Augmented Reality Card Game

CM3203 - Final Report

Joseph Wilson

Supervisor: Dr Bailin Deng

Moderator: Dr Philipp Reinecke

School of Computer Science and Informatics
Cardiff University

This dissertation is submitted for the degree of
Computer Science (BSc)

May 2020

Abstract

In this paper, I will showcase my project being based upon a video game, which gameplay is based upon real-life cards. It utilises augmented reality for image recognition to enhance the interaction of a real-world environment through computer-generated perceptual information. The report will showcase the development stages of the implementation from the specification and design, providing my overall opinion in regards to success and limitations I will also state how this game would be improved in the future with more time and my overall reflections from this project.

Acknowledgements

I would like to take the time within this section to show my gratitude towards my supervisor Dr Bailin Deng for providing support, during this project, especially during this unprecedented times of COVID-19. He has allowed the project to be completed to a standard, which I am proud of even though the current circumstances, hindering the outcome.

The YouTube channel named Brackeys, also deserves to be mentioned, as the free content their team produces helped contribute to the development of the learning within the tools I have used. I would finally like to acknowledge Dr Daniel Finnegan for allowing me to use this proposed project idea, as the concept stems from him.

Table of contents

List of figures **xi**

1 Introduction **1**
- 1.1 Aims of the Project . . . 1
- 1.2 Scope of the Project . . . 2
- 1.3 Approach . . . 2
- 1.4 Broad Summary . . . 2

2 Background **5**
- 2.1 Context . . . 5
- 2.2 Existing Solutions . . . 5
- 2.3 Approach . . . 6
- 2.4 Tools used . . . 6
  - 2.4.1 Unity . . . 6
  - 2.4.2 Augmented Reality . . . 7
  - 2.4.3 Xcode . . . 7
  - 2.4.4 C# Scripts and Assets . . . 8

3 Specification & Design **9**
- 3.1 Specification . . . 9
- 3.2 System Architecture . . . 11
- 3.3 Main Menu Architecture . . . 11
- 3.4 Game Architecture . . . 13
- 3.5 Support Documentation . . . 14
  - 3.5.1 UML Diagrams . . . 14
  - 3.5.2 Gameplay . . . 14

4 Implementation **15**
- 4.1 Unity Development . . . 15

4.1.1 GUI Elements 16
4.1.2 Object components 16
4.2 Main Menu Scene 17
4.2.1 Scene Loader 17
4.2.2 PlayerData 18
4.3 Game Scene 18
4.3.1 BattleController 19
4.4 Augmented Reality 20
4.4.1 Vuforia Engine 20
4.4.2 AR Foundation 21
4.4.3 Resolution between Vuforia Engine & AR Foundation 22
4.5 Player and Enemy Entities 23
4.5.1 Scriptable Objects (PlayerCard/EnemyCard) 23
4.5.2 Player Movement 25
4.5.3 Enemy Movement 26
4.5.4 Player/Enemy Attacks 26
4.5.5 Player Skills 28
4.6 NPC Object 28
4.7 Audio Manager 29
4.7.1 Sound 29
4.7.2 Audio Manager Class 30
4.8 COVID-19 30

5 Results and Evaluation **31**
5.1 State System 31
5.2 Augmented Reality Detection 32
5.3 Database 32
5.4 Final Details 32
5.5 Testing 33
5.5.1 Gameplay performance testing 33
5.5.2 Initial Report - User testing 34
5.6 Evaluate methodology and programming languages 35
5.6.1 Unity Game Engine 35
5.6.2 C# 35

6 Future Work **37**
6.1 Current Implementation 37
6.1.1 User Testing 37
6.1.2 More Levels 38
6.2 Future Game Development 38
6.2.1 Artistic Implementation 38
6.2.2 Team Expansion 39
6.2.3 Database 39
6.2.4 Multiplayer 40
6.2.5 PVP 41

7 Conclusions **43**

8 Reflection **45**

Glossary **47**

Table of Abbreviations **49**

Appendix The initial sketches for the Game **51**

Appendix UML Diagrams **55**

Appendix An overview of the gameplay **59**

Appendix Supporting Code & Other Material **79**

References **83**

List of figures

2.1 A scene taken from Yu-Gi-Oh! to help visually demonstrate 5

3.1 This shows the flow state between the Main Menu and Game scenes 12

4.1 The hierarchy shows the different GameObjects that are implemented, as well as their relationship in the form of layers. 19

