4.4.2 Object Spawner

This objective was fundamental for the gameplay to provide a challenge to the user. The implementation was planned to be able to produce a Spawner for each Enemy and Pick-up GameObject so that they would be spawned at random times and rates anywhere on the Plane. This would provide a unique experience on each playthrough if done correctly, as well as provide a random level of difficulty.

4.4.3 Score

In order for the user to have a goal that can be accomplished in an endless level, creating a functioning score system was fundamental. The main way of increasing the Player's score was to implement distance scoring so that the score would constantly increase based on the length of a playthrough. This would then need to be stored as a high score that could update whenever the current score in a playthrough exceeded the high score that had been previously set. This value would need to be permanently stored for when the game would be closed and re-opened.

4.4.4 Object Models

For the user to have an immersive experience, the Player had to be designed to have a model that would fit the theme of the game and genre. It was decided that the Player model would resemble a spaceship. This consequently resulted in the decision to model the Enemies as asteroids. Some of the enemy GameObjects were planned to be able to move towards to the player, which would need to be shown as rotating for it to appear immersive. The Pick-ups would also be uniquely identifiable through the models that would be used to display them whilst matching the theme.

4.4.5 Pick-ups

Pick-ups were to be implemented in order to provide the user with assistance in gaining a higher score due to the challenge of having to deal with randomly generated Enemy objects on each playthrough. The pick-ups were planned to be implemented in the same manner as the Enemies but at a lower spawn rate. This would allow for a risk/reward factor for the user as they may consider traversing along the x-axis in different ways in order to collect the Pick-ups so that they can make use of them to progress further in the level at the risk of colliding with an Enemy in doing so, which would then end the playthrough earlier.

It was decided that the Pick-up objects would also be able to rotate on a chosen axis to help the user distinguish them from the Enemy GameObject, as well as to also appear more attractive to collect for an immersive experience.

4.4.6 Enemy Movement

For this objective the planned implementation would follow the same implementation used for the Player movement functionality. This would be altered slightly by making an Enemy object move along the z-axis towards the Player, and the two others were made to move along the x-axis only in either direction. The speed that these objects traverse would also be slower than the Player's speed to prevent the object's from being impossible to avoid. This would however, provide a challenge for the user as they would have to constantly move the Player to avoid Enemy objects that are more likely to collide with it due to their movement. (See Figure 2 in Appendix D)

4.5 Iteration 3 (Final Enhancements)

The third iteration consisted of additional features that improved the experience of the game without adjusting or implementing functionality that would affect the core gameplay mechanics. However, for this reason the features implemented in this iteration were considered as somewhat indispensable if necessary and as lower priority for completion.

4.5.1 User Interface

The UI was planned to be implemented to be simple and convenient to navigate, allowing the user start the game, restart the level, return to the main menu, close the game, read the controls, and to also be able to adjust the sound of the game. It was decided that the UI would be clear, easy to use and access, as well as match the theme of the game.

4.5.2 Sound

Background music was planned to be implemented in order to provide a further immersive experience for the user through numerous songs that play consecutively on a loop. It was decided that the music would have to match the theme of the game but also provide a faster paced tempo to convey the feeling of urgency to match the gameplay.

Sound effects were planned to be implemented to play whenever a collision occurred between the Player and any other GameObject in order to notify the user that a collision occurred, as well as to provide an additional immersive technique to the experience. The sound effects would vary depending on what the player collided with to match the sound that would be associated with the consequent result and setting.

4.5.3 Lighting

As a result of the skybox being dark in order to match the theme of space, the scene had to be well lit in order for the user to see clearly where they are moving the Player. The user would also need to be able to see all the GameObjects from a reasonable distance in order to manoeuvre through the game. This would be implemented by using the Unity directional light to provide a light focused on the player, as well as on the Plane in front of the Player. This needed to be positioned and lit suitably in order to avoid overwhelming the scene as this would hinder the immersion of the theme and experience.

4.5.4 Skybox

As a result of deciding to implement models that fit into the space theme, it was also decided to implement a skybox that matched the theme to produce an experience where the immersion would be improved rather than ruined. This would also add a slight difficulty to the game as it would be harder for the user to see objects that are further away.

Chapter 5 - Results

The results section provides a summary of the code implemented and design that went into the project and the results that were developed. An evaluation was conducted after reviewing the results and testing was completed to view the outcome of the project's success. The original list of objectives shown in the PDD was used as an initial checklist for implementation. However, due to the nature of Agile methodology it was considered whether certain features were more suitable to implement before others throughout the development stage. This resulted in slight changes to the planned order of implementation of objectives within each iteration, for example it was decided on reflection that the score objective would be implemented before implementing the Coin Pick-up due to it being required for the Coin Pick-up functionality to behave correctly and be tested.

The code was often refactored and functions were reused or tweaked if necessary, such as when adjusting the Spawner variables for each Pick-up. Some GameObjects shared access to Scripts through Components in order to reduce tight coupling where possible, whereas in other cases new Scripts where created to separate common functionality to meet the intended criteria of specific GameObject behaviour.

5.1 Iteration 1 (Basic Functionality)

The first iteration features were implemented according to the work plan in the PDD as it was decided that each GameObject would be implemented in that order for functionality to behave as expected and tested properly (see Appendix A). The following features focused on developing the foundation of the basic gameplay through GameObject generation, collision detection, player movement and controls. The features were implemented following the tutorials made by Brackeys for the first iteration (Brackeys, 2017) [10]

5.1.1 Plane Generation

A 3D cube GameObject was created and scaled to the shape of a flat stretched Plane in order for the Player GameObject to be able to traverse it for a few seconds. The Plane GameObject consisted of the four necessary Component's created by Unity in order for it to display during gameplay, which are called Transform, Cube (Mesh Filter), Box Collider, and Mesh Renderer (Unity, 2020) [11] [12] [13] [14].

The Transform Component was used to position the Plane at the centre of the scene, and also allowed for the scale of the Plane to be adjusted to the shape of a stretched rectangle. The Mesh Filter was used to set the shape of the Plane mesh to a Cube shape, which was then rendered onto the scene by the Mesh Renderer. The standard shader that Unity provides for GameObjects was used for the Plane in this iteration. Lastly, the Box Collider was used to prevent GameObjects from falling through the Plane when positioned on it.

The Plane was then saved as a Prefab by dragging the GameObject into a folder named Prefabs within the project so that its Component's would be saved if the GameObject needed to be instantiated as another GameObject for the next iteration.

5.1.2 Player Generation

A 3D cube GameObject was created to be the player that would be controlled by the user for the first iteration. The Player GameObject consisted of the same Component's used by the Plane GameObject along with the RigidBody Component (Unity, 2020) [15], although some values were adjusted for the Player's functionality. The Transform component was edited to position the Player on the Plane. The standard shader that Unity provides for GameObjects was used for the Player in this iteration but the colour of the material was changed to a shade of red so that the Player was easily distinguishable from the Plane. The Box Collider was used to allow the Player to collide with the Plane so that it would not fall through the Plane when positioned on and traversing it. The RigidBody was used to apply mass and drag to the Player. This was also necessary when scripting the Player movement for velocity forces to function correctly on it. The Freeze Rotation constraint within the component was also applied to the Player on each axis to prevent the player GameObject from spinning constantly during gameplay (Unity, 2020) [16].

The Player was then saved as a Prefab by dragging the GameObject into a folder named Prefabs within the project so that its Component's would be saved if the GameObject needed to be instantiated as another GameObject for the next iteration.

5.1.3 Player Movement

In order to allow the user to move the Player during gameplay, the functionality had to be written in a Script as this followed the ECS paradigm that Unity uses, through encapsulating functions into their own Scripts, as opposed to the functionality of a Component that contains a single specific built-in behaviour or functionality. This was implemented by adding a New Script Component to the Player, which only accepted a Script to be attached to it. The script was named PlayerMovement and was written in C# using MonoDevelop.

The code was written using the FixedUpdate function as this is used on RigidBody Component's to apply a force to it on every frame, meaning that the Player's movement and position would not be affected by a difference in framerate of the device being used (Unity, 2020) [17]. This in turn allows the user to have the same experience on any supported device during gameplay.

The first line of the function applies a constant force on the x-axis on the RigidBody as long as time has elapsed during the playthrough, which was implemented through the use of the Time.deltaTime method (Unity, 2020) [18]. This results in the Runner aspect as the user cannot control the pacing of the gameplay. The user can move the Player using either the left and right arrow keys or the ‘a’ and ‘d’ keys on a keyboard to the matching x-axis direction for this iteration.

Lastly, the script checks the position of the player on the x-axis and runs the FindObjectOfType method (Unity, 2020) [19], to restart the game by calling a function created in the GameManager Script if the Player moves away from the Plane (see code below). This prevents the user from moving the Player out of the bounds that would be set for the Enemy to spawn in, as this would allow the user to avoid losing and the game from ending as a collision would not occur.

```
public void FixedUpdate() {

  rigidBody.AddForce (0, 0, forwardForce * Time.deltaTime, ForceMode.VelocityChange);

        //move player right
    if (Input.GetKey("d") || Input.GetKey("right")) {
        rigidBody.AddForce(sidewaysForce * Time.deltaTime,0,0,ForceMode.VelocityChange); }
        //move player left
    if (Input.GetKey("a") || Input.GetKey("left")) {
        rigidBody.AddForce(-sidewaysForce * Time.deltaTime,0,0,ForceMode.VelocityChange); }
        //checks the x position of the player to end the game
    if (rigidBody.position.x > 8.1f) {
            FindObjectOfType<GameManager>().EndGame(); }
        //checks the x position of the player to end the game
    if (rigidBody.position.x < -8.1f) {
            FindObjectOfType<GameManager>().EndGame(); }
    }
```

5.1.4 Game Manager

Although the implementation of a Scene Manager Script was not considered for the first iteration in the PDD and work plan, it became apparent when testing the Player’s movement and position that it would be convenient for testing to re-load the scene whenever the Player either moved out of bounds or collided with an Enemy. This was implemented through the SceneManager.LoadScene and Invoke functions (see code below) (Unity, 2020) [20] [21].

```
    bool gameHasEnded = false;
    public float restartDelay = 0.0f;

    public void EndGame () {
        if (gameHasEnded == false) {
            gameHasEnded = true;
            Invoke ("RANDOM RUNNER", restartDelay); }
    }
    void Restart () {
        SceneManager.LoadScene ("RANDOM RUNNER"); }
```

5.1.5 Third-Person Camera

The Main Camera GameObject provided by the Unity engine was used to display each scene. The Camera was implemented to follow the Player by creating a script called FollowPlayer that would be attached to the Camera. This simply required the Player's position value to be known while it was active on each frame for the Camera position to be updated. However, this resulted in the Camera being placed in the same position as the Player, causing it to be invisible and providing a first-person experience that was not intended. This was then resolved by making a public vector field so that it could be edited in the Inspector (Unity, 2020) [22], to set an offset to allow the Camera to be placed behind the Player position (see code below).

```
  public Transform player;
  public Vector3 offset;

// Update is called once per frame
    void Update() {
        //when the player is destroyed the position is no longer updated
        if (player != null) {
            transform.position = player.position + offset; }
    }
```

5.1.6 Enemy Generation

The Enemy was implemented as a 3D cube GameObject in order to provide simple collision testing with the Player. The Enemy consisted of the same Component's as the Plane but was scaled to be the same as the Player. The colour of the Enemy was not changed from the default grey in order to be distinguishable from the Player. The Enemy GameObject was then saved as a Prefab by dragging it in to the Prefab folder in order for multiple copies to be placed around the Plane for gameplay and testing purposes.

5.1.7 Collision Detection

Although collision detection was automatically provided between GameObject's through the Box Collider Component, this would result in the Player being prevented from moving forward when colliding with the Enemy until it moved around it, allowing the game to be carried on. In order to stop the game from continuing a script was created called EnemyCollision, which was attached to the Enemy and resulted in the Player being removed from the scene when colliding with the Enemy. This was written using the GameObject.Tag data type and Destroy method (see code below) (Unity, 2020) [23] [24].

```
 void OnCollisionEnter(Collision col) {
        if (col.gameObject.tag.Equals("Player")) {
            Destroy(col.gameObject);
            Destroy(gameObject);
            FindObjectOfType<GameManager>().EndGame(); }
    }
```

5.2 Iteration 2 (Main Functionality)

5.2.1 Plane Spawner

In order to produce an endless affect for the game the Plane would need to be constantly generated. This was implemented by writing methods in the PlayerMovement Script to constantly spawn the Plane on the z-axis at a fast speed in order to prevent the user from noticing, creating a seamless and immersive experience. The Plane GameObject had to be attached to the Script in the Inspector in order for its position to be used when generating a new Plane through the Instantiate method, which was written in the Spawning function (Unity, 2020) [25]. This function would then be called when the game starts by using the InvokeRepeating method within the Start function (see code below) (Unity, 2020) [26].

```
//the speed at which the plane spawns on the z axis
public float planeSpawner = 100f;

Vector3 firstposition;
Vector3 secondposition;

[SerializeField]
GameObject platform;

[SerializeField]
Transform firstobject;

void Start() {
    rigidBody = this.GetComponent<Rigidbody>();
    firstposition = firstobject.transform.position;
    InvokeRepeating("Spawning", 0f, 0.3f); }

private void Spawning() {
    GameObject _object = Instantiate (platform) as GameObject;
    _object.transform.position = firstposition + new Vector3 (0f, 0f, planeSpawner);
    firstposition = _object.transform.position; }
```

Unfortunately, it was found to be too difficult to spawn the Plane randomly in different positions on the x-axis and y-axis, whilst maintaining the Player and Enemy functionality and so, this functionality was not implemented.

5.2.2 Object Spawner

The random generation of Enemy objects on the Plane during gameplay was implemented by creating a GameObject that would contain a single Component for a Script to be attached that would generate the spawn values of the objects. The Script was called EnemySpawner and was written following the code used in a video tutorial (Rosscoe Tutorials, 2016) [27]. This script would require a GameObject prefab to be placed within the Component on the Inspector in order for it to function during gameplay.

In order for GameObject's to be constantly generated a 'Coroutine' was used as this is stated by Unity to be "like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame" (Unity, 2020) [28]. The Coroutine was named WaitSpawner, which would run once called by the Start function through the StartCoroutine method (Unity, 2020) [29]. The Coroutine would then be constantly executed due to the use of a while loop that would always be executed due to its bool condition always being met.

Position and spawn rate fields were then implemented through methods using the data types that would apply to them, which were then input in the Inspector to allow for multiple adjustments to be made to the generation different GameObject's using this Script. The values that were set in the Inspector would then be used by the Random.Range method to randomly generate the Enemy (Unity, 2020) [30], as a 'Clone' GameObject based on the set values that the Plane covers to make it seem as though the Enemy can only be spawned on the Plane. Lastly, the WaitForSeconds constructor was implemented to allow a set time to be input before the loop can be executed each time (Unity, 2020) [31], as well as a similar method called spawnWait being executed in the Update function to randomly set an additional random waiting time before an object could be generated (see code below). These variables were necessary to prevent the Spawner from generating too many Enemy GameObjects at a constant rate, which would stop the functionality of the game.

```
void Start () {
        StartCoroutine (waitSpawner()); }

void Update () {
        spawnWait = Random.Range (spawnLeastWait,spawnMostWait); }

    IEnumerator waitSpawner () {
        yield return new WaitForSeconds (startWait);

        while (!stop) {
            Vector3 spawnPosition = new Vector3 (Random.Range
        (-spawnValues.x, spawnValues.x), 1,Random.Range (-spawnValues.z, spawnValues.z));
            Instantiate (obstacle[randEnemy], spawnPosition + transform.TransformPoint
         (0, (float)-3.5, 0), gameObject.transform.rotation);
            yield return new WaitForSeconds (spawnWait); }
```

This Script was then duplicated and re-used for each Enemy and Pick-up GameObject to be generated through their own Spawner GameObject, with adjustments being made to alter their rate of generation.

5.2.3 Score

The score functionality was implemented by following a video tutorial (gamesplusjames, 2015) [32]. This was initially implemented by displaying text on the gameplay screen through the use of the Canvas GameObject (Unity, 2020) [33]. A Script called Score was then created and attached to this text field in order for it to increase while the Player was active (see code below).

```
public float score = 0f;

  //Update is called once per frame
    void Update() {
        if (scoreIncrease == true) {
            //increase score over time
            score += Time.deltaTime; }

        if (player == null) {
            scoreIncrease = false; }
```

However, the score would then reset to zero when the game ended. To provide greater functionality for this a high score value had to be implemented so the user could then set a score that would be saved known as a high score by managing to reach a score value that is higher than the displayed high score text value. This was implemented by setting the value reached on the score value to the high score value if the criteria were met of it being greater (see code below).

```
//sets the highscore requirement
        if (score > highscore) {
            highscore = score; }

//displays the score and highscore value on the screen
        scoreText.text = "SCORE: " + (int)score;
        highscoreText.text = "HIGH SCORE: " + (int)highscore;
```

Although this functionality worked during testing during a playthrough, it was noticed that both values would still reset to zero upon the game ending. This was resolved by allowing the high score value to be stored permanently using the PlayerPrefs method (see code below) (Unity, 2020) [34].

```
//Start is called before the first frame update
    private void Start() {
        if (PlayerPrefs.HasKey("Highscore")) {
         //retrieves the stored highscore value
         highscore = PlayerPrefs.GetFloat("Highscore"); }
    }

//sets the highscore requirement
        if (score > highscore) {
            highscore = score;
            //saves the highscore permanently so it does not reset
            PlayerPrefs.SetFloat("Highscore", highscore); }
```

5.2.4 Object Models

It was decided that the Player GameObject would use the Spaceship model (see Appendix D) from the asset that was imported from the UAS (Ebal Studios, 2019) [35]. Once imported, the chosen model prefab was placed within the GameObject, which then had to be scaled accordingly to be visible in the scene, as well as have a matching Collider size, and the Camera offset had to be adjusted in order to prevent issues with functionality.

It was decided that the Enemy GameObject's would use the Asteroids model's (see Appendix D) from the asset that was imported from the UAS (VK GameDev, 2019) [36]. The chosen model's prefabs had to be scaled accordingly to be visible in the scene, as well as have a matching Collider size in order to prevent issues with functionality.

It was decided that the Coin Pickup GameObject would be implemented by using the Energy Cell model prefab (see Appendix D) from the asset that was imported from the UAS (AntiDed GameDev, 2019) [37]. The chosen model prefab had to be scaled accordingly to be visible in the scene, as well as have a matching Collider size in order to prevent issues with functionality.

It was decided that the Time Slowdown Pickup GameObject would be implemented by using the Sand Hourglass model prefab (see Appendix D) from the asset that was imported from the UAS (3d.rina, 2019) [38]. The chosen model prefab had to be scaled accordingly to be visible in the scene, as well as have a matching Collider size in order to prevent issues with functionality.

It was decided that the Invincibility Pickup GameObject would be implemented by using the edited Shield model prefab (see Appendix D) from the asset that was imported from the UAS (VK GameDev, 2019) [39]. The chosen model prefab had to be scaled accordingly to be visible in the scene, as well as have a matching Collider size in order to prevent issues with functionality.

5.2.5 Pick-ups

The functionality for the Coin GameObject was implemented by creating a Script called CoinScore that would destroy the object when it collided with the Player. It was also decided that this collision would increase the score value by ten therefore, a method was written in the Score Script to do this, which would then be called by the CoinScore Script (see code below).

```
//Use this for initialization
    public Score scoreCoins;
//OnTriggerEnter is called when a collision between GameObjects occur
    private void OnTriggerEnter(Collider other) {
        //increase score
        scoreCoins.scoreCoin(); }
//scoreCoin is called to increase the score
    public void scoreCoin() {
         score+= 10f; }
```

It was also decided that this object would constantly rotate, which was easily implemented by attaching a Script called PickupRotateXY (see code below).

```
// Update is called once per frame
    void Update() {
        if (gameObject.name == "Coin Pickup(Clone)") {
        transform.Rotate (0, 180 * Time.deltaTime, 90 * Time.deltaTime); }
```

The functionality for the Time Slowdown GameObject was implemented by following a video tutorial in order to temporarily reduce the game speed when this object collided with the Player (Brackeys, 2017) [40]. This was initially implemented by creating a Script called TimePickup that was attached to the object in order to destroy it upon collision with the Player.

A second Script called TimeManager was then made for the functionality to be produced by using the Time.timeScale, Time.unscaledDeltaTime, Time.fixedDeltaTime, and Mathf.Clamp methods/variables (Unity, 2020) [41] [42] [43] [44]. During testing it was identified that although the Time.timeScale variable slows the game speed this is not displayed properly as it severely affects the framerate, which was resolved by adjusting the Time.FixedDeltaTime value. Another issue was then identified where the game speed would not revert back to its normal speed due to this not being set, which had to be resolved by making a calculation with the slowdown duration and the Time.unscaledDeltaTime to occur in the Update function to set the Time.timeScale back to normal. However, this would result in the game speed constantly increasing each time the method was called. This was resolved by setting the Time.timeScale value after it had returned to normal to stop increasing through the Mathf.Clamp method. This script was then attached to a newly created GameObject named TimeManager, which was then attached as a variable to the Time Slowdown GameObject through the Inspector.

```
//Use this for initialization
    public float slowdownFactor = 0.1f;
    public float slowdownLength = 10f;

    //Update is called once per frame
    void Update() {
        Time.timeScale += (1f / slowdownLength) * Time.unscaledDeltaTime;
        Time.timeScale = Mathf.Clamp(Time.timeScale, 0f, 1f); }
    //Slowdown is called to reduce the game time
    public void Slowdown() {
        Time.timeScale = slowdownFactor;
        Time.fixedDeltaTime = Time.timeScale * .02f; }
```

The Slowdown function is then called by the TimePickup Script to produce the functionality if the collision occurs (code shown below).

```
void OnTriggerEnter(Collider other) {
        if (other.name == "Player") {
            timeManager.Slowdown();
            Destroy(gameObject); }
}
```

It was also decided that this object would not rotate to signify time being frozen to the user for immersion, and for the object to be distinguishable from the other moving pick-up objects.

The functionality for the Invincibility GameObject was implemented by editing the PlayerMovement Script to disable the Player Box Collider Component for a set amount of time after colliding with the Invincibility object (see code below).

```
//OnTriggerEnter is called when a collision between GameObjects occur
    private void OnTriggerEnter(Collider other) {
        if (other.name == "Invincibility Pickup(Clone)") {
            Destroy(other.gameObject);
            gameObject.GetComponent<BoxCollider>().enabled = false;
            Invoke("setTriggerTrue", 3.2f); }
    }

    //setTriggerTrue is called to set the Box Collider value of the GameObject
    private void setTriggerTrue() {
        gameObject.GetComponent<BoxCollider>().enabled = true; }
```

It was also decided that this object would constantly rotate, which was implemented by attaching a Script called PickupRotateX.

5.2.6 Destructor

After testing the Enemy Spawner it was identified that the performance of the game would be significantly slow and eventually freeze due to massive amount of GameObject's being active in the scene as they would never be removed. Although this was not considered in the PDD and work plan this had to be dealt with. In order to resolve this, two methods were implemented to remove the GameObjects over the time of the gameplay.

The first method used was to implement a 3D sphere GameObject which followed the Player that would destroy any GameObject's that it collided with but would never collide with the Player. This was simply implemented by positioning it to be behind the Player in the scene and then placing the GameObject within the Player GameObject, which resulted in it moving in the same directions as the Player. The scale of the object was greatly increased so objects could not be avoided when it moved along the x-axis due to Player movements. A script called DestroyAll was then attached to it, which used a collision trigger to destroy anything that collided with the object.

The second method used was to destroy an object by attaching a Script called EnemyDestroy to it that would remove it from the scene after a set amount of elapsed time from when it was generated, which was implemented after seeing a post on the Unity forum (Unity, 2013) [45]. It was decided that the time before an object would be destroyed would be fifteen seconds as this was a suitable amount of time for the object to be active in the scene before no longer being able to impact the Player (see code below).

```
//Use this for initialization
    float lifetime = 15f;

    //Update is called once per frame
    void Update() {
        //applies methods to relevant GameObjects
        if (gameObject.name == "Asteroid(Clone)") {
            Destroy (gameObject, lifetime); }
```

These methods greatly stabilised the gameplay performance as GameObject's were destroyed at roughly the same rate that they were being generated. This prevented the user experience and immersion from being ruined due to stutters or freezing that would make the game unplayable.

5.2.7 Enemy Movement

In order to provide a challenging experience, it was decided that Enemies would be generated that would move on either the x-axis or the z-axis. This was implemented by duplicating the Enemy GameObject three times and assigning each new object a different model. The PickupRotateX Script was attached to the original Enemy GameObject as well as the newly created GameObject's that would move along the x-axis. The PickupRotateXY was attached to the GameObject that would move along the z-axis towards the Player. This applied rotation to each of the enemies, which provided immersion as asteroids do rotate.

A script called MovingAsteroidX was attached to the Enemy that moved along the x-axis to the right of the Player. A script called MovingAsteroid_X was attached to the Enemy that moved along the x-axis to the left of the Player. A script called MovingAsteroidZ was attached to the Enemy that moved along the z-axis to the right of the Player. These scripts were implemented to simply provide a force to the related GameObject in the same way that the Player would move.

5.3 Iteration 3 (Final Enhancements)

5.3.1 User Interface

The UI was initially implemented after creating a new scene called Menu, which was used to create the main menu UI. This UI was implemented by importing an asset made by Unity from the UAS (Unity, 2018) [46]. Conveniently, this asset was designed to match the theme of space and so, the visual look of it did not need to be adjusted. A particle system was also included which provided a pleasing aesthetic of moving particles when the menu was loaded. The asset also provided its own functionality through the Scripts and Components that were stored in it, which allowed for the created GameObject buttons to be easily edited to meet the functionality listed in the project objectives. A Button Component was attached to each GameObject in order for the button's to be interactable when pressed (Unity, 2020) [47].

The menu was edited to remove the provided Continue button (see Figure 4) as this was not set as an objective due to the nature of the gameplay. The Start button in the edited main menu (see Figure 5) was set to load the gameplay scene when clicked by running the StartGame function from the attached Menu script, which used the static method SceneManager.LoadScene to switch the scene based on the manually set order of the scenes in the engine build settings (see code below).

```
public void StartGame() {
        SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex + 1); }
```

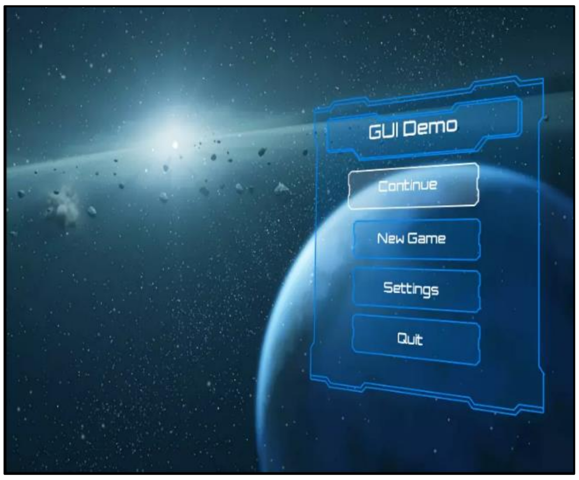


Figure 4: The originally imported Menu Settings asset

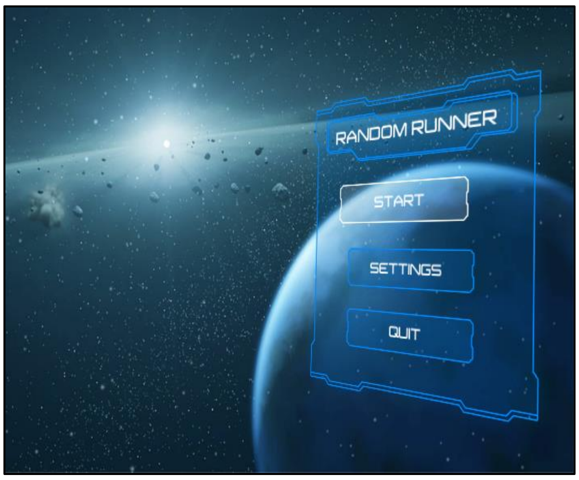


Figure 5: The edited Menu Settings asset

The Menu settings panels from the original asset used visual sliders and toggles (see Figure 6) along with matching components to allow the user to easily interact with and change settings if the necessary functionality was implemented through attached scripts (the asset did not have any).

The Help panel (see Figure 7) was adjusted to display the relevant GameObject's that the user would need to be aware of to be able to expect what they should collide with and what they should avoid colliding with in the gameplay scene. The objects also rotated in this panel the same way they do in the gameplay scene to aid the user in identifying them from a further distance.

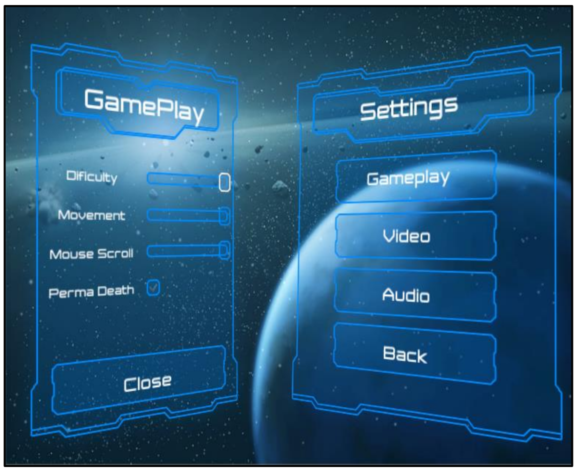


Figure 6: The imported Menu Settings Panel

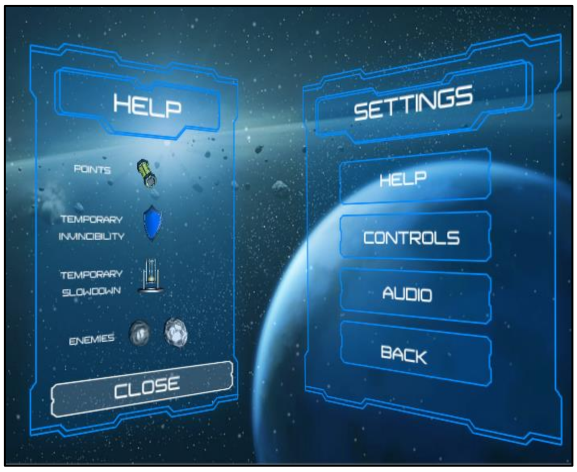


Figure 7: The edited Menu Settings Panel

Icon images were imported into the project through an asset that was found on the UAS (Kvazi, 2018) [48]. The images were then applied to the toggle GameObject's through the use of the Image component (Unity, 2020) [49]. The Controls panel showed the user which button corresponds with each movement direction (see Figure 8), although the icons would help signify this if it was not viewed by the user.

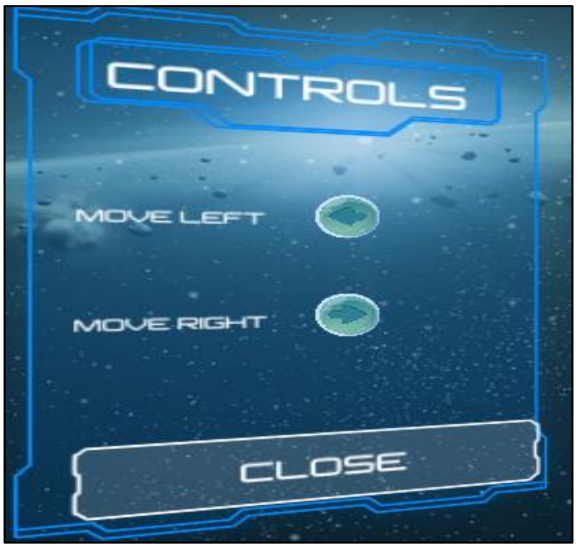


Figure 8: The edited Controls Pane with Icon images

The Audio panel would allow the user to toggle the background music audio when clicked (see Figure 9) through the use of the Toggle Component (Unity, 2020) [50]. This was implemented by initially creating an empty GameObject named SoundManager within the scene that the Audio Source Component was then attached to in order for the chosen audio track to be played when the scene was loaded. The SoundManager was then attached to the Toggle Component so that the

AudioSource.mute function would be called whenever the toggle was clicked in order to mute or unmute the audio track, which was provided by the Audio Source Component (Unity, 2020) [51].

The image was also edited using the Toggle Component to change to show a chosen image when clicked before reverting back in order to provide feedback to the user that the toggle state had changed (see Figure 9 and 10).

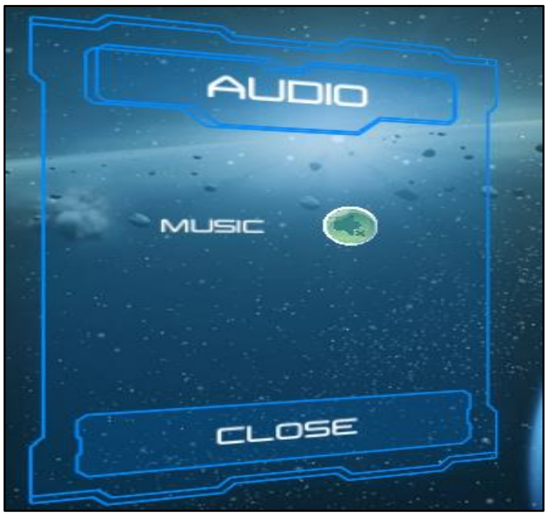


Figure 9: The edited Audio Panel with Mute toggle image

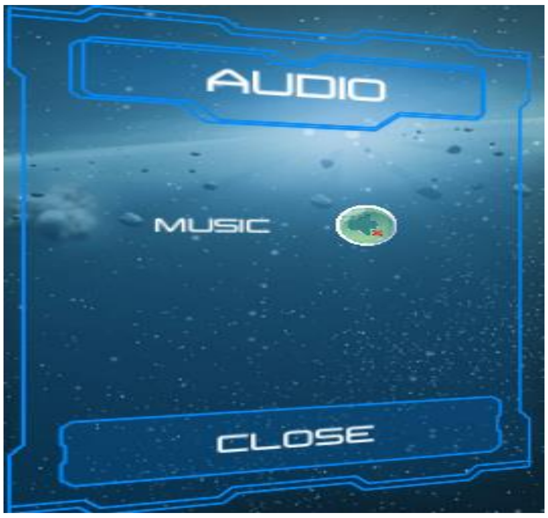


Figure 10: Mute toggle image changing when clicked

After the playthrough ends in the gameplay scene due to the player colliding with an enemy or going out of bounds, a Restart Menu had to be implemented to display for the user to either retry the level, return to the main menu scene, or to close the game. In order to do this the Canvas GameObject was used to store an edited panel to provide this functionality, which was set to render once the player had been destroyed in the GameManager script.

The game would also pause when this occurred in order to stop GameObjects from spawning by setting the timescale static variable, within the Time class to zero (see code below). This functionality meant that an animation could not occur on Player death so this was not implemented.

```
//TimeRestart is called to pause game time when restart menu is displayed
void TimeRestart() {
restart.gameObject.GetComponent<Canvas>().enabled = true;
Time.timeScale = 0f; }
```

However, through testing it was found that the timescale would not reset when the scene was re-loaded or switched, which made it unplayable. To resolve this, functions were written to set the timescale to one once the scene had been updated (see code below).

```
public static bool GameIsResumed = false;

    //Start is called before the first frame update
    public void Start() {
        //set game time to play when game is restarted
        if (GameIsResumed == true) {
            Time.timeScale = 1f;
            GameIsResumed = false; }
    }

    //Update is called once per frame
    private void Update() {
        if (player == null) {
            TimeRestart(); }

        //set game time to play when game is restarted
        if (GameIsResumed == true) {
            Time.timeScale = 1f; }
    }

    //Restart is called when the restart button is clicked
    public void Restart() {
        GameIsResumed = true;
        SceneManager.LoadScene ("RANDOM RUNNER"); }

    public void MainMenu() {
        GameIsResumed = true;
        SceneManager.LoadScene ("MENU"); }
```

Lastly, due to the game being intended to be played on mobile it was decided that the user would be able to move the Player using the icon images displayed in the Controls panel. This was implemented by rendering the GameObject's for these images in the Canvas and then adjusting their position to match the side of the screen that the Player would move. An EventTrigger Component was attached twice to each image GameObject so that the player movement functions could be called when they were pressed and held (Unity, 2020) [52]. The PlayerMovement Script had to be edited to account for the GameObject's being held, which was implemented through the code used from a video tutorial (Awesome Tuts, 2018) [53]. Each EventTrigger Component would then store the functions AllowPlayerMovement and DontAllowPlayerMovement, with the EventTrigger that moved the Player left having a set bool state enabled and the other being disabled in order to register which direction the Player would move to.

5.3.2 Sound

Sound was implemented for both background music and SFX (sound effects). The background music was imported into the project through an asset that was found on the UAS (Breitbarth, 2017) [54]. This audio asset was chosen as it matched the theme of the game and could be listened to on a loop as the general sound was maintained throughout the track providing a pleasant ambience. However, the tempo of the beat in the track was somewhat fast paced, which has been found to "affect simulated and perceived driving speed" (Brodsky, 2001) [55]. This would result in the user having to deal with a subliminal challenge through the feeling of a sense of urgency, which also fit the gameplay experience through the nature of survivability and driving.

It was decided that the background music would use this track for both the main menu and gameplay scene. The background music was implemented by creating a new GameObject named SoundManager. The Audio Source Component was then added to it as this is used for playing sounds in a 3D environment (Unity, 2020) [56]. In the Inspector this component was then configured to play the chosen audio track within the asset when enabled. The component also easily allowed for the audio to be played on start-up as well as to be looped through the checkboxes it provides.

After implementing this the audio successfully played on start-up in the main menu scene however, when loading the gameplay scene, the audio would stop playing. The simple way to resolve this was to implement the SoundManager in this scene also and configure its audio source component in the same way but this then resulted in the audio starting again, which would be noticeable if the user spent a period of time on the main menu before playing the game. This aspect was felt as taking some immersion away from the seamless feel of the gameplay and so, this was acted on.

In order to prevent the audio from being loaded re-loaded when the scene would change, the SoundManager GameObject in the main menu scene had to not be destroyed when loading the gameplay scene. As a result, this feature was implemented through the use of a Singleton class that was written within the MusicInScenes Script (see code below) through the DontDestroyOnLoad static method after reviewing the documentation (Unity, 2020) [57], as well as on a forum post on the Unity forum page (Unity, 2020) [58].

```
        get {
            return instance; }
    }

    //Awake is called to run the check on startup
    void Awake() {
        if (instance != null && instance != this) {
            Destroy(this.gameObject);
            return; }

        else {
            instance = this; }
        DontDestroyOnLoad(this.gameObject);
```

This script was then attached to the GameObject which allowed the SoundManager to continue playing the audio track when the scene switched, providing the seamless effect to assist with immersion (see Figure 11).

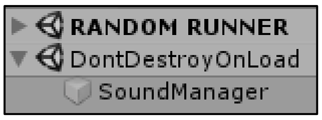

RANDOM RUNNER
DontDestroyOnLoad
SoundManager

Figure 11: Screenshot showing the SoundManager being retained after a scene switch has occurred, through the DontDestroyOnLoad static method

The SFX used were imported into the project through an asset that was found on the UAS (Zero Rare, 2017) [59]. The SFX were implemented into the gameplay scene by initially adding the Audio Source component to each relevant GameObject rather than on a SoundManager. Each component would then store the chosen audio track that matched the action that the GameObject would produce as a result of a collision with the Player. The audio track would then need to be played when the corresponding collision would occur, which was implemented by initializing each component in the script that was attached to each GameObject, and then creating an instance of the component in the Start function that would then be called in the collision function by using the AudioSource.Play function (Unity, 2020) [60].

However, an issue arose when testing this implementation where the SFX audio would not play when a collision occurred due to the GameObject being destroyed before it could be called. This was resolved by implementing a long enough delay on the destruction of the GameObject, in order for the SFX audio to play (see code below).

```
Destroy (gameObject, 0.3f);
```

5.3.3 Camera Transition

As an Enemy GameObject called Chasing Asteroid was implemented in the second iteration that followed the Player in order to act as a GameObject destructor, it was decided that a camera transition would be implemented to briefly show this before being set to follow the Player. This idea was not considered for the first iteration in the PDD and work plan but was inspired by the transition that occurs in the Temple Run series (Imangi Studios, 2018) [61], where the Camera transitions from in front of the player (see Figure 12) to behind the player (see Figure 13) after showing the enemies that are chasing it through rotation to provide context for the reason to constantly move forward.