4.2 The scene view, showing how an entity is linked to the image target. Look at image target behaviour. 21

1 The initial designs for the main menu 52

2 The initial designs for the game scene 53

3 The initial designs for the battle system 54

4 This showcases the interactions between the components in the 'Main Menu' scene 56

5 This showcases the interactions between the components in the 'Game' scene based from the initial system architecture 57

6 This showcases the interactions between the components in the 'Game' scene after the implementation process 58

7 The main menu - starting screen 60

8 The main menu - settings screen 61

9 The main menu - level screen 62

10 The game scene - draw indicator 63

11 The game scene - player main phase 64

12 The game scene - enemy main phase (Wave 1) 65

13 The game scene - enemy main phase cont. 66

14 The game scene - enemy attack phase 67

15 The game scene - second main phase 68

16 The game scene - pause menu 69

17 The game scene - player select phase 70
18 The game scene - player battle phase 71
19 The game scene - player battle phase cont. 72
20 The game scene - player battle phase (effect) 73
21 The game scene - Wave 2 74
22 The game scene - more examples of gameplay 75
23 The game scene - Victory menu 76
24 The game scene - Defeat menu 77

25 This figure shows the general working environment, whilst using Unity game engine. 80
26 This figure shows original code used for AR Foundation for the prototype of multiple image tracking 80
27 This figure shows the animator window and how the different states interact. 81
28 This figure shows the different methods implemented for the basic AI of an enemy GameObject. 82

Chapter 1

Introduction

Modern-day advancements of technology have led to wider integration of computer system within everyday objects. A large stem of these fundamental ideas and dependencies upon computer systems have crossed over to the structured form of play, games, which has evolved into video games. The culture that has developed from video games has become a backbone and a way of life for this century. Careers, job opportunities and even celebrities have stemmed from video games, establishing the influence it has on today's society.

2 pioneering technologies integrating gaming into everyday objects and the real-world environment are: virtual reality [24] and augmented reality [18]. Pokémon Go [28] is an example of a widely successful application that utilises augmented reality for its driving feature and unique gameplay interactions.

1.1 Aims of the Project

This project aims to use the pioneering technology of augmented reality to enhance the experience of a real-world environment, utilising a card game as the backbone structure, showcasing the amalgamation of real-world objects and modern-day technology through computer systems. Consequently, I will be creating an augmented reality card game that utilises sensory modalities to be able to blend visual GUI overlays, models and so on through sensory projection into our real-world, allowing the user to be able to interact.

The intended audience of this project can loosely be anyone interested in games or involved with augmented reality development. The age demographic can more appropriately be statistically measured by the average age of video game players [22]. The game I am creating like any other game's success and appeal cannot be completely measured in the form of quantitative data, as with anything of a similar nature it is qualitative being based upon the user's opinion.

1.2 Scope of the Project

The scope of the project initially encompasses the original primary deliverables: develop a functional working smartphone application, integrating augmented reality into the card game, make the card game intuitive, but also fun for a sense of purpose. As with any large project development, the project scope has altered throughout the development process: this can be a cause of a lack of time management, lack of knowledge regarding capabilities, as well as external factors we do not have control of.

This will all be developed upon further within this report, explaining the impact of the said issue and how I have tried to resolve this to the best degree possible. The project scope and development process change of focus are also supplemented by the entirety of this project nucleus being a game. This, therefore, means that throughout, development changes were made in the hopes of making the gameplay more interesting following principles mimicking the standards used in already established video games.

1.3 Approach

Due to the complexity of this project and the deliverables required, I decided that the best approach would be to use an individualised Scrum agile development methodology. This methodology utilises a Sprint being a one time-boxed iteration of a continuous development cycle [27], meaning that I would have to achieve a set goal by the end of the said time period.

I chose to use this development method firstly based on the fact that this is an individual project, meaning that I have the freedom to work at my own pace to best suit my work methodology and style, however it allows for concrete timing to ensure that I am working on course. This flexible approach allows the dedication of different segments within this project to be completed, but more importantly, it allows reviews to be conducted in case a segment has not worked as intended or simply requires more time.

1.4 Broad Summary

From a personal point of view this problem that it is based upon allows me to conceive a premise, which a decade ago would not be possible during my childhood. It is based upon the concept of a childhood TV show and manga: Yu-Gi-Oh! [23], hence why I decided to use this project idea from Dr Daniel Finnegan, which was done with his consent and permitted if stated within acknowledgements.