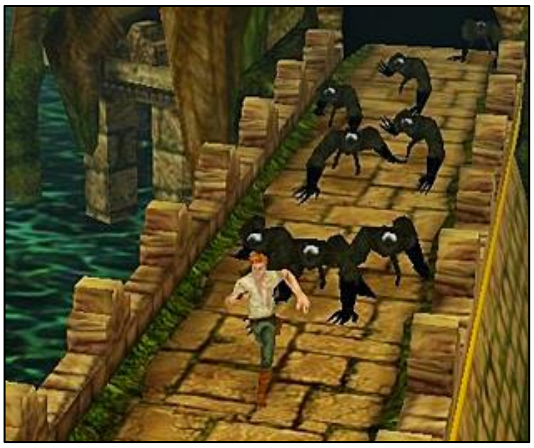


Figure 12: Camera starting position on start-up

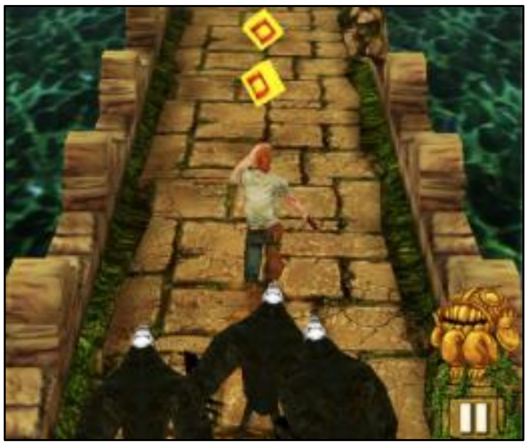


Figure 13: Camera rotates behind the player

This feature was implemented by editing the FollowPlayer script attached to the Camera GameObject to use the Lerp function provided in the documentation (Unity, 2020) [62], as well as through code posted on the Unity forum page (Unity, 2020) [63] (see code below).

```
//Update is called once per frame
void Update() {
    if (player != null) {
        transform.position = Vector3.Lerp (transform.position, transform.position =
        player.position, Time.deltaTime);

        //find the vector pointing from our position to the target
        _direction = (Target.position - transform.position).normalized;

        //create the rotation we need to be in to look at the target
        _lookRotation = Quaternion.LookRotation (_direction);

        //rotate over time according to speed until we are in the required rotation
        transform.rotation = Quaternion.Slerp (transform.rotation, _lookRotation,
        Time.deltaTime * RotationSpeed); }
```

The Camera was then positioned and rotated accordingly through its Transform component to display this transition rotation correctly however (see Figure 14 and 15), through testing it was shown that this implementation resulted in the camera being placed further behind the player once the transition had ended, as well as the camera moving strangely when the player moved along the x-axis, which provided an inconvenient gameplay experience. This was resolved by implementing a timer that would set the camera position to the originally set position in the FollowPlayer Script once enough time had passed for the transition to occur (see code below).

```
public float timer = 0f;

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

      timer++;
       //follow player while it is moving
       if (timer > 400) {
           Timer(); }
     }
```

```
    //Timer is called to set the camera position on the player GameObject
    void Timer() {
        transform.position = player.position + offset;
        transform.rotation = player.rotation; }
```

A Script was also created called PickupRotateY and was attached to the Enemy GameObject so that it could be seen rotating in order to signify to the user that it was moving towards the Player.

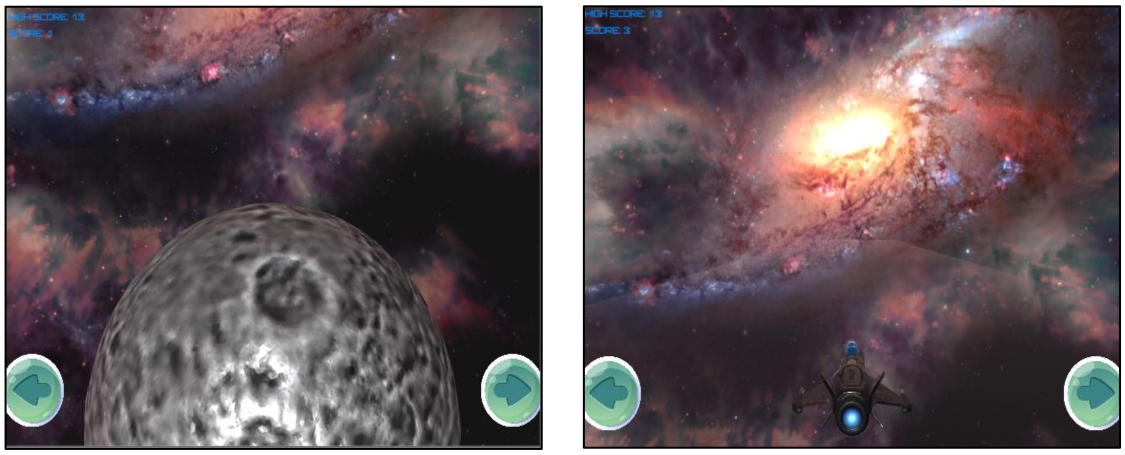


Figure 14: Camera transitions from the position on the left image to the right image at the start of each playthrough to show that the player is being chased.

5.3.4 Lighting

The Directional Light GameObject that is automatically created by Unity when making a new project was found to be suitable for use in the final iteration. It was minorly adjusted by changing its rotation through the Transform Component to make the light point down on top of the GameObjects that spawned on the Plane. This would assist the user in identifying GameObjects readily as their axis rotation would be better seen whilst the Camera approaches them.

5.3.5 Skybox

As the game would be played using a third-person Camera and the user could not rotate the Player, a 2D Skybox was applied to the game scene to that fit the theme of being in space for visual and immersive purposes. The Skybox asset was found on the UAS, and then imported into the project (Pulsar Bytes, 2017) [64]. The desired Skybox material was then applied to the game scene by adding it to the skybox material field in the Unity Lighting Settings following the documentation (Unity, 2020) [65]. The Camera component within the Camera GameObject then needed to be updated to display the Skybox to the user, which was done by selecting the Skybox option in the Clear Flags field.

5.3.6 Plane Transparency

A transparent shader was applied to the plane in order for the user to see the Skybox below the player for the immersive experience that being in space would provide as a result of being infinite in its depth, which assisted in providing an endless effect (see Figure 14). This was implemented by changing the GameObject default shader material properties in the Inspector to make the shader transparent. The Transparent option was selected in the Rendering Mode field and the Albedo field was adjusted to set the colour value to be transparent, whilst the Smoothness field was set to one. Although the Plane became see through it was not made to be completely transparent so that it could be used to guide the user to stay within the set bounds, reducing the likelihood of the game ending as a result of moving out of bounds.

Chapter 6 - Conclusions and Discussion

6.1 Project Objectives

The purpose of this project was to develop an Endless Runner type game with the Unity engine which featured themes that are commonly shared between successful games in the genre. Some of the common themes mentioned in Section 3.1 were endless level, random level generation and obstacles. It was decided that this game would be differentiated from others to have a slightly different experience by not following the implementation of gradual difficulty within the endless level. This meant that set patterns would not be implemented in the level generation which tend to be shown through having different areas, assets and obstacles, and instead the decision would be made to make the generation of the enemies completely random. This would then result in a truly random experience for the consumer on each playthrough.

The success of this implementation was measured and evaluated by the ratings that the game received from survey participants (see Appendix B). The survey received 11 responses with the design of the game receiving 'Very Good' as the highest response, the random generation also receiving 'Very Good' as the highest response, the likelihood of the same being played by the participant if available receiving 'Highly Likely' as the highest response, and lastly when asked how likely they would be to play the game more than once if available, the highest response was 'Likely'. Therefore, it can be assumed that the project was successful at attempting to emulate the success of Endless Runner games.

Agile methodology appeared to be a sensible choice and fairly successful as most of the initially planned objectives were completed whilst also allowing for room to adjust, change, and remove any objectives without halting progress in each iteration. The workload was managed sensibly due to this and did not result in progress being halted significantly at any point due to having to wait to work on a particular objective before being able to work on another one.

OneDrive was helpful for maintaining backups of the project at each iteration and fortunately was not used to restore work to an earlier backup as there were no incidents of hardware failure or errors with the code that would cause the project to not work.

Random generation of the Enemies was implemented through the knowledge provided in tutorials as planned, and tweaked accordingly through testing and experience. The playable character was successfully implemented, which collided with objects and moved as expected. The score and Pick-ups functioned as expected allowing for an engaging experience. The user interface was implemented to look simple and clear to prevent any confusion when interacted with.

The project was successfully built as an executable file that would run on Windows, which was used for testing. The project should also run on Mac and Linux operating systems although this was not tested as it is not part of the project scope. The project could also be exported to mobile platforms where the intended consumer would expect to be able to play the game however, this was not tested as it is not part of the project scope.

6.2 Personal Development

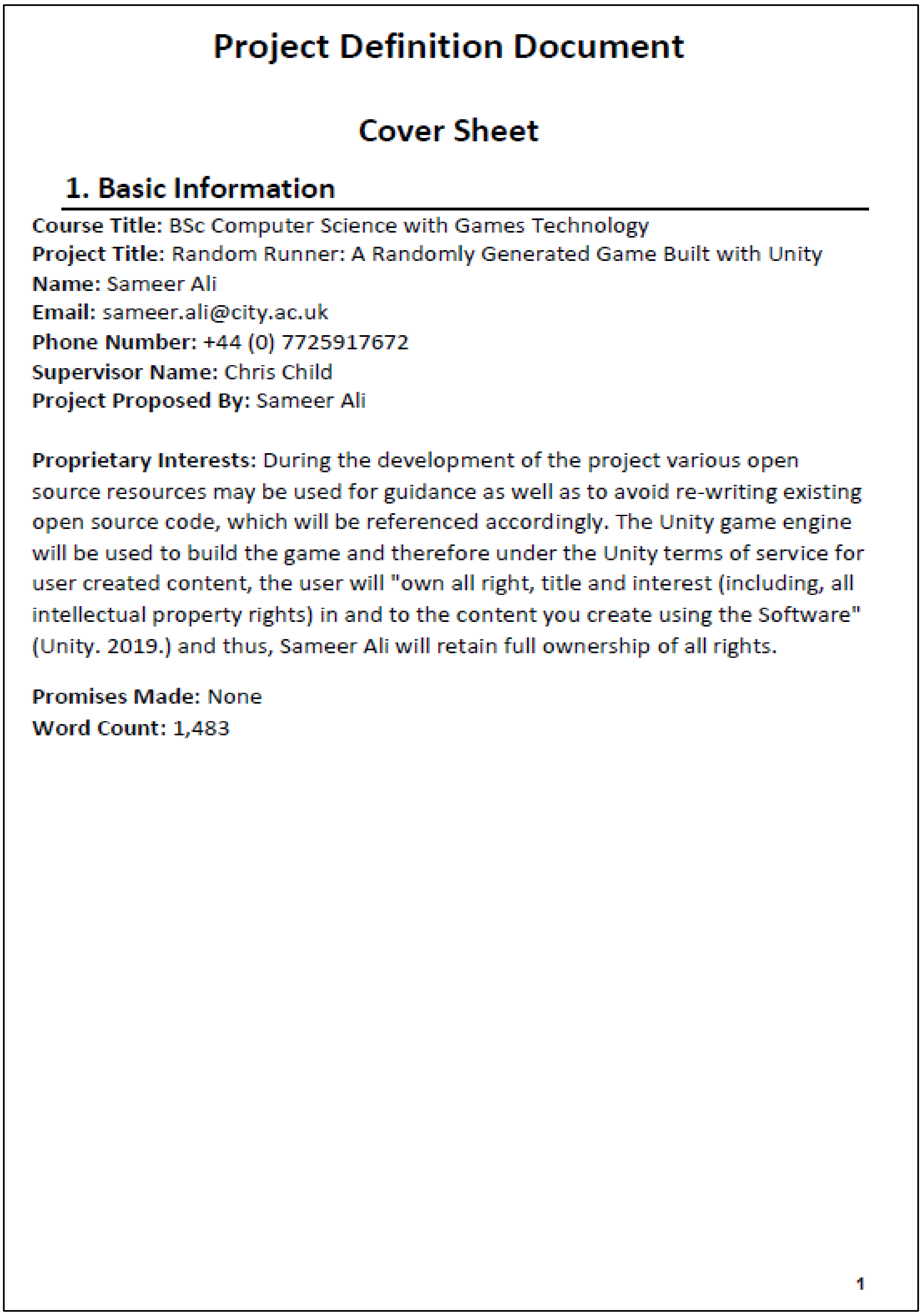
The project was found to be extremely important and beneficial with regards to aspirations of working in the games industry due to the experience and knowledge gained of design, development and implementation when using the Unity engine for working on a project individually. The importance of being well prepared in order to maintain a heavy workload and organisation was constantly present through the use of following a structure set by following a software methodology in order to produce consistent productivity at all times during the cycle of the project design and development. Lastly, the knowledge gained with regards to scripting and programming provided confidence and reassurance, which was found to be rather difficult to understand previously.

6.3 Further Improvements

Although the project was considered to have been successful and productive there were multiple improvements that could have been made throughout the design and development of the project. Given more time and experience it would have been ideal to implement additional features and design Component's to improve the playing and visual experience for the user such as implementing a high score table or a shop system to unlock further cosmetics through playtime, in order to maintain the audience's interest and attraction. However, as the game is not intended to be released commercially, it was decided that there would be less time allocated to these ideas so that the main functionality would be prioritised in order to maintain a balanced workload.

Appendices

Appendix A – Project Definition Document



Project Definition Document

Cover Sheet

1. Basic Information

Course Title: BSc Computer Science with Games Technology
Project Title: Random Runner: A Randomly Generated Game Built with Unity
Name: Sameer Ali
Email: sameer.ali@city.ac.uk
Phone Number: +44 (0) 7725917672
Supervisor Name: Chris Child
Project Proposed By: Sameer Ali

Proprietary Interests: During the development of the project various open source resources may be used for guidance as well as to avoid re-writing existing open source code, which will be referenced accordingly. The Unity game engine will be used to build the game and therefore under the Unity terms of service for user created content, the user will "own all right, title and interest (including, all intellectual property rights) in and to the content you create using the Software" (Unity. 2019.) and thus, Sameer Ali will retain full ownership of all rights.

Promises Made: None
Word Count: 1,483

1

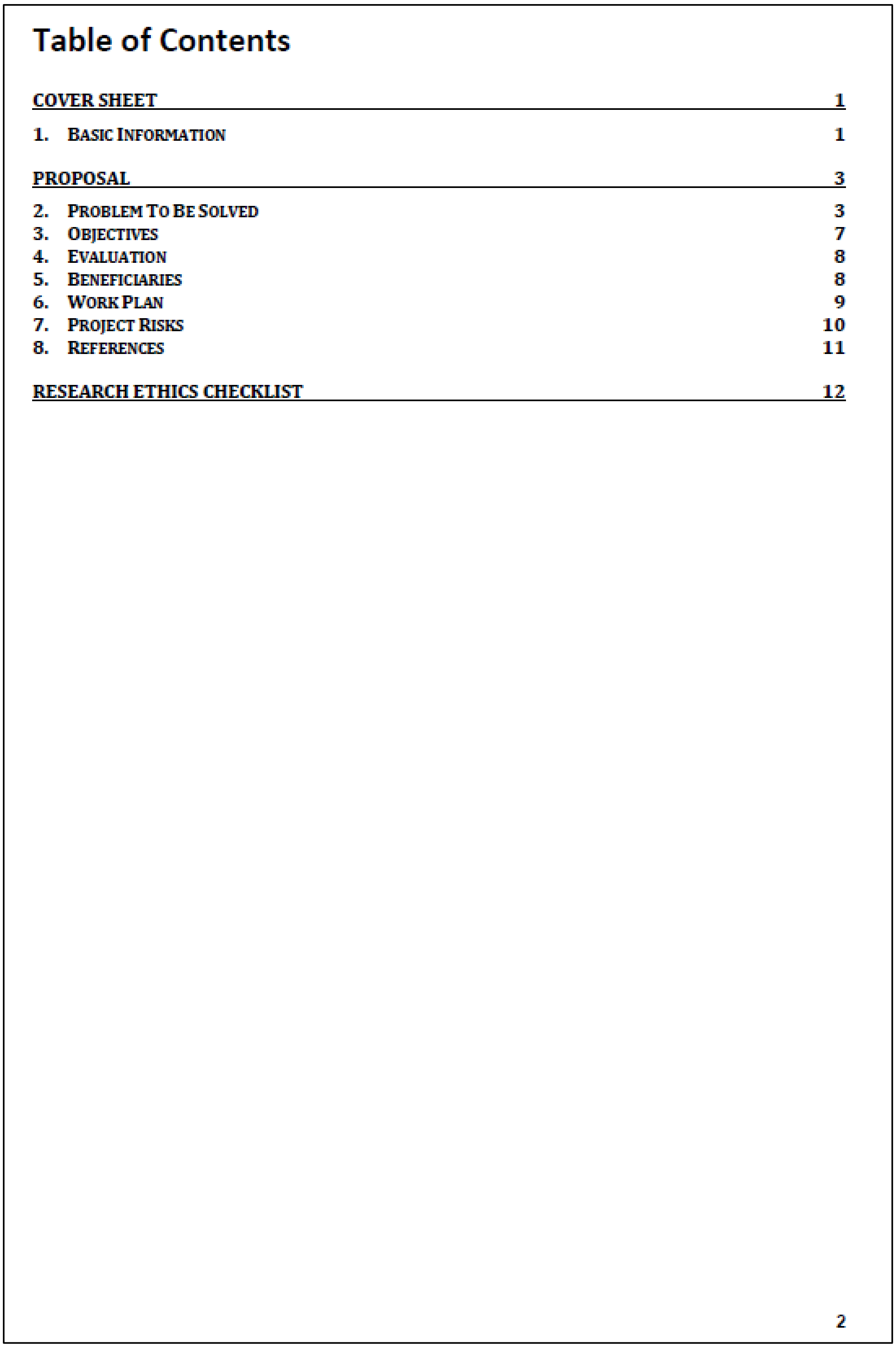


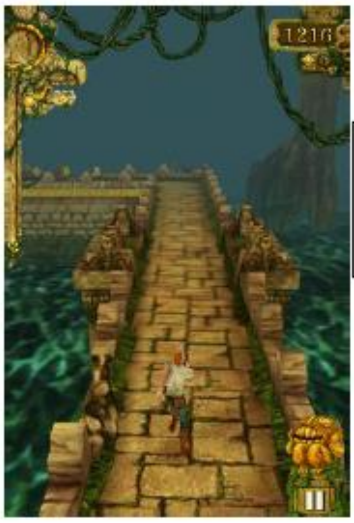
Table of Contents

2

Proposal

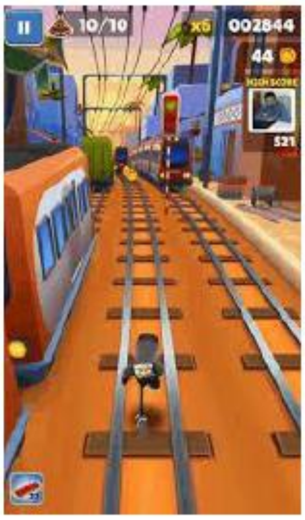
2. Problem To Be Solved

The 'endless runner' genre has become a prevalent staple in the world of mobile gaming over the last decade. This genre emerged as a result of the widespread success that '2D platformer' games received such as Super Mario in the 90s, inspiring the development of mobile games within this genre due to their simplicity as well as a 'pickup and play' feel that mobile consumers are known to want as a result of consumption habits following a more short-term attention span (Ben Chong, 2015) [1]. "Mobile phones are the most commonly used devices for playing video games" (Limelight Networks, 2019) [2], with the Apple App Store having its own dedicated section for browsing through games to install and play.



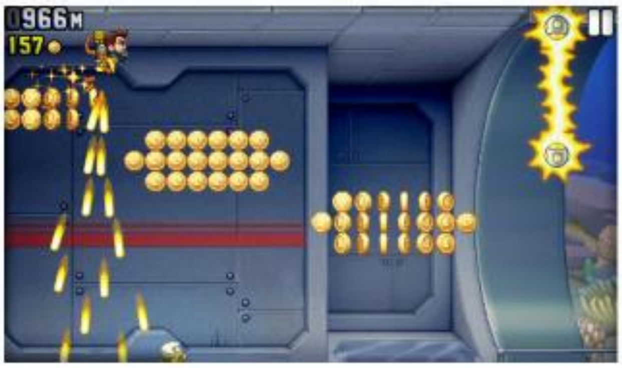
Temple Run
Imangi Studios, 2011

[5] 3D Third Person endless gameplay image



Subway Surfers
Kiloo & SYBO Games, 2012

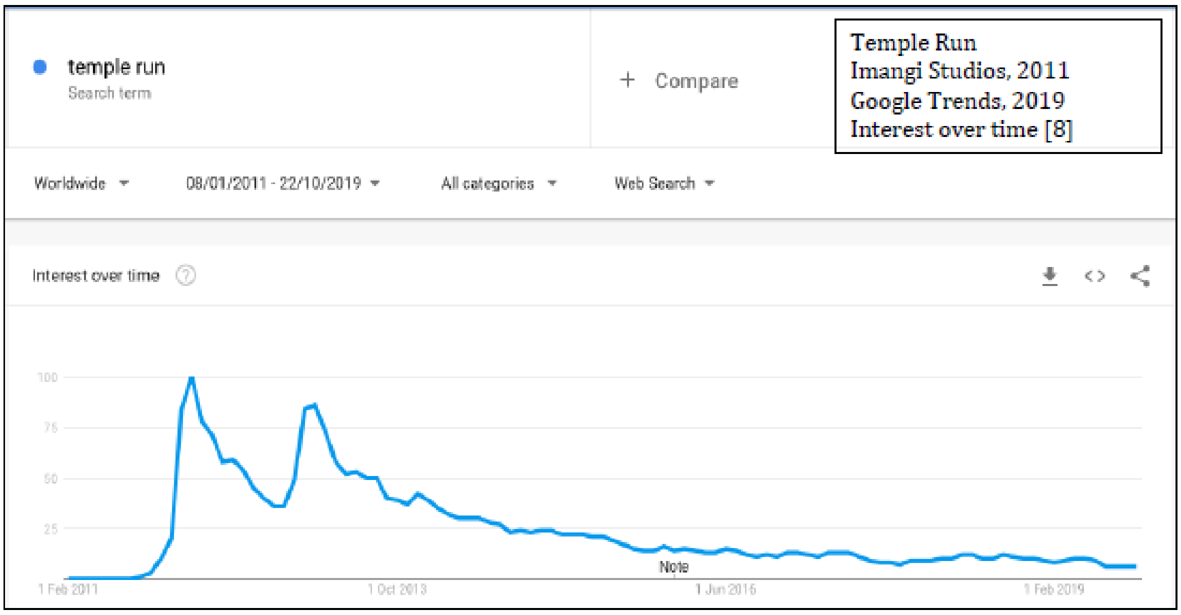
[6] 3D Third Person endless level gameplay image



Jetpack Joyride
Halfbrick Studios, 2011

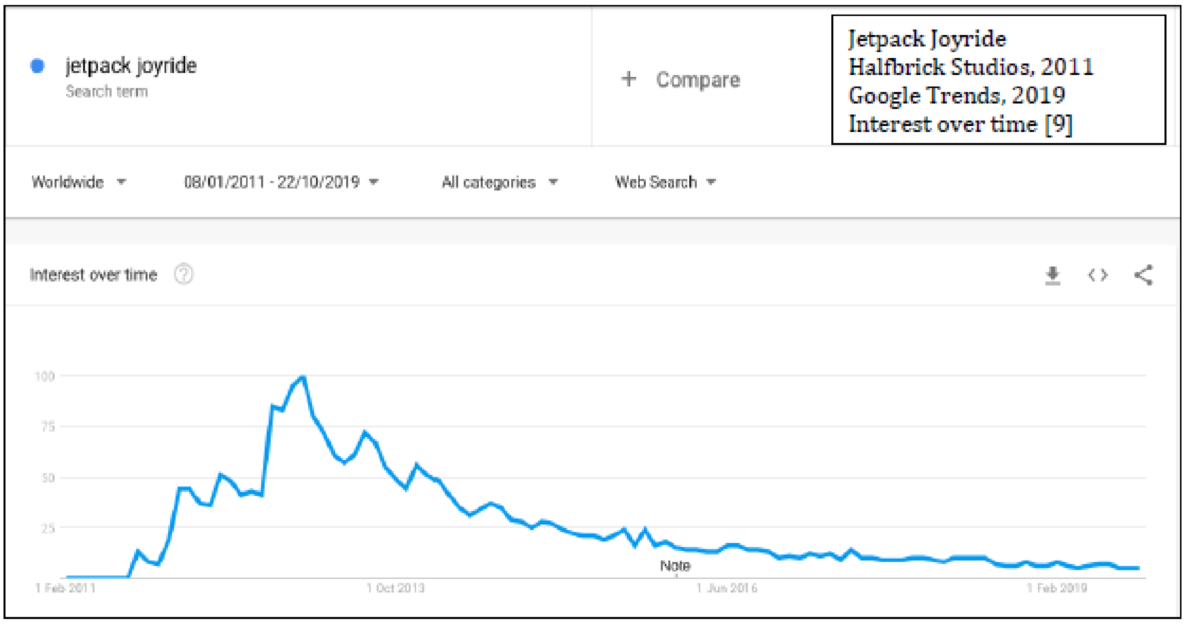
[7] 2D Side Scroller endless level gameplay image

The fundamentals of an endless runner consist of the player moving through a never-ending area, where the objective is to get as far as possible, increasing their score whilst not being able to stop until they lose all lives and repeat the process to set a higher score. Although this may seem like a very simple concept this genre was once at the top of the mobile gaming world with games such as the Temple Run series (Imangi Studios, 2011) [5], receiving over 1 billion downloads in 2014 (Matt Peckham, 2014) [3]. However, since then the popularity of games within this genre has seen a constant decline. A simple search on 'Google Trends' provides a visual statistical of data over the last decade regarding the interest shown for 3 of the biggest games in this genre.


temple run
Search term
Worldwide
08/01/2011 - 22/10/2019
All categories
Web Search
Interest over time

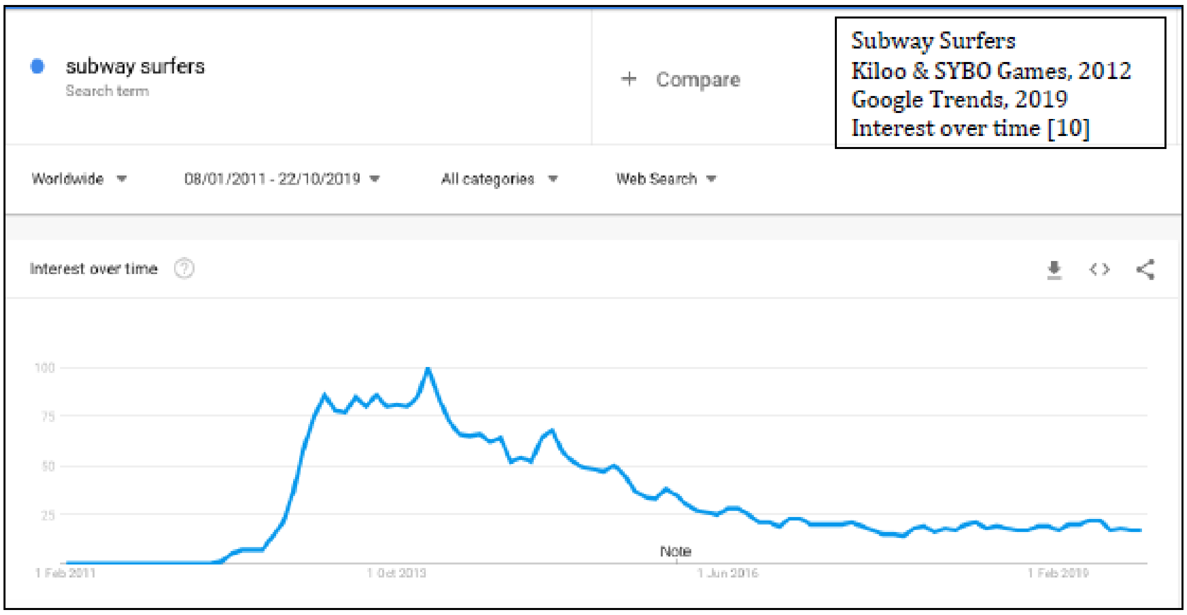
Temple Run
Imangi Studios, 2011
Google Trends, 2019
Interest over time [8]

4


Jetpack Joyride
Halfbrick Studios, 2011
Google Trends, 2019
Interest over time [9]

jetpack joyride
Search term

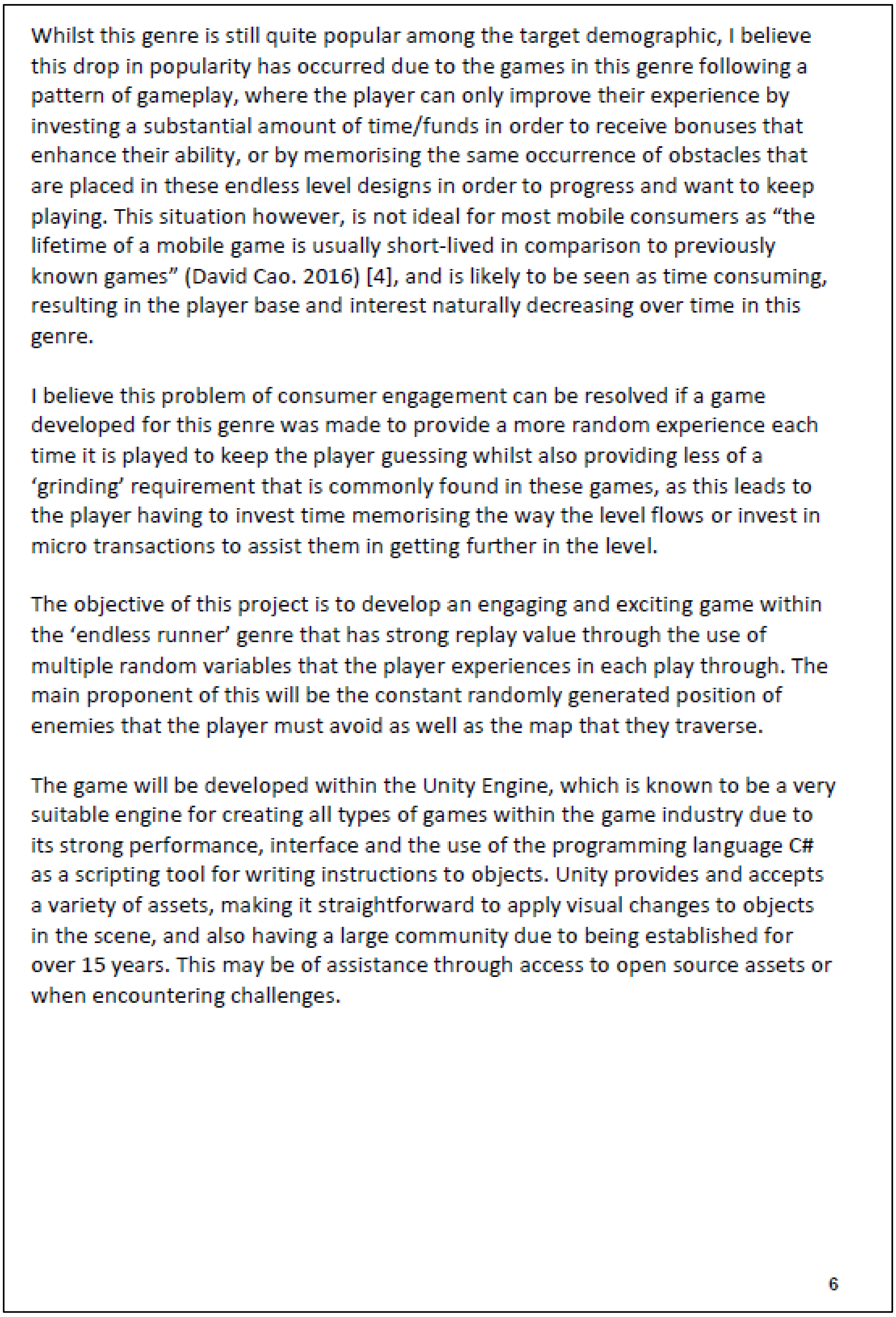
Worldwide · 08/01/2011 - 22/10/2019 · All categories · Web Search


Subway Surfers
Kiloo & SYBO Games, 2012
Google Trends, 2019
Interest over time [10]

subway surfers
Search term

Worldwide · 08/01/2011 - 22/10/2019 · All categories · Web Search

5

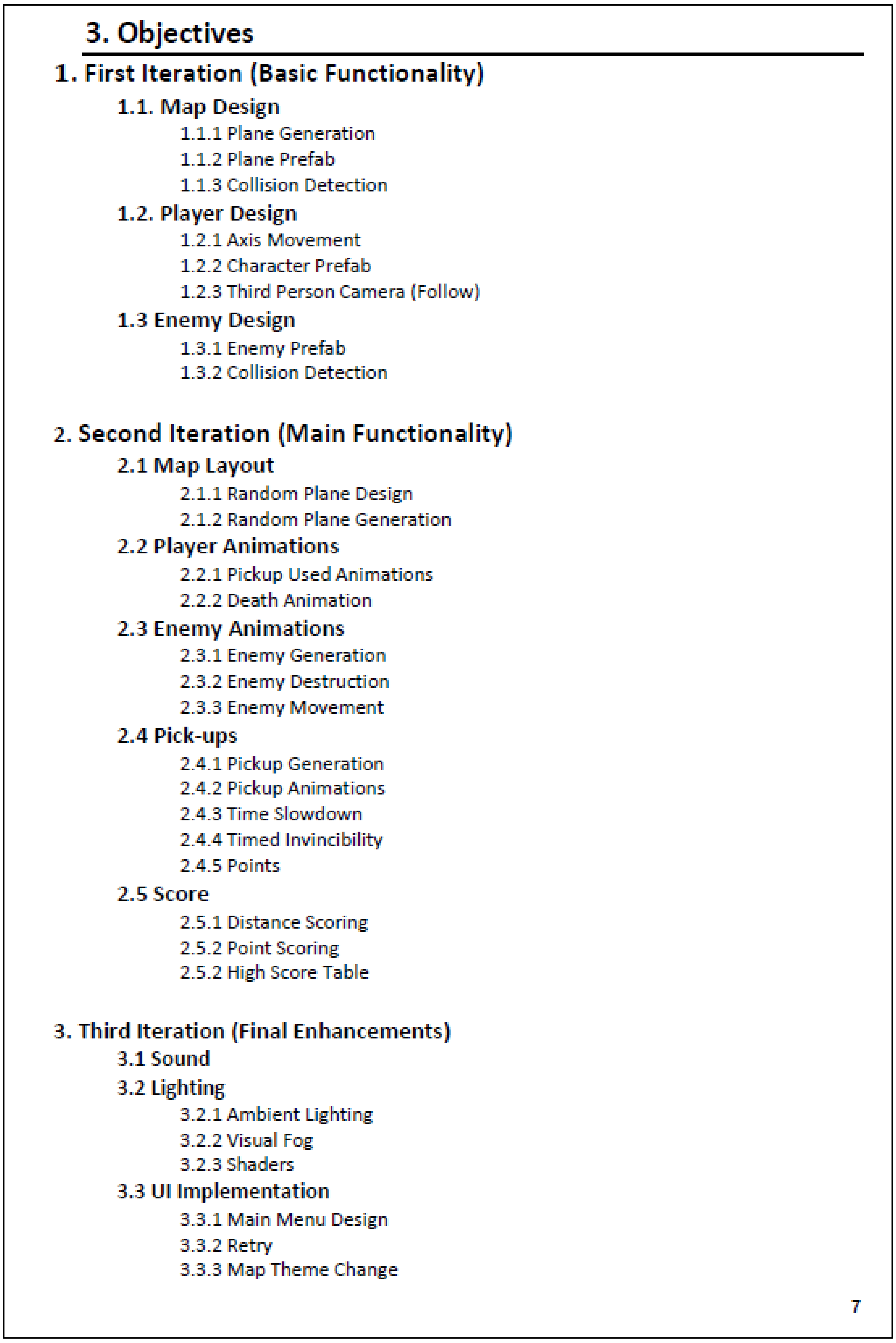


Whilst this genre is still quite popular among the target demographic, I believe this drop in popularity has occurred due to the games in this genre following a pattern of gameplay, where the player can only improve their experience by investing a substantial amount of time/funds in order to receive bonuses that enhance their ability, or by memorising the same occurrence of obstacles that are placed in these endless level designs in order to progress and want to keep playing. This situation however, is not ideal for most mobile consumers as "the lifetime of a mobile game is usually short-lived in comparison to previously known games" (David Cao. 2016) [4], and is likely to be seen as time consuming, resulting in the player base and interest naturally decreasing over time in this genre.

I believe this problem of consumer engagement can be resolved if a game developed for this genre was made to provide a more random experience each time it is played to keep the player guessing whilst also providing less of a 'grinding' requirement that is commonly found in these games, as this leads to the player having to invest time memorising the way the level flows or invest in micro transactions to assist them in getting further in the level.

The objective of this project is to develop an engaging and exciting game within the 'endless runner' genre that has strong replay value through the use of multiple random variables that the player experiences in each play through. The main proponent of this will be the constant randomly generated position of enemies that the player must avoid as well as the map that they traverse.

The game will be developed within the Unity Engine, which is known to be a very suitable engine for creating all types of games within the game industry due to its strong performance, interface and the use of the programming language C# as a scripting tool for writing instructions to objects. Unity provides and accepts a variety of assets, making it straightforward to apply visual changes to objects in the scene, and also having a large community due to being established for over 15 years. This may be of assistance through access to open source assets or when encountering challenges.