I would like to explicitly state that the premise for the idea stems from YuGiOh!. No intellectual property has been infringed, as no cards, gameplay or models have been used from the series Yu-Gi-Oh! or other trademarked entities. I have only simply mentioned this in terms of allowing the reader to have a more concrete understanding of the project in its entirety and the vision Dr Daniel Finnegan helped use to explain this project idea. The rest of this paper will encompass the following contributions:

- The R&D process of this project in terms of the software and necessary packages required.
- The overall implementation of the design of the system.
- The changes that have been made throughout this project.
- The future development in terms of what I would do to further progress if I had more time.
- An evaluation of the final deliverable of this project.
- A reflection on myself and what I have gotten out of this project.

Chapter 2

Background

2.1 Context

The popular trading card game Yu-Gi-Oh! from 1996 has provided the vision of the interaction of the foreign entities being holographically placed within the real-world, fusing real-world objects with the enhancements of technology. At the given time it was simply regarded as a niche feature – an idea within that generation cannot be executed to a given desire, but only demonstrated through the manga or anime.

Fig. 2.1 A scene taken from Yu-Gi-Oh! to help visually demonstrate

2.2 Existing Solutions

Over the past decade, the presence of augmented reality being used within video games has increased. The most mainstream example to enter the market is Pokémon GO. The market

for video card games has also increased with major well-established video game developer companies such as Blizzard, creating hit games like Hearthstone [21].

The context of the problem is therefore fundamentally creating a functional card game that utilizes augmented reality, making the seemingly impossible concept of the real-world integrating with technology in 1996 a reality. An internet browser search of the simple term 'Augmented Reality Card Game', showcases that some independent developers have dabbled into this premise of essentially instantiating models upon cards, with most having limited functionality in terms of the gameplay aspect.

2.3 Approach

My approach to fulfil the aims of this project was that evidently, I needed to produce a form of card game that utilises augmented reality to provide a unison of the real-world, instantiating game objects upon the image targets being cards. I further developed this by creating an augmented reality card game that is dynamic and utilizes modularity (explained further) to allow easy expansion: level creation, adding more image targets and linked models, audio sources etc. I will, therefore, showcase a solution to the problem that shows a basic level idea of functionality, but would allow easy expansion if other required video game creation skill-sets were applied which are not applicable to this problem, such as model creation or graphics design.

2.4 Tools used

2.4.1 Unity

The game engine of choice is Unity, which is a cross-platform game engine developed by Unity Technologies. This platform allows the universal development of both 2D and 3D games for a multitude of platforms including desktop, web, mobile and console. The utilities within Unity have made it so it has been adopted by industries outside video gaming, such as film, engineering, and construction.

The fundamentals of Unity are that it utilises an entity-component system [3] architectural pattern allowing the user to add objects/entities within the game engine. The objects/entities are then manipulated by adding various components to them, which are either provided by the Unity engine itself such as basic colliders or audio sources, or alternatively, a custom script can be added to allow the game engine to do so as you desire.

After exploring different options and extensive research, Unity was justifiably the most applicable option to help tackle this problem, as it fulfilled the most requirements compared to options of other game engines, IDE, or any alternative options. This is a subjective opinion based upon my research of how to best solve the problem, however not experiencing other methods firsthand I cannot say this is definitively the best approach. The foundation of the problem is that it is an augmented reality application, which limits a lot of possible choices in terms of software required to handle these project requirements. Unity fulfils this but also has other key instruments necessary, including a 3D and 2D engine. This is required for the user interface and models of the game objects, and it allows for the development of being built for smartphone devices. The other aspect is the learning curve required to best utilise the high-fidelity and functionality of Unity being most suited to ‘developers in comparison to its main competitor: Unreal Engine.s

2.4.2 Augmented Reality

I explored 2 different main approaches to fulfilling the augmented reality aspect of this game, which both have their advantages and disadvantages that occurred during the implementation aspect of the development process. The first and suggested augmented reality software platform is the Vuforia Engine [26], which allows easy functionality of adding advanced computer vision to any application. The second option is the new AR Foundation [16], which is built upon subsystems (platform-agnostic interface for surfacing different types of information). In a general consensus, they are both packages that can be easily installed within Unity. The development process will showcase the advantages and disadvantages of both AR packages and my overall justification on why the Vuforia Engine was implemented in the current build of the video game.

2.4.3 Xcode

Xcode being an IDE allows applications to be created through the development environment, so a multitude of iOS applications can be created. For the purpose of my development build, I will not be using Xcode is this manner. Rather, it will be used as it allows easy user profiles to be built from the signing of certificates, bundle identifier, version, and target iOS device, which are all needed for the deployment. Xcode is specific to macOS, so the overall ability to create iOS devices means that a form of Apple device running macOS is required, fortunately, I have an Apple MacBook Pro. This could be an issue for someone that does not have access to a macOS device, especially during the current situation. They would be able to develop taking into consideration of an iOS device, but would not be able to fully deploy it.