3. Objectives

1. First Iteration (Basic Functionality)

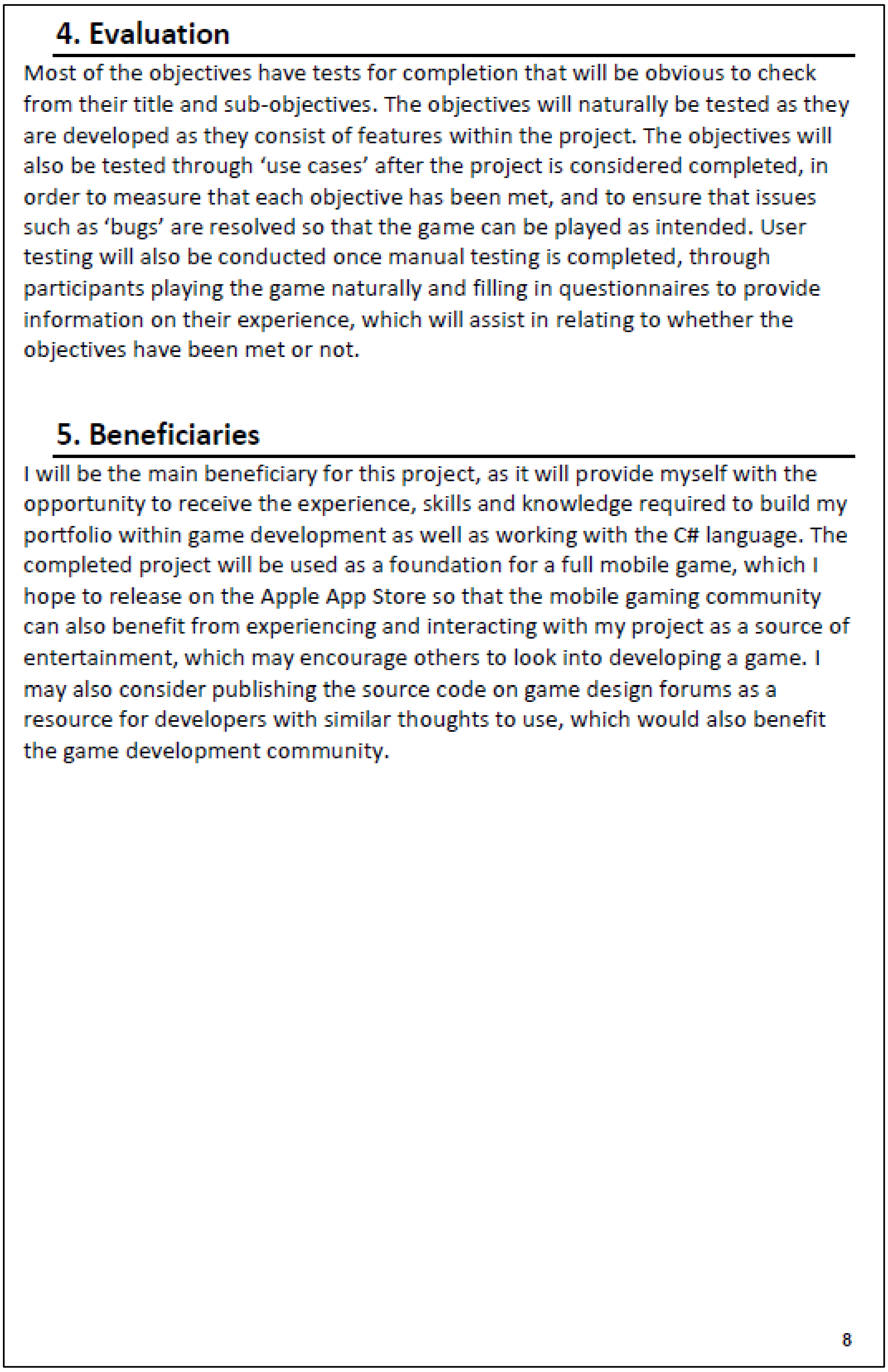
- **1.1. Map Design**
  - 1.1.1 Plane Generation
  - 1.1.2 Plane Prefab
  - 1.1.3 Collision Detection
- **1.2. Player Design**
  - 1.2.1 Axis Movement
  - 1.2.2 Character Prefab
  - 1.2.3 Third Person Camera (Follow)
- **1.3 Enemy Design**
  - 1.3.1 Enemy Prefab
  - 1.3.2 Collision Detection

2. Second Iteration (Main Functionality)

- **2.1 Map Layout**
  - 2.1.1 Random Plane Design
  - 2.1.2 Random Plane Generation
- **2.2 Player Animations**
  - 2.2.1 Pickup Used Animations
  - 2.2.2 Death Animation
- **2.3 Enemy Animations**
  - 2.3.1 Enemy Generation
  - 2.3.2 Enemy Destruction
  - 2.3.3 Enemy Movement
- **2.4 Pick-ups**
  - 2.4.1 Pickup Generation
  - 2.4.2 Pickup Animations
  - 2.4.3 Time Slowdown
  - 2.4.4 Timed Invincibility
  - 2.4.5 Points
- **2.5 Score**
  - 2.5.1 Distance Scoring
  - 2.5.2 Point Scoring
  - 2.5.2 High Score Table

3. Third Iteration (Final Enhancements)

- **3.1 Sound**
- **3.2 Lighting**
  - 3.2.1 Ambient Lighting
  - 3.2.2 Visual Fog
  - 3.2.3 Shaders
- **3.3 UI Implementation**
  - 3.3.1 Main Menu Design
  - 3.3.2 Retry
  - 3.3.3 Map Theme Change



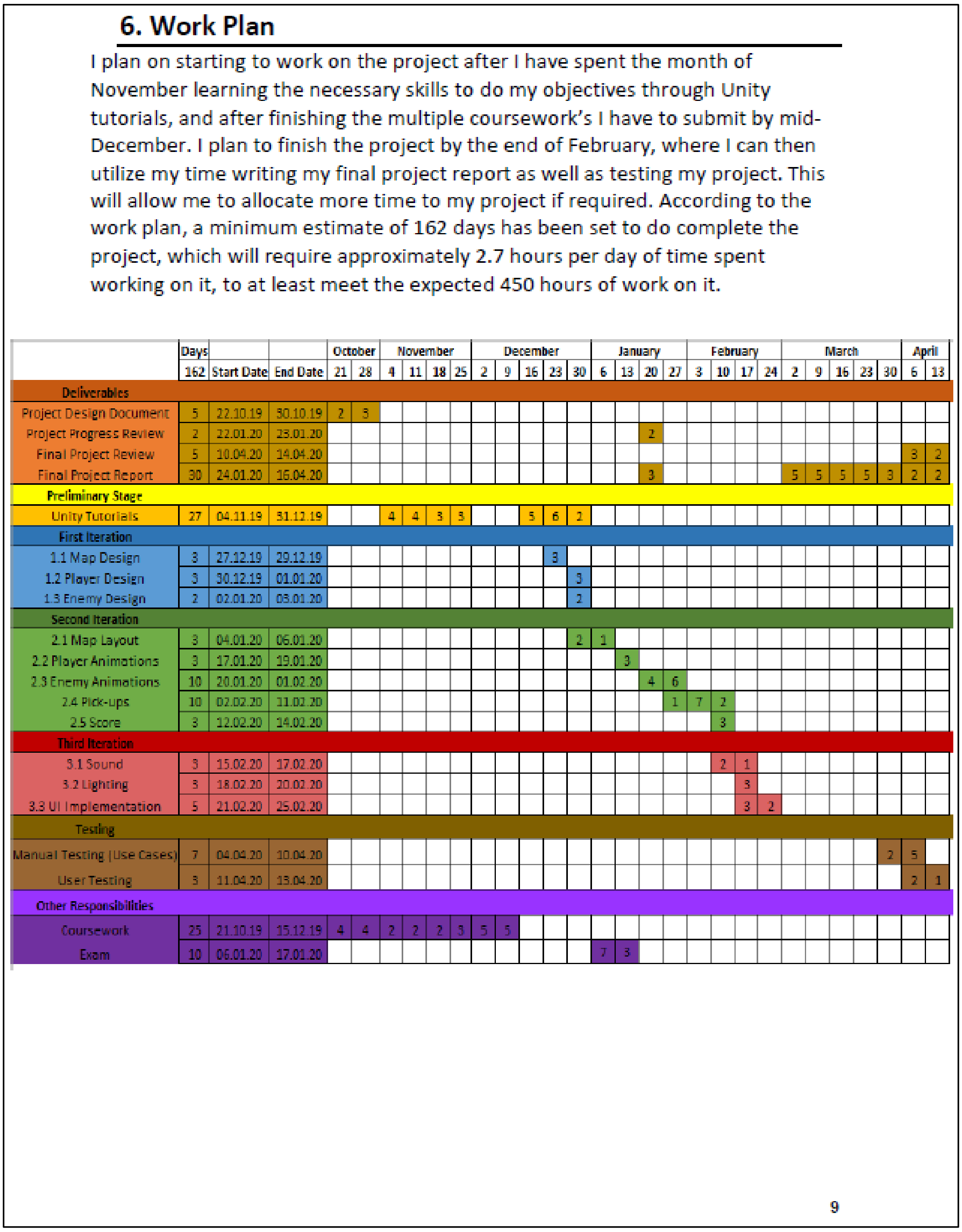
4. Evaluation

Most of the objectives have tests for completion that will be obvious to check from their title and sub-objectives. The objectives will naturally be tested as they are developed as they consist of features within the project. The objectives will also be tested through 'use cases' after the project is considered completed, in order to measure that each objective has been met, and to ensure that issues such as 'bugs' are resolved so that the game can be played as intended. User testing will also be conducted once manual testing is completed, through participants playing the game naturally and filling in questionnaires to provide information on their experience, which will assist in relating to whether the objectives have been met or not.

5. Beneficiaries

I will be the main beneficiary for this project, as it will provide myself with the opportunity to receive the experience, skills and knowledge required to build my portfolio within game development as well as working with the C# language. The completed project will be used as a foundation for a full mobile game, which I hope to release on the Apple App Store so that the mobile gaming community can also benefit from experiencing and interacting with my project as a source of entertainment, which may encourage others to look into developing a game. I may also consider publishing the source code on game design forums as a resource for developers with similar thoughts to use, which would also benefit the game development community.

8

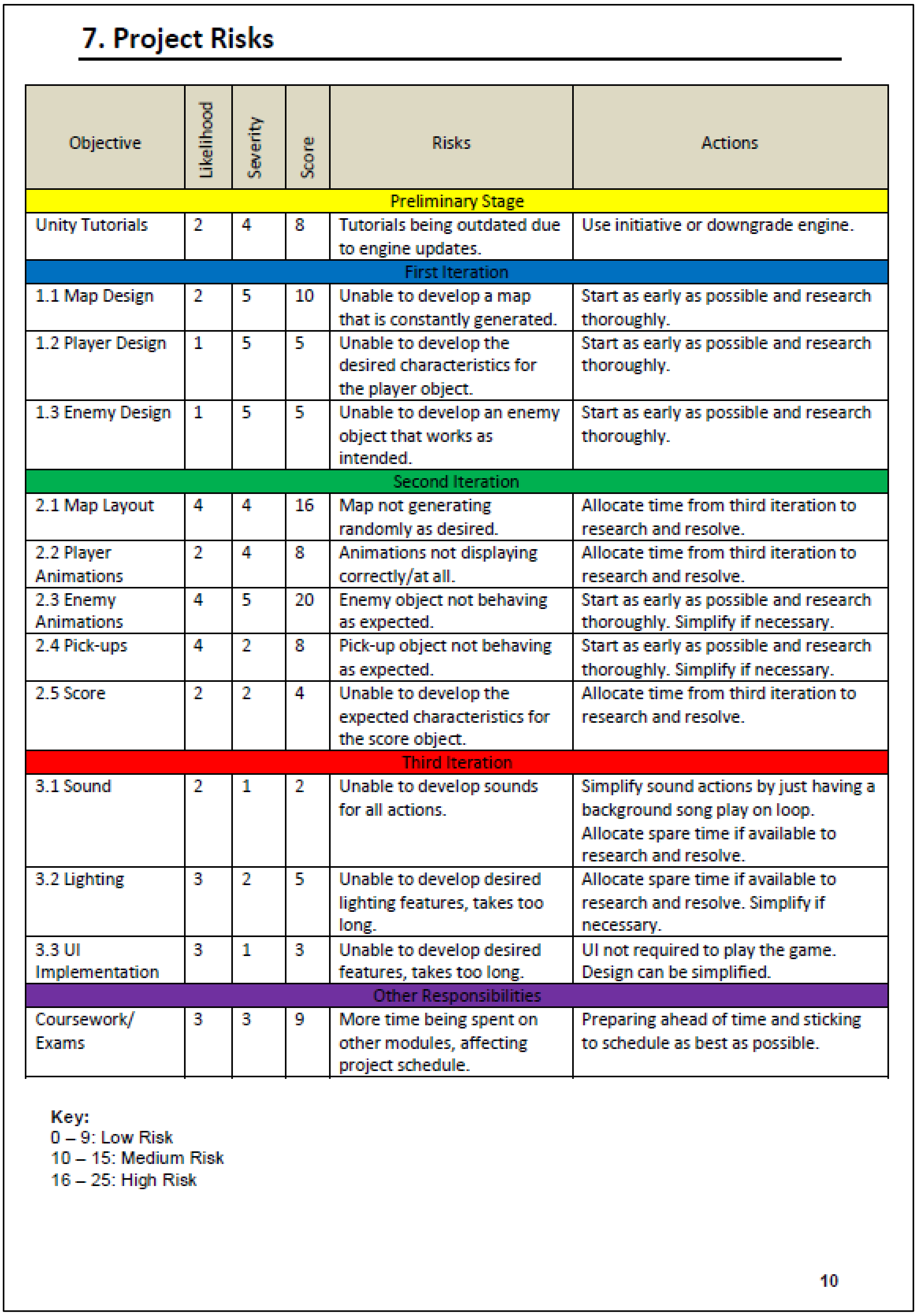


6. Work Plan

I plan on starting to work on the project after I have spent the month of November learning the necessary skills to do my objectives through Unity tutorials, and after finishing the multiple coursework's I have to submit by mid-December. I plan to finish the project by the end of February, where I can then utilize my time writing my final project report as well as testing my project. This will allow me to allocate more time to my project if required. According to the work plan, a minimum estimate of 162 days has been set to do complete the project, which will require approximately 2.7 hours per day of time spent working on it, to at least meet the expected 450 hours of work on it.

	Days			October		November				December					January				February				March					April	
	162	Start Date	End Date	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	3	10	17	24	2	9	16	23	30	6	13
Deliverables																													
Project Design Document	5	22.10.19	30.10.19	2	3																								
Project Progress Review	2	22.01.20	23.01.20														2												
Final Project Review	5	10.04.20	14.04.20																									3	2
Final Project Report	30	24.01.20	16.04.20														3						5	5	5	5	3	2	2
Preliminary Stage																													
Unity Tutorials	27	04.11.19	31.12.19			4	4	3	3			5	6	2															
First Iteration																													
1.1 Map Design	3	27.12.19	29.12.19										3																
1.2 Player Design	3	30.12.19	01.01.20											3															
1.3 Enemy Design	2	02.01.20	03.01.20											2															
Second Iteration																													
2.1 Map Layout	3	04.01.20	06.01.20											2	1														
2.2 Player Animations	3	17.01.20	19.01.20													3													
2.3 Enemy Animations	10	20.01.20	01.02.20														4	6											
2.4 Pick-ups	10	02.02.20	11.02.20															1	7	2									
2.5 Score	3	12.02.20	14.02.20																	3									
Third Iteration																													
3.1 Sound	3	15.02.20	17.02.20																	2	1								
3.2 Lighting	3	18.02.20	20.02.20																		3								
3.3 UI Implementation	5	21.02.20	25.02.20																		3	2							
Testing																													
Manual Testing (Use Cases)	7	04.04.20	10.04.20																								2	5	
User Testing	3	11.04.20	13.04.20																									2	1
Other Responsibilities																													
Coursework	25	21.10.19	15.12.19	4	4	2	2	2	3	5	5																		
Exam	10	06.01.20	17.01.20												7	3													

9



7. Project Risks

Objective	Likelihood	Severity	Score	Risks	Actions
Preliminary Stage					
Unity Tutorials	2	4	8	Tutorials being outdated due to engine updates.	Use initiative or downgrade engine.
First Iteration					
1.1 Map Design	2	5	10	Unable to develop a map that is constantly generated.	Start as early as possible and research thoroughly.
1.2 Player Design	1	5	5	Unable to develop the desired characteristics for the player object.	Start as early as possible and research thoroughly.
1.3 Enemy Design	1	5	5	Unable to develop an enemy object that works as intended.	Start as early as possible and research thoroughly.
Second Iteration					
2.1 Map Layout	4	4	16	Map not generating randomly as desired.	Allocate time from third iteration to research and resolve.
2.2 Player Animations	2	4	8	Animations not displaying correctly/at all.	Allocate time from third iteration to research and resolve.
2.3 Enemy Animations	4	5	20	Enemy object not behaving as expected.	Start as early as possible and research thoroughly. Simplify if necessary.
2.4 Pick-ups	4	2	8	Pick-up object not behaving as expected.	Start as early as possible and research thoroughly. Simplify if necessary.
2.5 Score	2	2	4	Unable to develop the expected characteristics for the score object.	Allocate time from third iteration to research and resolve.
Third Iteration					
3.1 Sound	2	1	2	Unable to develop sounds for all actions.	Simplify sound actions by just having a background song play on loop. Allocate spare time if available to research and resolve.
3.2 Lighting	3	2	5	Unable to develop desired lighting features, takes too long.	Allocate spare time if available to research and resolve. Simplify if necessary.
3.3 UI Implementation	3	1	3	Unable to develop desired features, takes too long.	UI not required to play the game. Design can be simplified.
Other Responsibilities					
Coursework/ Exams	3	3	9	More time being spent on other modules, affecting project schedule.	Preparing ahead of time and sticking to schedule as best as possible.

Key:
0 – 9: Low Risk
10 – 15: Medium Risk
16 – 25: High Risk

8. References

[1] Ben Chong. (2015). Endless Runner Games: How To Think And Design (Plus Some History) Available at: https://www.gamasutra.com/blogs/BenChong/20150112/233958/Endless_Runner_Games_How_to_think_and_design_plus_some_history.php [Accessed 23 Oct. 2019].

[2] Limelight Networks. (2019, p. 3). The State Of Online Gaming – 2019 (Study) Available At: https://www.limelight.com/resources/white-paper/state-of-online-gaming-2019/

[3] Matt Peckham. (2014). Temple Run Scampers Past One Billion Downloads Available At: https://time.com/2822645/temple-run-one-billion-downloads/ [Accessed 23 Oct. 2019].

[4] David Cao. (2016, p. 4). Game Design Patterns In Endless Mobile Minigames (Study) Available At: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwiXmeKSzsflAhX1SxUIHZurAlgQFjAAegQIBxAC&url=https%3A%2F%2Fpdfs.semanticscholar.org%2Ffdee%2F592cea99cbb1c65d737771a092aa5cf25c4d.pdf&usg=AOvVaw3nr9FMofFtW2mLpwhi7FVk [Accessed 28 Oct. 2019]

[5] Wikipedia. (2012, Revised 2019). Temple Run [image] Available At: https://en.wikipedia.org/wiki/Temple_Run#/media/File:Temple_Run_gameplay.png [Accessed 28 Oct. 2019]

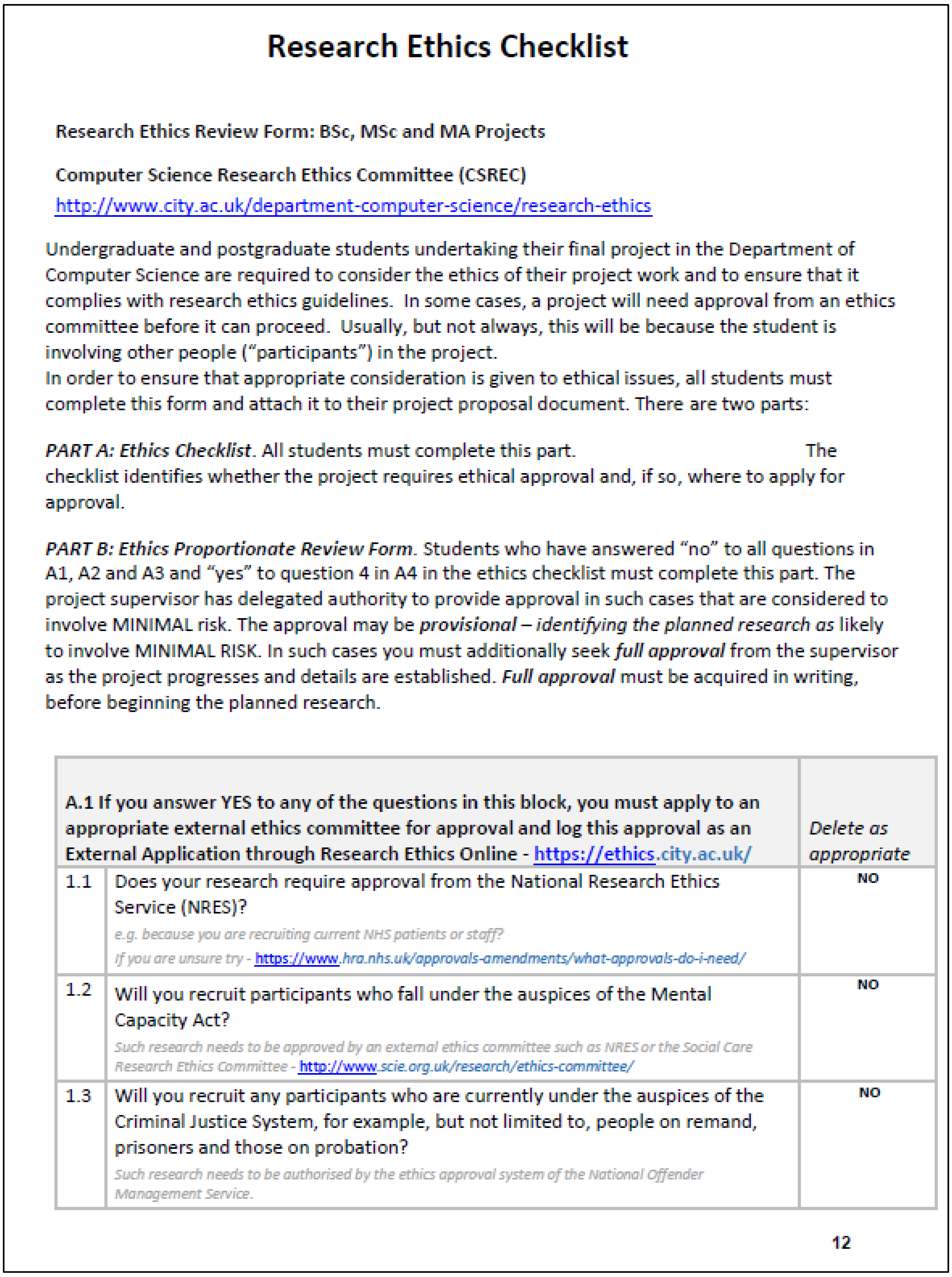
[6] Tenor. (2017). Subway Surfers [image] Available At: https://tenor.com/view/subway-surfers-gif-9012698 [Accessed 28 Oct. 2019]

[7] Common Sense Media. (no date) Jetpack Joyride [image] Available At: https://www.commonsensemedia.org/review-gallery/modal/2150181?width=660&height=415&iframe=1&slide=0 [Accessed 28 Oct. 2019]

[8] Google Trends. (2019). Temple Run Interest Over Time [image] Available At: https://trends.google.com/trends/explore?date=2011-08-01%202019-10-23&q=temple%20run [Accessed 23 Oct. 2019].