2.4.4 C# Scripts and Assets

The supporting documentation contains all of the C# scripts that are in the final build. The C# scripts showcase the implementation of these components with each GameObject. This is in relation to the UML diagram should provide information on how each C# script has been used. The assets regarding the images and SFX are all royalty-free and chosen with this consideration. The documentation also contains the assets regarding the packages installed from the Unity store.

Chapter 3

Specification & Design

3.1 Specification

The specification of the project stems from the initial report, as well as the overall aims of the project. The overall premise was simple I need to construct and design a video game that allows the implementation of augmented reality within the real-world, allowing the user to be able to manipulate and interact, fulfilling the requirement of any video-game which is enjoyment. The term enjoyment is a subjective and opinion based and was not able to be measured in a qualitative data manner, which will be justified upon in section Initial Report - User testing. The final specification of the game is an evolution of the video games development, compromising between realistic outcomes and alternatives to limitations that are involved. Different areas that I originally envisioned for the specification were PvP and a larger database of models, why each did not occur is all thoroughly explained within the report.

- The first general requirement was that I want the video game to be developed by being connected through different modular components, as the video game is card-based. Research of other relevant projects that use a card-based system showcase that overtime expansions occur, such as adding new cards to the video game, so the architecture needed to allow for this dependency, hence the modular approach used. This approach was used throughout the whole creation of the system architecture.

- The understanding of other video card games conceded the concept of a state system would be needed so that a smooth flow of the interaction between the user upon the GUI elements, placing the image targets and other functionality such as the battle aspect could occur.

- The next requirement is that it needs to be able to successfully detect an image target, augmenting the assigned model into the real-world. The models will then need to have some form of mechanics implemented, allowing them to perform tasks needed within the gameplay aspect of this project.
  - The final decision for the augmented models and objects is that they allow interaction with each other through controllers for movement, buttons for gameplay aspects like attacks and collider-based detection to make a trigger occur. A basic AI system needs to be in place for the enemy objects so that PvE can occur.
  - The augmented models also needed to have corresponding assigned images in a database for the augmentation. The database also needs to be able to be expanded easily for future implementation. This is further explored in the section Future Work.
  - The models themselves are player, enemy and NPC entities, so they needed to have different gameplay aspects such as assigned variable attributes and operators, allowing for gameplay features such as: health points, attack and defence statistics and special abilities.
- It also needed to be able to perform the basic mannerisms that are normality within modern video games. These include:
  - It needed to be able to flow between different scenes passing required variables, which in this instance is allowing the user to interact with the main menu, so that they can initiate a level.
  - The GUI of both the main menu and level scene needed to have interactable objects that perform basic tasks. The implementation needed to allow for different GUI elements which are easily recognisable on their functionality so that the user can have smooth navigation experience and be able to play the game easily.
- The final aspects of what the system does are the finer details that were added to the video game through a better understanding of the complex game engine Unity, which I was able to add during the implementation process. In hindsight, these are aspects that should have been added to the original specification, as it allows for a higher-level end product. The additions were also due to a compromise of some of the limitations of work that could not be done, hence they were added to counter-pose the overall workload requirement from this CM3203 project.
  - Animations for the models.

- SFX for the gameplay interactions.
- An audio manager.

Refer to The initial sketches for the Game to see the initial sketches of the video game.

3.2 System Architecture

The system architecture is based upon the overall structure of the development of a game within Unity and the evolution of my understanding, which as mentioned earlier is assigning scripts and other components to in-game objects to perform operations. This is the final system architecture which also takes into consideration my initial designs and thought process of designing the game: GUI, mechanics, functionality, and gameplay. This all works smoothly with conjunction to my modular approach, as it allowed the assignment of different components to objects, allowing them to work as the main controllers and managers.

The controller and managers are divided between the 2 main scenes being the 'MainMenu' and 'Game', as they both have different requirements, but need to work in harmony to pass the required information. The 'Game' scene is the basic first level and throughout the report a better understanding of how it fulfils my requirement of: 'I, therefore, will showcase a solution to the problem that shows a basic level idea of functionality, but would allow easy expansion if other required video game creation skill-sets were applied...'