[9] Google Trends. (2019). Jetpack Joyride Interest Over Time [image] Available At: https://trends.google.com/trends/explore?date=2011-08-01%202019-10-23&q=jetpack%20joyride [Accessed 23 Oct. 2019].

[10] Google Trends. (2019). Subway Surfers Interest Over Time [image] Available At: https://trends.google.com/trends/explore?date=2011-08-01%202019-10-23&q=subway%20surfers [Accessed 23 Oct. 2019].



Research Ethics Checklist

Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)
http://www.city.ac.uk/department-computer-science/research-ethics

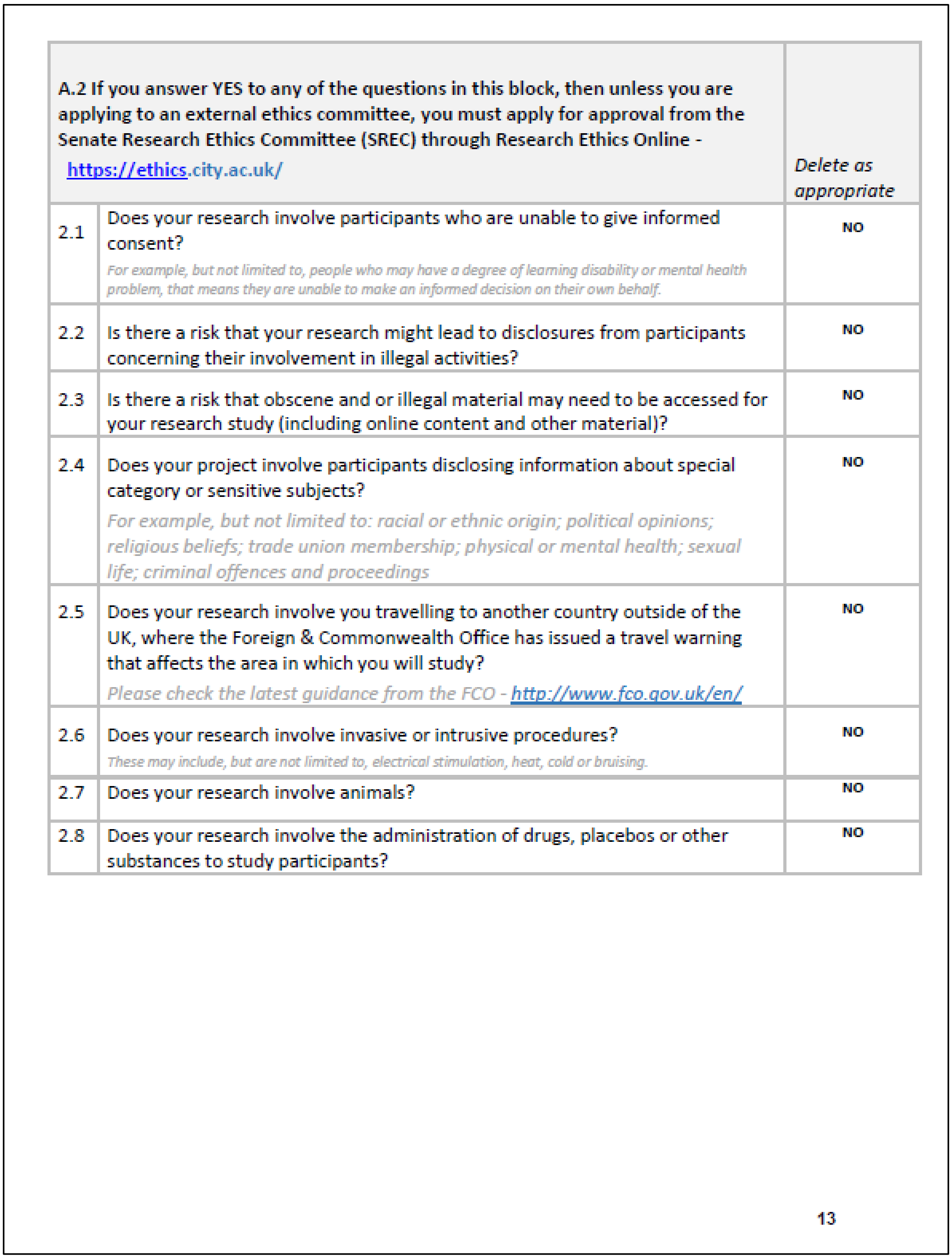
Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.
In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be *provisional – identifying the planned research as* likely to involve MINIMAL RISK. In such cases you must additionally seek *full approval* from the supervisor as the project progresses and details are established. *Full approval* must be acquired in writing, before beginning the planned research.

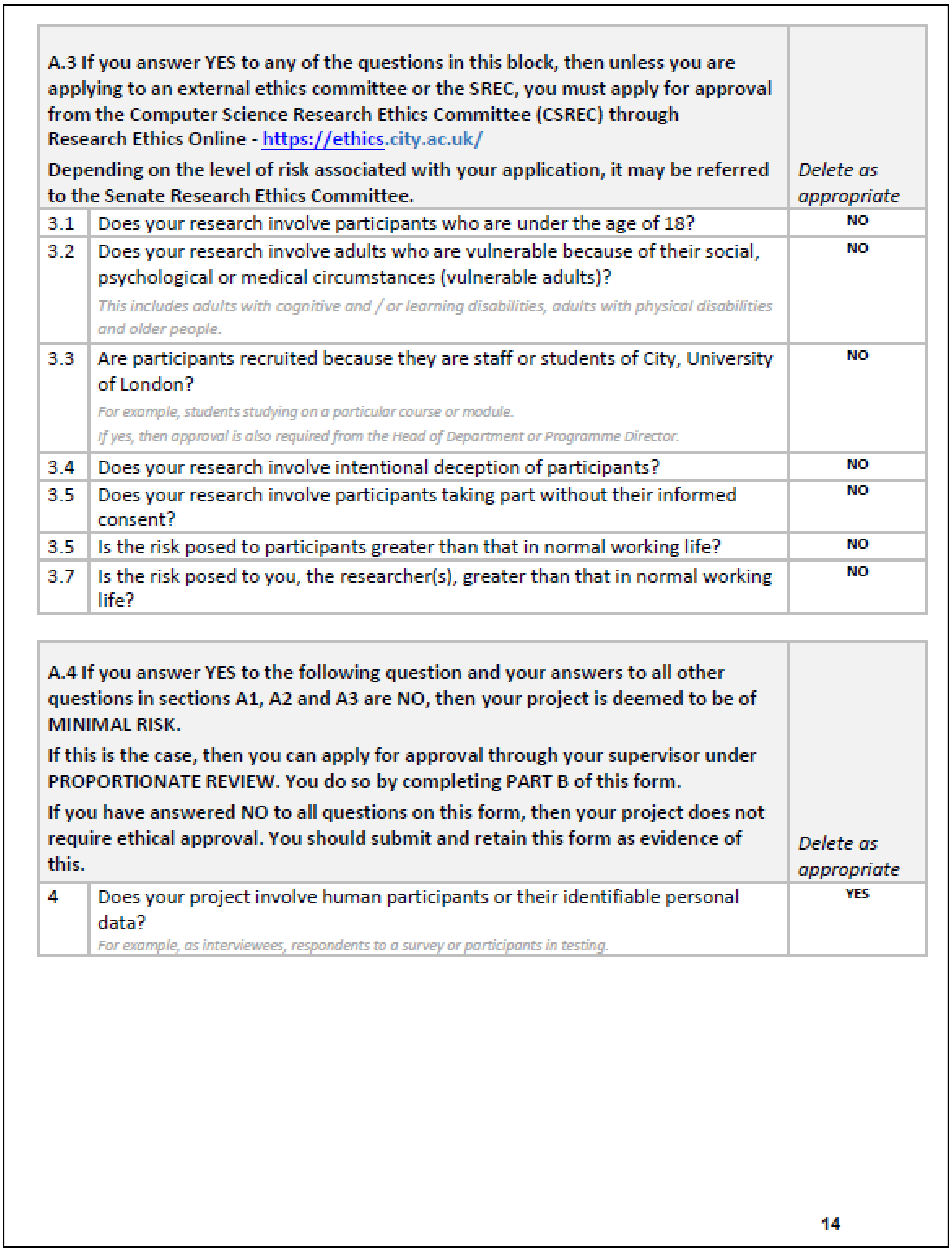
A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**

12



A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO -* http://www.fco.gov.uk/en/	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**

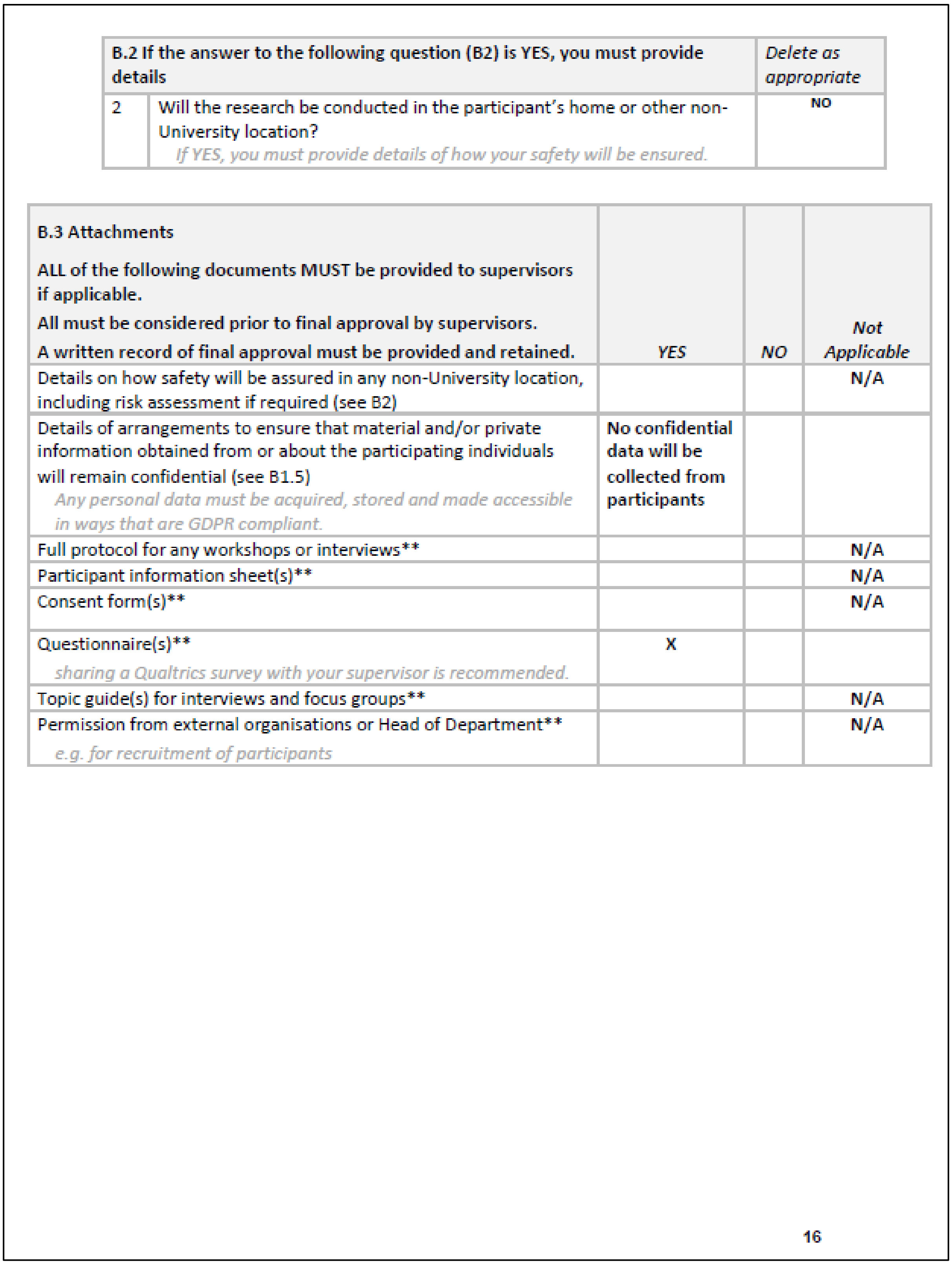
13



A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	NO
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	NO
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	NO
3.4	Does your research involve intentional deception of participants?	NO
3.5	Does your research involve participants taking part without their informed consent?	NO
3.5	Is the risk posed to participants greater than that in normal working life?	NO
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	NO

A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	YES

14



B.2 If the answer to the following question (B2) is YES, you must provide details		*Delete as appropriate*
2	Will the research be conducted in the participant's home or other non-University location? *If YES, you must provide details of how your safety will be ensured.*	**NO**

B.3 Attachments **ALL of the following documents MUST be provided to supervisors if applicable.** **All must be considered prior to final approval by supervisors.** **A written record of final approval must be provided and retained.**	*YES*	*NO*	*Not Applicable*
Details on how safety will be assured in any non-University location, including risk assessment if required (see B2)			**N/A**
Details of arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential (see B1.5) *Any personal data must be acquired, stored and made accessible in ways that are GDPR compliant.*	**No confidential data will be collected from participants**		
Full protocol for any workshops or interviews**			**N/A**
Participant information sheet(s)**			**N/A**
Consent form(s)**			**N/A**
Questionnaire(s)** *sharing a Qualtrics survey with your supervisor is recommended.*	**X**		
Topic guide(s) for interviews and focus groups**			**N/A**
Permission from external organisations or Head of Department** *e.g. for recruitment of participants*			**N/A**

16

***If these items are not available at the time of submitting your project proposal, then provisional approval can still be given, under the condition that you must submit the final versions of all items to your supervisor for approval at a later date. All such items must be seen and approved by your supervisor before the activity for which they are needed begins. Written evidence of final approval of your planned activity must be acquired from your supervisor before you commence.*

Changes
If your plans change and any aspects of your research that are documented in the approval process change as a consequence, then any approval acquired is invalid. If issues addressed in Part A (the checklist) are affected, then you must complete the approval process again and establish the kind of approval that is required. If issues addressed in Part B are affected, then you must forward updated documentation to your supervisor and have received written confirmation of approval of the revised activity before proceeding.

Templates for Consent and Information
You must use the templates provided by the University as the basis for your participant information sheets and consent forms. You **must** adapt them according to the needs of your project before you submit them for consideration.
Participant Information Sheets, Consent Forms and Protocols must be consistent. Please ensure that this is the case prior to seeking approval. Failure to do so will slow down the approval process.
We strongly recommend using Qualtrics to produce digital information sheets and consent forms.

Further Information

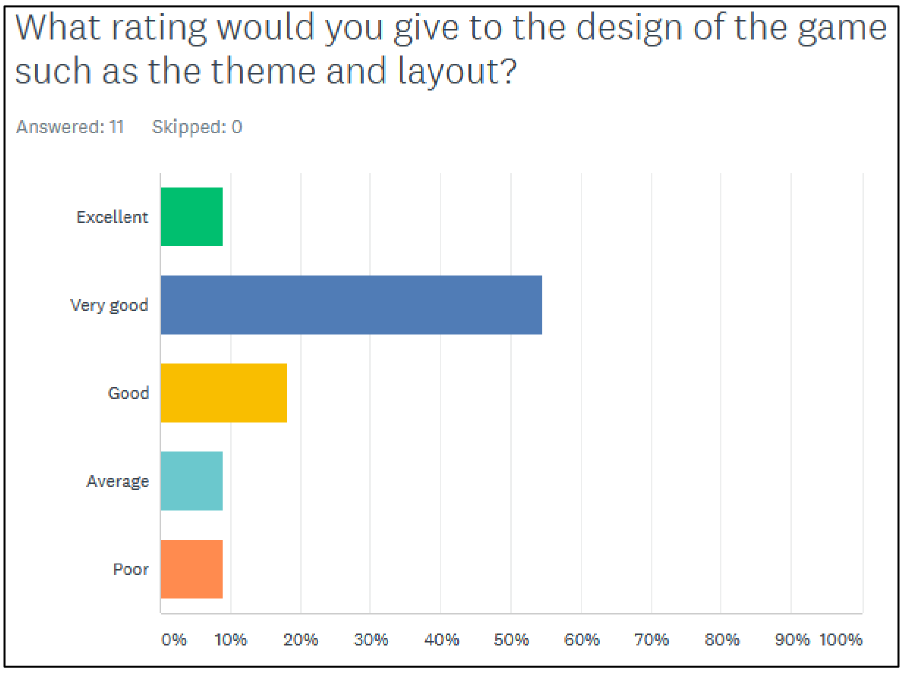
http://www.city.ac.uk/department-computer-science/research-ethics

https://www.city.ac.uk/research/ethics/how-to-apply/participant-recruitment

https://www.city.ac.uk/research/ethics

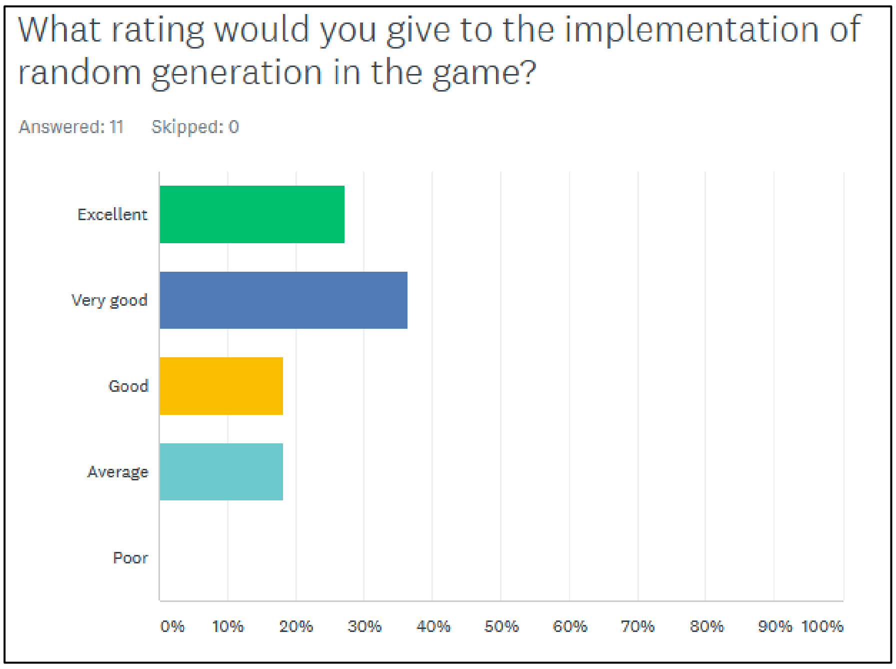
Appendix B - SurveyMonkey

Q1


What rating would you give to the design of the game such as the theme and layout?