3.3 Main Menu Architecture

- **Scene Loader:** acts as the component for controlling all of the interactable fields (user input, button, and sliders), as well controlling the difficulty level, audio level and resolution, but also fulfilling any triggers that allow the 'Game' scene to load.
- **Player Data:** this is the handler for storing the user inputted data and the difficulty index. The player data in further iterations would be replaced with a login type system.
- **Level Manager:** this is the handler to see if a level has been won, displaying the information to the user. The basic methodology of the level handler means it can be expanded for future levels.

Specification & Design

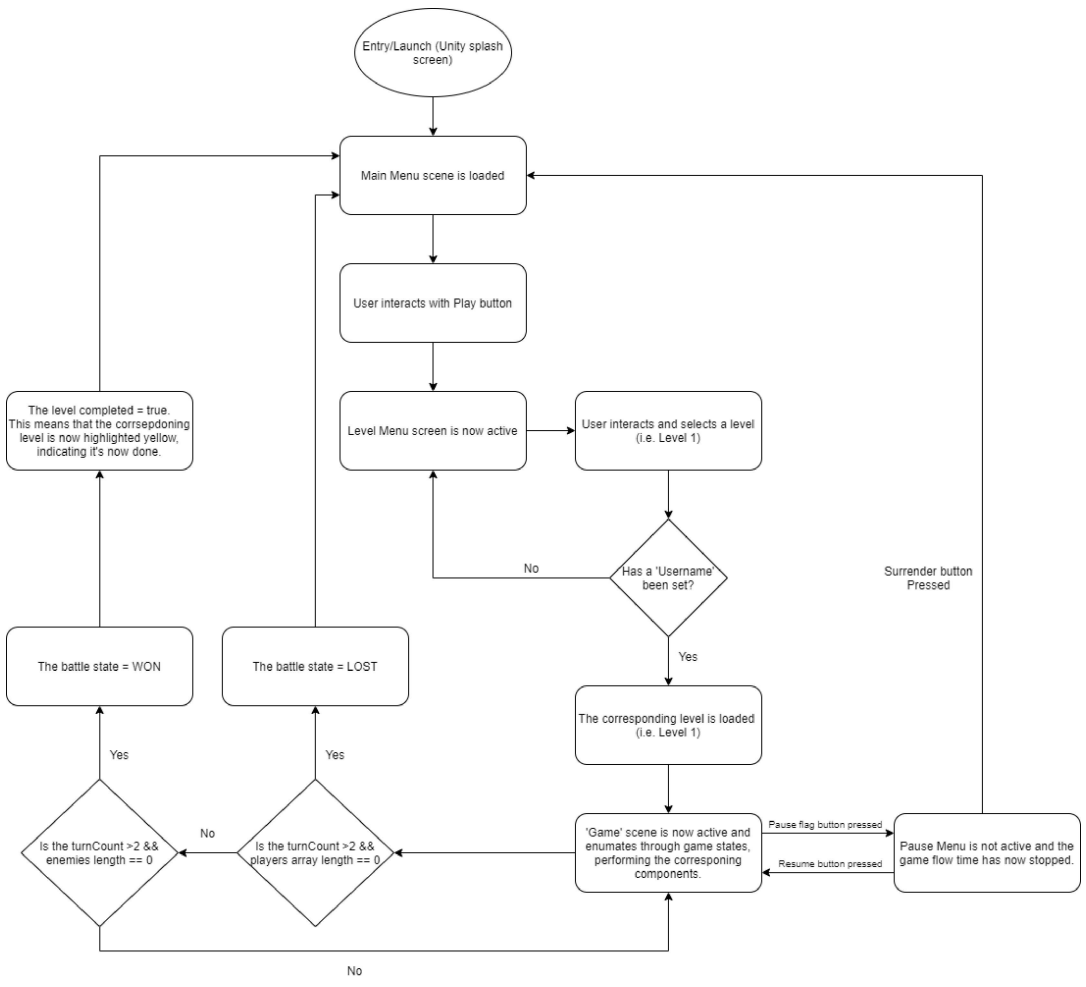

Fig. 3.1 This shows the flow state between the Main Menu and Game scenes

3.4 Game Architecture

- **AR Camera:** this is the interface that allows the capture device to be displayed on the user's computer or smartphone. This from the augmented reality package.
- **Audio Manager:** this is used to manage any audio files assigned within the video game, controlling the audio sources using the Unity scripting API. The audio manager I have created, however, evolves the basic level scripting API, making it more modular and expandable.
- **Battle Controller:** this is the heart of the system where all of the inputted information will pass through and be transferred to the other GameObjects. It controls the game flow of the game using an enumeration algorithm for different game states, allowing certain triggers to occur. It also contains all of the objects for the GUI displayed information such as reading the static information stored within the Player Data located within the 'Main Menu' scene. The BattleController also contains the components for 2 other managers:
  - **Entity Manager:** this is a dynamically assigning array that allows the detection of the player and enemy entities using the Unity tag system. It is dynamic in the sense that the array will update and allows any player and enemy entity to be added or removed, reallocating the array's size. It also utilises the dynamically allocated array to allow for a target for the enemy to be picked, as well as detecting if the player and enemy entities do not exist, meaning the game state won or lost must occur.
  - **Manager Level[number]:** this is a modular addition that is a requirement for each different level upon the gameplay. It also uses the enumeration algorithm, but uses wave states that can easily be adjusted, so that more levels could be added to the game.
- **Select Manager:** this is a manager that uses ray casts to detect the user touch or mouse input, so that the person can select the player entity they want to control, using the Unity layer-mask feature. This means that any GameObject can be selected through this manager if it has the layer-mask of 'Player'. It also assigns a shader to allow intuitive detection of what the player entity has been chosen.
- **Player Controller:** this is the controller that develops upon the 'Select Manager' for allowing the movement of the selected user's player object through the interaction of GUI elements, which are a joystick and buttons.

- **Enemy Controller:** a basic AI system needs to be implemented for the PvE aspect of gameplay, so the different enemy entities can intuitively pick an object to a target, controlling their movement and attacks performing the requirements needed from them.

- **Modular Cards:** the storing of data for both the player and enemy entities uses a scriptable object that provides a template for all the required data fields within the card. It allows for easy expansion, as the user simply needs to fill in the template of attributes from the scriptable object and add the component to the 'PlayerStats' script field. It is modular as the creation will not require any other changes and will read the data stored within the card, but just the unavoidable components: scripts, rigidbody, collider and animations need to be added for the individuality and creation of a new card linked to a model.

- **Vuforia Database:** the Vuforia Engine is very intuitive, as it has an online database where image targets can easily be added to the gameplay, which is simply linked through a license key and downloading the target manager database for the development platform of Unity.

3.5 Support Documentation

3.5.1 UML Diagrams

Refer to appendix B: UML Diagrams. To help support the rest of this report I have included UML diagrams both as an appendix and support documentation to show how the system architecture evolved from the initial components within the implementation process.

3.5.2 Gameplay

Refer to appendix C: An overview of the gameplay. To help further supplement the specificity and development of my overall specification and design choices of my video card game, it may be in the best interest to view the appendices of gameplay when examining this report and implementation process. It may provide a better insight into how the separate modular entities work in unison, showcasing the overall flow of the gameplay.

Chapter 4

Implementation

The initial sketches that showed my overall vision for the video game in conjunction with the creation of the flow diagram and system architecture gave me a strong foundation, to begin with. The information provided in the Specification & Design is, however, a more concrete blueprint visually auditing my design. The actual Specification &Design was more or less a work and development idea, but I had a general concept of how I believed everything should piece together. I did use this information accordingly to have a clear plan of the overall initial Sprint and beginning modules that had to be created.

As addressed throughout this report, the game engine Unity and the programming language C# is foreign to me, so the initial phases of implementation that were required was to build and develop the game in unison with expanding my overall knowledge of the best methods to do so. The foreignism of using a new game engine, Unity, as expected came with some inherent and unforeseen issues that will be discussed. Unfortunately, the disaster of COVID-19 occurred during the flow-state of this project, which also contributed to some unforeseen issues.

4.1 Unity Development

The foundation of Unity is understanding the entity-component system. I chose to spend the beginning weeks of this project following YouTube tutorials, expanding my knowledge of Unity. I also created some smaller Unity projects that helped develop my understanding, as well as providing the concept of what components would be best required to fulfil this project. The developed knowledge also helped inspire the overall system architecture, implementing features I did not originally invoke. An example of this would be the ‘Audio Manager’ and ‘SFX’. A general overview is using the Unity game engine is provided in figure 25.

The initial development of the final project went through several iterations and builds, constantly flourishing the premise, as well as streamlining each component. To make sure to take into account the modular approach. To help further clarify this statement. An aspect of the system architecture that may have originally been hard-coded in to test the functionality, which would then be revisited ensuring that it was more modular and could be used for an abundance of entities, accounting for any future development.

Through the different experiments and learning curve of Unity, I will discuss some of the main components utilised providing some clarification on my reasoning, as I try to communicate some of the final modular entity-components that help contribute towards the video game.