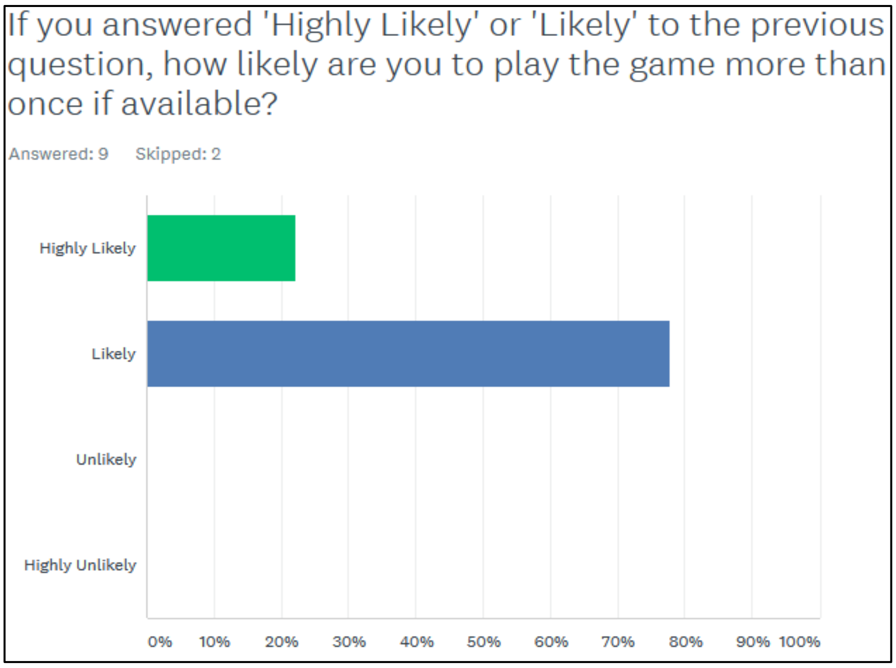
Answered: 11 Skipped: 0

Q2


What rating would you give to the implementation of random generation in the game?

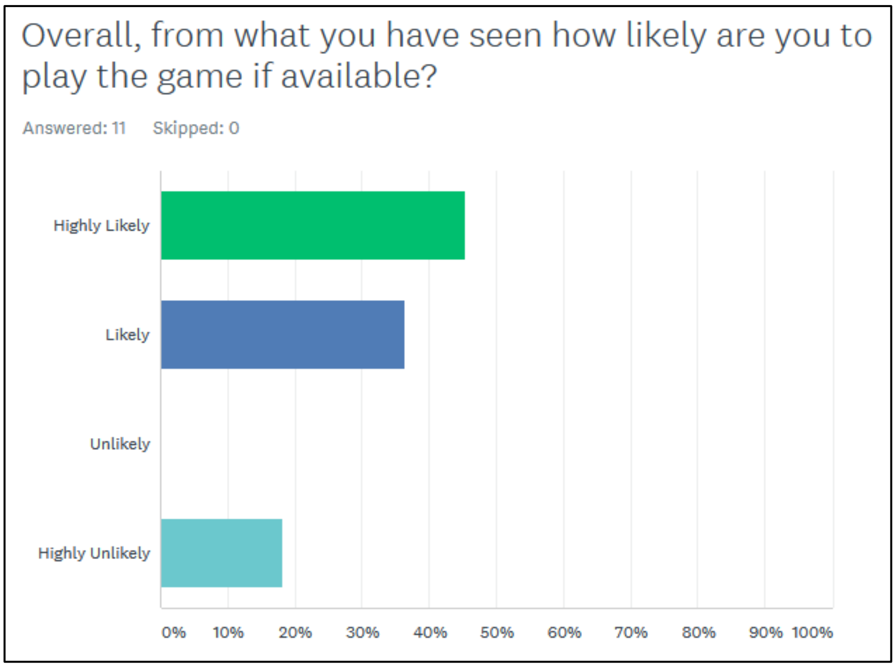
Answered: 11 Skipped: 0

Q3


If you answered 'Highly Likely' or 'Likely' to the previous question, how likely are you to play the game more than once if available?

Answered: 9 Skipped: 2

Q4


Overall, from what you have seen how likely are you to play the game if available?

Answered: 11 Skipped: 0

Appendix C – UML Diagrams

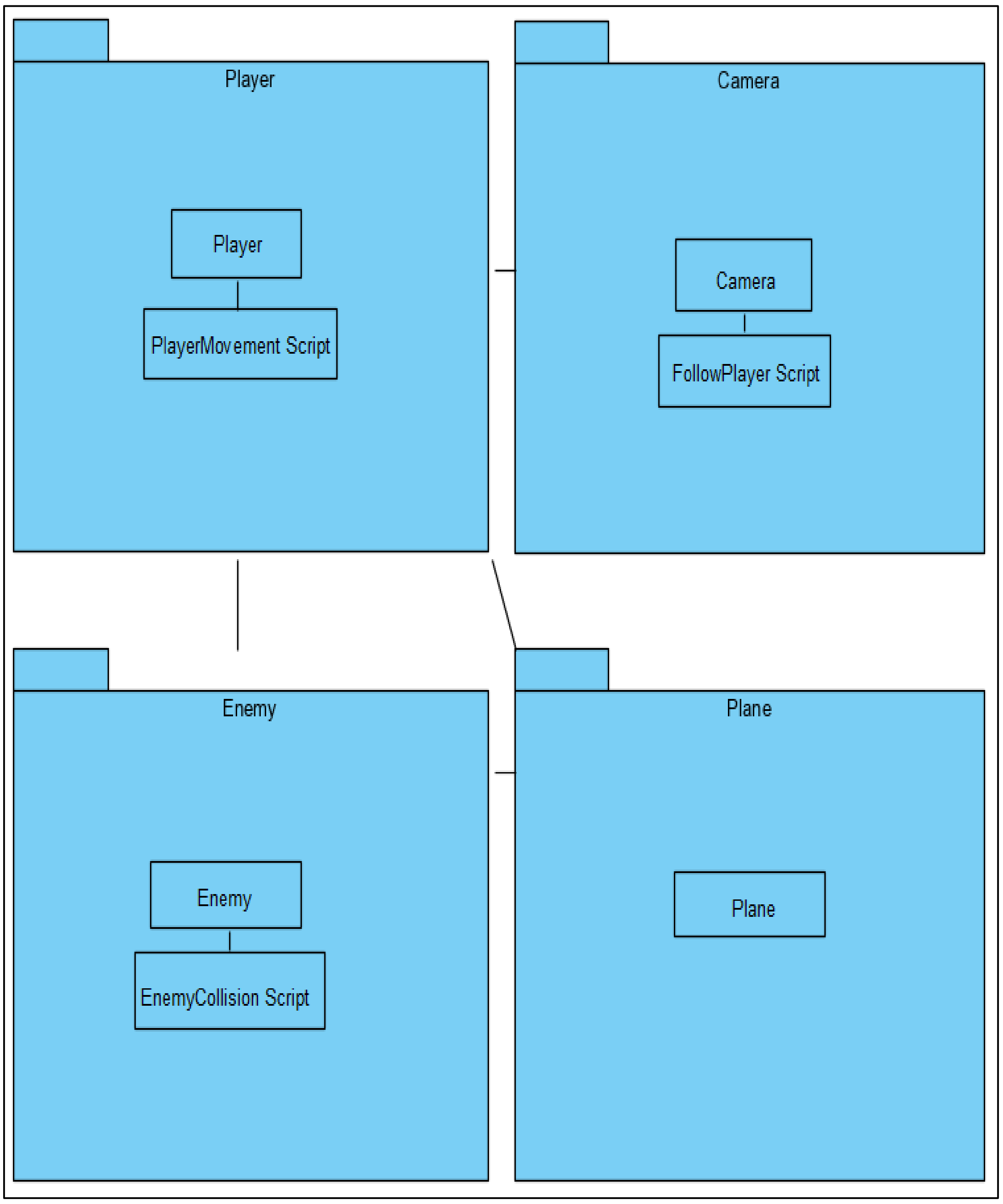


Figure 1: Planned implementation of features for the first iteration showing a brief overview of the features and scripts split into relevant packages and the interaction that they have with other features

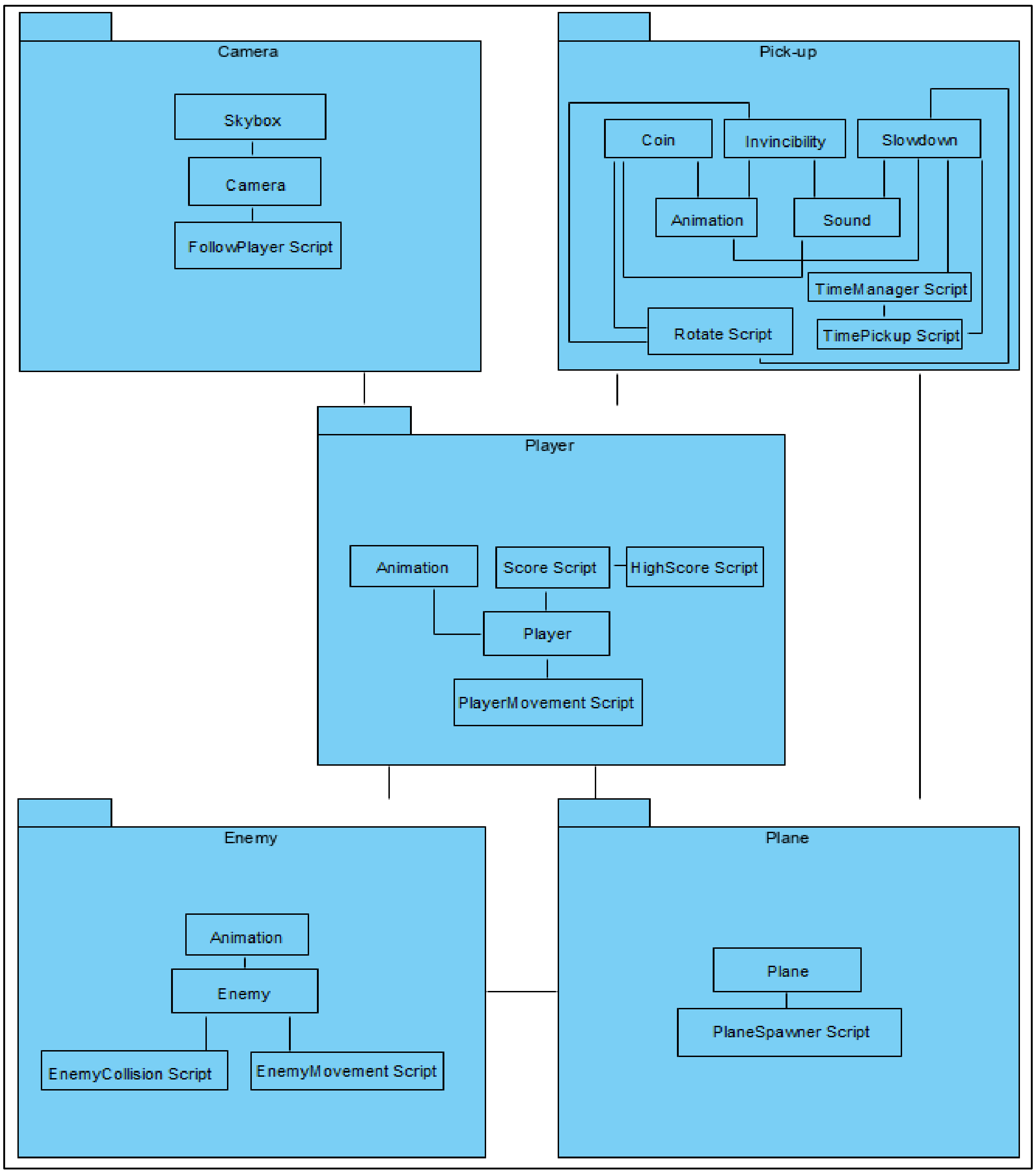


Figure 2: Planned implementation of features for the second iteration showing a brief overview of the features and scripts split into relevant packages and the interaction that they have with other features

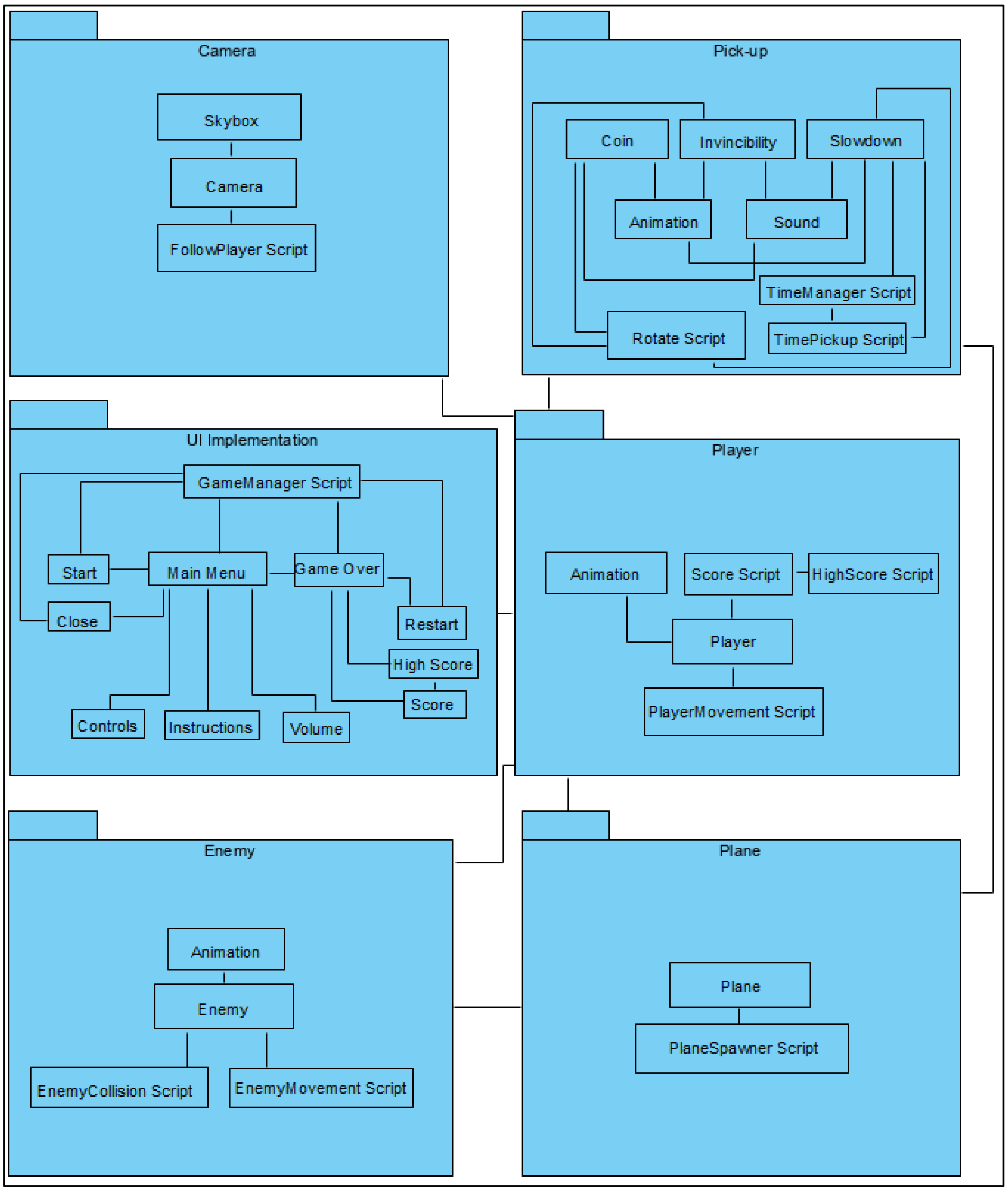


Figure 3: Planned implementation of features for the third iteration showing a brief overview of the features and scripts split into relevant packages and the interaction that they have with other features

Appendix D – Assets

Spaceship

The model chosen for the Player from the ‘StarSparrow’ asset.

Skybox

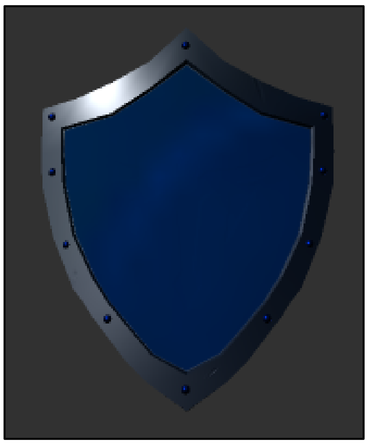
The image chosen for the front of the Skybox from the ‘SpaceSkies Free’ asset.

Sand Hourglass

The model chosen for the Time Slowdown pickup from the ‘Sand Clock’ asset.

Shield

The initial model chosen for the Shield Pickup from the ‘Shield’ asset.



Edited Shield

The edited model used to match the theme of the game.

EnergyCell_1_2

Energy Cell

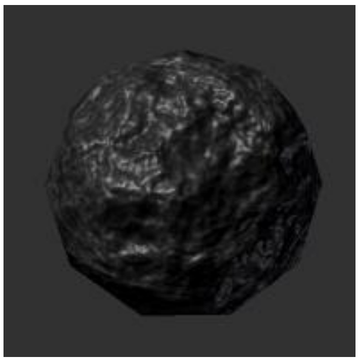
The model chosen for the Coin pickup from the ‘HiTech SciFi Energy Cell’ asset.

Asteroids

From the ‘Asteroids low-poly pack’ asset.

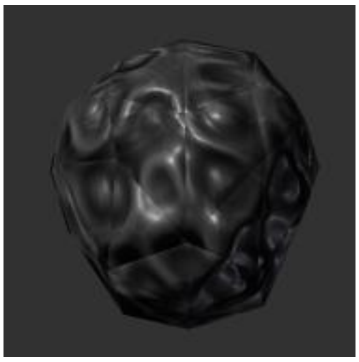
Still Asteroid

The model chosen for the stationery Enemy object.



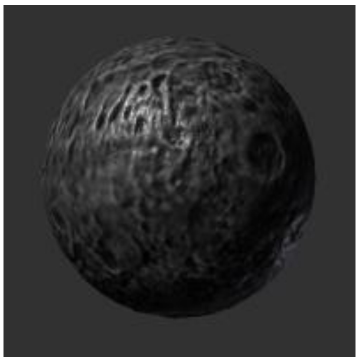
X Moving Asteroid

The model chosen for the Enemy object moving on the x-axis.



Z Moving Asteroid

The model chosen for the Enemy object moving on the y-axis.

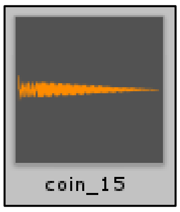


Chasing Asteroid

The model chosen for the Enemy object following the Player.

Sound Effects

From the ‘Retro Rare Sound Effects’ asset.



Coin SFX

The chosen audio source played when a Coin is collected by the Player.

Hit SFX

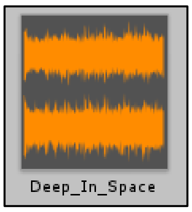
The chosen audio source played when the Player is destroyed by an Asteroid.

Time Slowdown SFX

The chosen audio source played when a Time Slowdown pickup is collected by the Player.

Invincibility SFX

The chosen audio source played when a Invincibility pickup is collected by the Player.