4.1.1 GUI Elements

An array of some of the different GUI element which is built within Unity is utilised. The GUI elements are basic and self-explanatory that are featured in most video games including: a canvas, panels, buttons, sliders, text, input-field, and dropdowns. The GUI elements already have a basic script implemented, allowing for methods to be called upon when interacted. These methods can be already basic Unity methods such as SetActive(bool), which simply will active/deactivate the GameObject depending on the Boolean value. Alternatively, the scripts implemented can be custom methods which have been scripted by the developer, allowing the GUI elements to provide more functionality.

4.1.2 Object components

Unity provides an excel number of components that through methods can be manipulated to provide ‘life’ to the video game. However, firstly I would like to state, the components can only be fully utilised through the manipulation of scripts, invoking their methods. The components that Unity provides were extremely helpful in the development process of the enemy and player models, as it helps handle the physics required, collision detection and animation states.
The main Unity components used:

- **[name] Collider:** the collider component can have both 2D and 3D shapes associated that provide a level detection through the manual setup of adding a detection zone, triggering the methods that are associated when in or out of this detection zone. An example of this would be if one object passed through another object collision detection zone, it triggers an attack event to occur.

- **Rigidbody:** the rigid body component provides the physics for the object meaning that the object will now have mass allowing it to apply to the laws of gravity. This was mainly utilises in the movement of both player and enemy objects, as its requirement is to cater for movement.
- **Animator:** the animator component utilises a state system to change the animations of an entity which can be manipulated to occur as desired. For the object to be able to animate a (.FBX) file is required.
- Prefabs were also required.

4.2 Main Menu Scene

The main menu scene was the first implementation of this project. I chose to start with this because as shown in the UML diagram the overall complexity for the main menu is lower in comparison to the Game scene. It provided a good foundation for understanding how the different objects worked and how to properly assign the different components. The entity component system worked coherently with my initial sketches, as I was able to place the GUI elements within the canvas before adding the scripts to allow me to manipulate, as I desired.

The initial GUI design was very simple only having implemented the play button to load the next scene, but throughout the project, I implemented more user-friendly features, which you would expect in the main menu, hence the 'settings' button. I utilised the button scripts of On Click () to be able to set different GameObject panels to SetActive(bool), meaning that a simple method of transitioning to different menus within the Main Menu scene was implemented. The UML diagram showcases the different attributes and operations administered within each script.

4.2.1 Scene Loader

The 'Scene Loader' script utilises a Unity scripting API being 'SceneManagement' and 'Audio'. The 'SceneManagement' API, as expected allows Unity to able to transition between the different scenes in the building settings either through their name or index value.

The 'Audio' scripting API allows audio to be implemented within the video game, which in this instance is just background audio, providing an ambient feel. I utilised a slider for this implementation calling upon the 'SetVolume(float volume)' methods. It utilises a Unity feature of being able to assign a dynamic float variable in unison with the range being [-80,

0], which is the Audio Mixer dB range, so the user can control the volume accordingly. The other methods implemented in this class use the same methodology to be able to dynamically change the functionality within the game calling upon the index value to cause an effect to occur.

4.2.2 PlayerData

The 'PlayerData' script was one of the final implementations, allowing the video game to be unique to each user by allowing them to set their 'username', as well as passing the difficulty index. The inputting of the username was purposefully very basic, as this current state of the game does not have PvP or save states implemented, hence it would be unethical to require excess information that could cause potential security risks such as an email address and a password.

The implementation of the 'PlayerData' script was enacted at a later stage, due to my lack of knowledge within Unity. The passing of variables and methods are usually conducted by calling upon the GameObject within another script. However, this was not possible, you cannot pass GameObjects without calling upon a prefab or implementing a 'DontDestroyOnLoad' function call, as GameObject components only last during the lifetime of said GameObject. This was an approach I did not want to use. The UML diagram shows that a simpler and easier fix was devised, wsich was making the variables 'static', hence the variables last the lifetime of the script instead.

The main menu, as expected due to the limited complexity was a very smooth process, therefore no major issue arose. I do believe however, it is important to state within my implementation, as it showcases the realisation and initial steps for the overall creation of the more conglomerate components utilised within the 'Game Scene', fulfilling my aim within the Specification and Design.