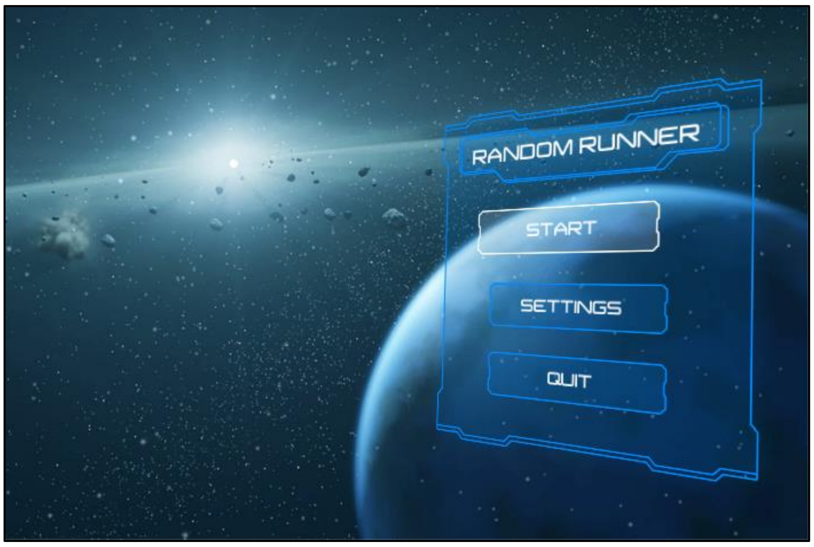


Music

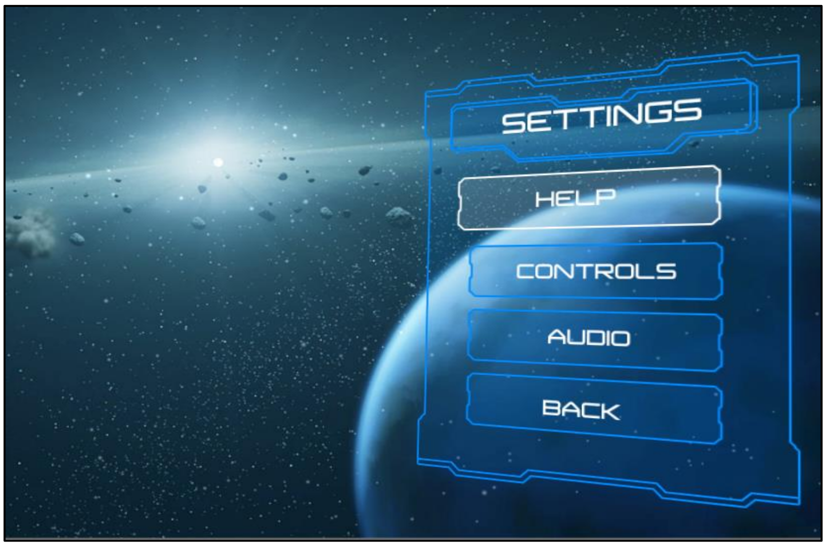
The chosen audio source played on loop as background music from the 'Deep In Space' asset

UI Screens

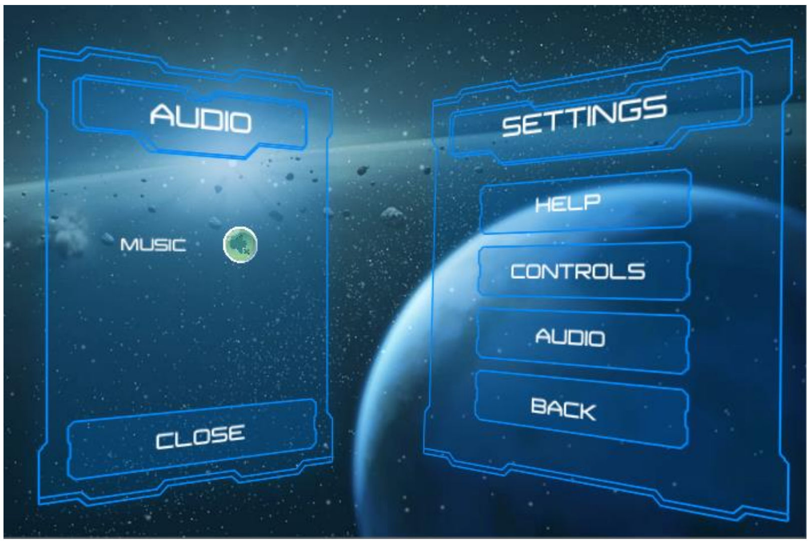
From the 'Unity Samples: UI' asset.



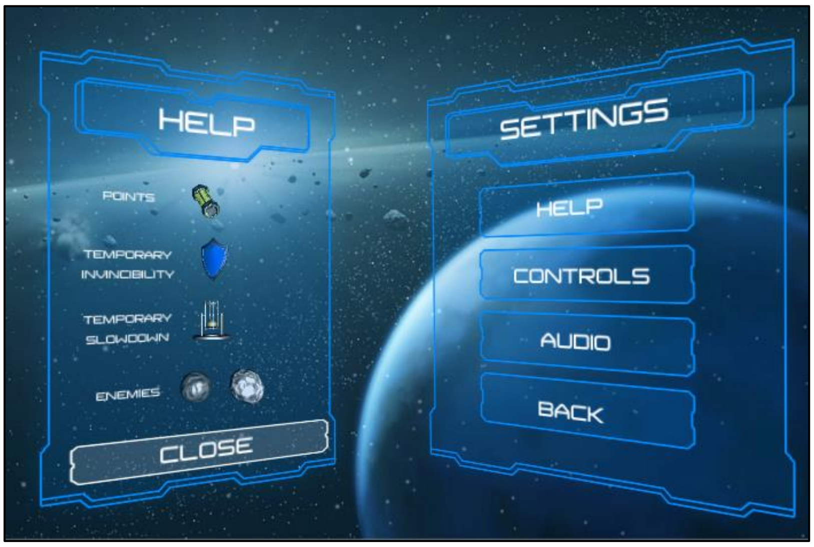
Main Menu Screen



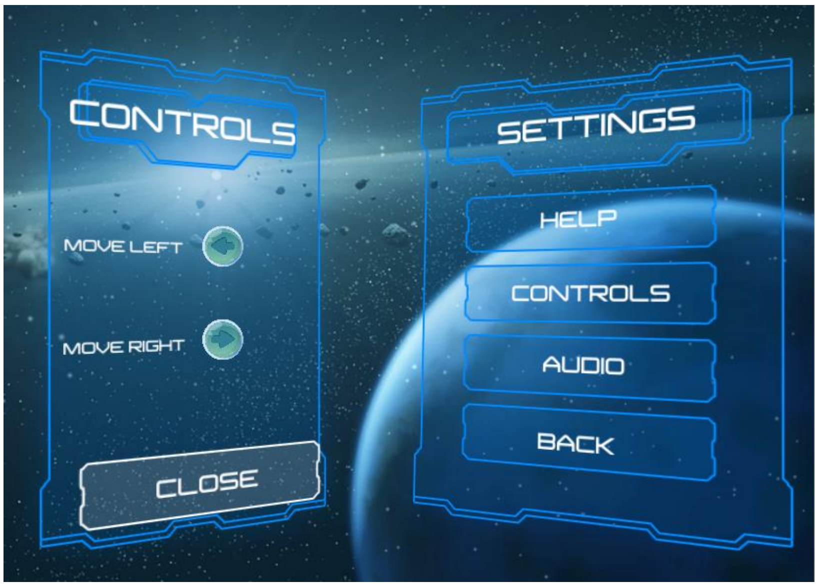
Settings Screen



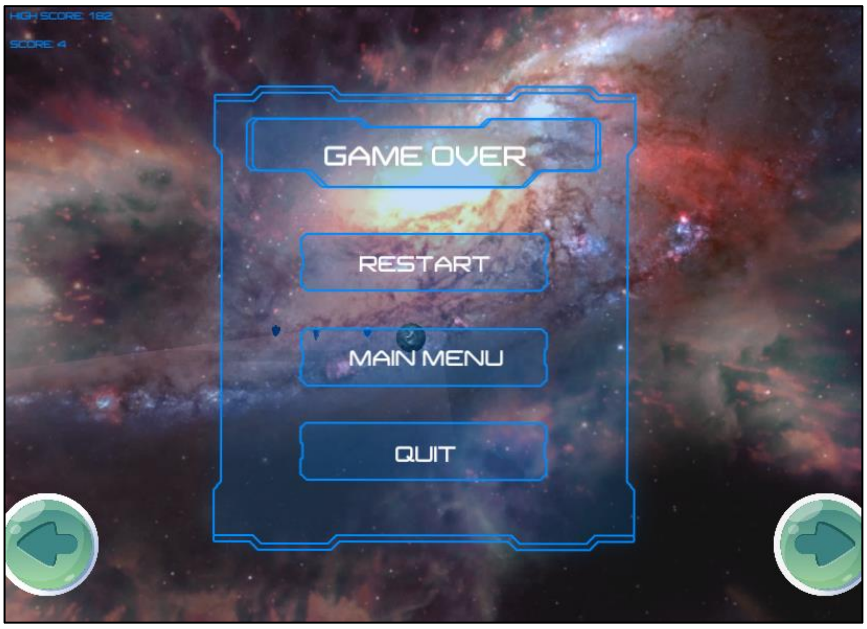
Audio Panel



Help Panel



Controls Panel



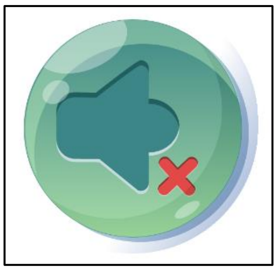
Restart Screen

UI Buttons

From the 'Icons UI' asset.

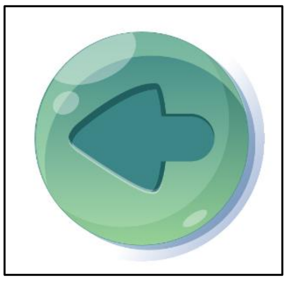
Music Toggle

The chosen music toggle button.



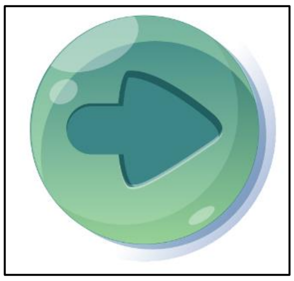
Music Toggle Indicator

The chosen music toggle indicator button.



Move Left

The chosen left movement button.



Move Right

The chosen right movement button.

References

[1] Apple. (2011). “Temple Run”. Available At: https://apps.apple.com/app/apple-store/id420009108 [Accessed 21 Oct. 2019].

[2] Time. (2014). “Temple Run Scampers Past One Billion Downloads”. Available At: https://time.com/2822645/temple-run-one-billion-downloads/ [Accessed 21 Oct. 2019].

[3] Unity. (2020). “Unity for all”. Available At: https://unity.com/ [Accessed 04 Nov. 2019].

[4] Creative Bloq Staff. (2019). “Unity vs Unreal Engine: which game engine is for you?”. Available at: https://www.creativebloq.com/advice/unity-vs-unreal-engine-which-game-engine-is-for-you [Accessed 21 Oct. 2019].

[5] Unity. (2020). “Scripting in Unity for experienced programmers”. Available At: https://unity.com/how-to/programming-unity [Accessed 21 Oct. 2019].

[6] Unity. (2020). “Creating and Using Scripts”. Available At: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html?_ga=2.77374741.1864982482.1582144970-640506962.1534187723 [Accessed 21 Oct. 2019].

[7] David Cao. (2016, p. 4). Game Design Patterns In Endless Mobile Minigames (Study) Available At: https://pdfs.semanticscholar.org/fdee/592cea99cbb1c65d737771a092aa5cf25c4d.pdf [Accessed 28 Oct. 2019]

[8] Unity. (2017). “MonoDevelop”. Available At: https://docs.unity3d.com/560/Documentation/Manual/MonoDevelop.html [Accessed 27 Dec. 2019]

[9] SurveyMonkey. (2020). “SurveyMonkey”. Available At: https://www.surveymonkey.com/ [Accessed 26 Apr 2020.].

[10] Brackeys. (2017). “How to make a Video Game”. Available At: https://www.youtube.com/playlist?list=PLPV2Kylb3jR53Jce9hP7G5xC4O9AgnOuL [Accessed 27 Dec. 2019].

[11] Unity. (2020). “Transform”. Available At: https://docs.unity3d.com/ScriptReference/Transform.html [Accessed 27 Dec. 2019].

[12] Unity. (2020). “Mesh Filter”. Available At: https://docs.unity3d.com/ScriptReference/MeshFilter.html [Accessed 27 Dec. 2019].

[13] Unity. (2020). “Mesh Renderer”. Available At: https://docs.unity3d.com/ScriptReference/MeshRenderer.html [Accessed 27 Dec. 2019].

[14] Unity. (2020). “Box Collider”. Available At: https://docs.unity3d.com/ScriptReference/Transform.html [Accessed 27 Dec. 2019].

[15] Unity. (2020). “Rigidbody”. Available At: https://docs.unity3d.com/ScriptReference/Rigidbody.html [Accessed 29 Dec. 2019].

[16] Unity. (2020). “Rigidbody.freezeRotation”. Available At: https://docs.unity3d.com/ScriptReference/Rigidbody-freezeRotation.html [Accessed 29 Dec. 2019].

[17] Unity. (2020). “Monobehaviour.FixedUpdate”. Available At: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html [Accessed 29 Dec. 2019].

[18] Unity. (2020). “Time.deltaTime”. Available At: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html [Accessed 29 Dec. 2019].

[19] Unity. (2020). “Object.FindObjectOfType”. Available At: https://docs.unity3d.com/ScriptReference/Object.FindObjectOfType.html [Accessed 30 Dec. 2019].

[20] Unity. (2020). “SceneManager.LoadScene”. Available At: https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html [Accessed 30 Dec. 2019].

[21] Unity. (2020). “Monobehaviour.Invoke”. Available At: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Invoke.html [Accessed 30 Dec. 2019].

[22] Unity. (2020). “The Inspector window”. Available At: https://docs.unity3d.com/Manual/UsingTheInspector.html [Accessed 30 Dec. 2019].

[23] Unity. (2020). “GameObject.Tag”. Available At: https://docs.unity3d.com/ScriptReference/GameObject-tag.html [Accessed 31 Dec. 2019].

[24] Unity. (2020). “Object.Destroy”. Available At: https://docs.unity3d.com/ScriptReference/Object.Destroy.html [Accessed 31 Dec. 2019].

[25] Unity. (2020). “Object.Instantiate”. Available At: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html [Accessed 31. 2019].

[26] Unity. (2020). “Monobehaviour.InvokeRepeating”. Available At: https://docs.unity3d.com/ScriptReference/MonoBehaviour.InvokeRepeating.html [Accessed 31 Dec. 2019].

[27] Rosscoe Tutorials. (2016). “Unity Tutorial | Spawning Random Enemies at Random Times and Positions”. Available At: https://www.youtube.com/watch?v=WGn1zvLSndk&t=720s [Accessed 1 Jan. 2020].

[28] Unity. (2020). “Coroutines”. Available At: https://docs.unity3d.com/Manual/Coroutines.html [Accessed 1 Jan. 2020].

[29] Unity. (2020). “Monobehaviour.StartCoroutine”. Available At: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html [Accessed 1 Jan. 2020].

[30] Unity. (2020). “Random.Range”. Available At: https://docs.unity3d.com/ScriptReference/Random.Range.html [Accessed 1 Jan. 2020].

[31] Unity. (2020). “WaitForSeconds”. Available At: https://docs.unity3d.com/ScriptReference/WaitForSeconds.html [Accessed 1 Jan. 2020].

[32] gamesplusjames. (2015). “Score & High Score System”. Available At: https://www.youtube.com/watch?v=9HvTwtfBaYM [Accessed 18 Jan. 2020]

[33] Unity. (2020). “Canvas”. Available At: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html [Accessed 18 Jan. 2020].

[34] Unity. (2020). “PlayerPrefs”. Available At: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html [Accessed 18 Jan. 2020].

[35] Ebal Studios. (2019). “Star Sparrow Modular Spaceship”. Available At: https://assetstore.unity.com/packages/3d/vehicles/space/star-sparrow-modular-spaceship-73167 [Accessed 1 Feb. 2020].

[36] VK GameDev. (2019). “Asteroids low-poly pack”. Available At: https://assetstore.unity.com/packages/3d/environments/sci-fi/asteroids-low-poly-pack-142164 [Accessed 1 Feb. 2020].

[37] AntiDed GameDev. (2019). “HiTech SciFi Energy Cell”. Available At: https://assetstore.unity.com/packages/3d/environments/sci-fi/hitech-scifi-energy-cell-154526 [Accessed 1 Feb. 2020].

[38] 3d.rina. (2019). “Sand Clock”. Available At: https://assetstore.unity.com/packages/3d/props/interior/sand-clock-8466 [Accessed 1 Feb. 2020].

[39] Zakhan. (2016). “Shield”. Available At: https://assetstore.unity.com/packages/3d/props/weapons/shield-61351 [Accessed 1 Feb. 2020].

[40] Brackeys. (2017). “How to make Slow Motion In Unity”. Available At: https://www.youtube.com/watch?v=0VGosgaoTsw [Accessed 5 Feb. 2020].

[41] Unity. (2020). “Time.timeScale”. Available At: https://docs.unity3d.com/ScriptReference/Time-timeScale.html [Accessed 5 Feb. 2020].

[42] Unity. (2020). “Time.unscaledDeltaTime”. Available At: https://docs.unity3d.com/ScriptReference/Time-unscaledDeltaTime.html [Accessed 5 Feb. 2020].

[43] Unity. (2020). “Time.fixedDeltaTime”. Available At: https://docs.unity3d.com/ScriptReference/Time-fixedDeltaTime.html [Accessed 5 Feb. 2020].

[44] Unity. (2020). “Mathf.Clamp”. Available At: https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html [Accessed 5 Feb. 2020].

[45] Unity. (2013). “Destroy bullet after ‘lifetime’ ”, Available At: https://answers.unity.com/questions/385030/destroy-bullet-after-lifetime-need-help-please-c.html [Accessed 10 Feb. 2020].

[46] Unity. (2019). “Unity Samples: UI”. Available At: https://assetstore.unity.com/packages/essentials/unity-samples-ui-25468 [Accessed 25 Feb 2020].

[47] Unity. (2020). “Button”. Available At:
https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Button.html
[Accessed 26 Feb. 2020]

[48] Kvazi. (2018). “Icons UI”. Available At:
https://assetstore.unity.com/packages/2d/gui/icons/icons-ui-95116 [Accessed 26 Feb 2020].

[49] Unity. (2020). “Image”. Available At:
https://docs.unity3d.com/2018.3/Documentation/ScriptReference/UI.Image.html
[Accessed 26 Feb. 2020].

[50] Unity. (2020). “Toggle”. Available At:
https://docs.unity3d.com/2017.3/Documentation/ScriptReference/UI.Toggle.html
[Accessed 27 Feb. 2020].

[51] Unity. (2020). “AudioSource.mute”. Available At:
https://docs.unity3d.com/ScriptReference/AudioSource-mute.html [Accessed 27 Feb. 2020].

[52] Unity. (2020). “EventTrigger”. Available At:
https://docs.unity3d.com/2018.3/Documentation/ScriptReference/EventSystems.EventTrigger.html
[Accessed 1 Mar. 2020].

[53] Awesome Tuts. (2018). “Unity Touch Controls Tutorial”. Available At:
https://www.youtube.com/watch?v=S8QNZ7oWXqw [Accessed 1 Mar. 2020].

[54] Breitbarth. (2017). “Deep In Space”. Available At:
https://assetstore.unity.com/packages/audio/music/electronic/deep-in-space-88071 [Accessed 15 Mar. 2020]

[55] Brodsky. (2001). “The effects of music tempo on simulated driving performance and vehicular control”. (Study). [Accessed 15 Mar. 2020]

[56] Unity. (2020). “AudioSource”. Available At:
https://docs.unity3d.com/ScriptReference/AudioSource.html [Accessed 15 Mar. 2020].

[57] Unity. (2020). “Object.DontDestroyOnLoad”. Available At:
https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html
[Accessed 18 Mar. 2020].

[58] Unity. (2020). “Unity C# Singleton?”. Available At:
https://answers.unity.com/questions/891380/unity-c-singleton.html [Accessed 18 Mar. 2020].

[59] Zero Rare. (2018). “Sound FX - Retro Pack”. Available At:
https://assetstore.unity.com/packages/audio/sound-fx/sound-fx-retro-pack-121743#description
[Accessed 25 Mar. 2020].

[60] Unity. (2020). “AudioSource.Play”. Available At:
https://docs.unity3d.com/ScriptReference/AudioSource.Play.html [Accessed 25 Mar. 2020].

[61] Imangi Studios. (2011). Temple Run [image]. Available At:
https://www.imangistudios.com/games.html [Accessed 2 Apr. 2020].

[62] Unity. (2020). “Vector3.Lerp”. Available At:
https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html [Accessed 2 Apr. 2020].

[63] Unity. (2020). “How do I rotate an object towards a Vector3 point?”. Available At: https://answers.unity.com/questions/254130/how-do-i-rotate-an-object-towards-a-vector3-point.html [Accessed 3 Apr. 2020].

[64] Unity. (2020). “Skybox”. Available At: https://docs.unity3d.com/Manual/class-Skybox.html [Accessed 10 Apr. 2020].

[65] Pulsar Bytes. (2017). “SpaceSkies Free”. Available At: https://assetstore.unity.com/packages/2d/textures-materials/sky/spaceskies-free-80503 [Accessed 10 Apr. 2020].