4.3 Game Scene

The implementation of the game scene is the main body of workload of this project fulfilling the aims from the specification. The game scene is a representation of the further development of this video game, showing how the modular design that integrates augmented reality could be further expanded upon. The overall system architecture and implementation process is an example of one level, however by simply duplicating the scene and then making minor changes to certain scripts a new level can be generated. The expansion also adheres for the

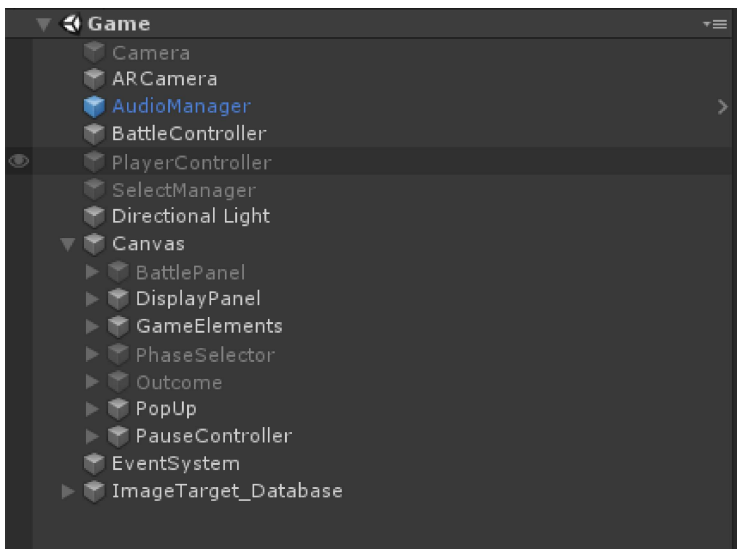

Fig. 4.1 The hierarchy shows the different GameObjects that are implemented, as well as their relationship in the form of layers.

fact this is an augmented reality card game, hence future iterations of builds and expansion of cards should be accounted for within process.

Similarly, to the development of the 'Main Menu Scene', the starting process involved laying out the different main GUI elements, building a blueprint on how the scripts will manipulate each GameObject. The interface followed a rough overview of the initial designs I provided. As stated, the UML diagram during the implementation process was not as thoroughly documented. I do not believe this caused any major issues, as the video game development simply followed Sprints by 1 a one-man team, thus no need to coordinate with others.

This is also supplemented by the addition of features that were added through the evolvement of my understanding of Unity. The expansion of adding different modular entities for controlling the gameplay involves creating a GameObject and adding a script component. The implementation does support having one GameObject with all the components added, but this defeats the overall design requirements.

4.3.1 BattleController

The UML diagram showcases that the BattleController GameObject is the heart of the system, as stated in the Specification & Design as well. It shows that in some form or method most of the different GameObjects or C# scripts pass through either the GameObject or the ditto named C# component, BattleController.

The BattleController main functionality is to cycle through the flowchart 3.1, enumerating through each of the different BattleStates, the relationship of which is shown through the realisation of the enumerator embedded within the script. It will enumerate through the main

cycle until a goal condition of BattleState.WON or BattleState.LOST has been reached. The different BattleStates are passed as a static variable, as the information needs to be passed to the other GameObjects, so their functionality can occur at the appropriate time.

IEnumerator

The sheer size of the class within the UML diagram through the number of attributes and operations is because the BattleController also handles the large array of GUI elements, providing some form of method to be able to manipulate them. The GUI elements have been split down into GameObject subsections based on the overall design requirements and relationships they have with each other The hierarchy shows the different GameObjects that are implemented, as well as their relationship in the form of layers.. This is because the BattleStates and what GUI elements are currently visible to the user work in conjunction, thus justifying this implementation.

4.4 Augmented Reality

The whole project specification is based upon the fact this is an augmented reality card game, therefore I believe it is instrumental that this feature is mentioned early within the overall implementation, as it is especially critical to the system. The AR system being a package allowed it to run in conjunction with the overall design, as it is a modular aspect that did not affect the overall creation of other video game features, hence it went through a life-cycle of being implemented or removed for easier testing purposes. The overall testing was made slower constantly utilising the AR camera, therefore simulating the environment helped increase the workflow. The AR system, however, went through the most difficulties, due to either lack of software capabilities, current issues, or external factors such as COVID-19. The AR package that was utilised swapped between Vuforia and AR foundation, due to each having their software limitations.

4.4.1 Vuforia Engine

The original project description from Dr Daniel Finnegan suggested using the Vuforia Engine to control the AR contents within the video game. I, therefore, implemented the Vuforia Engine originally to test out the overall instantiation of models upon the image target, testing out the overall performance and environment experience. The usage of Vuforia Engine is also supported by the fact that it is used by wide known companies such as Mercedes and LEGO® NEXO KNIGHT [4], ensuring that it has the overall functionality required.

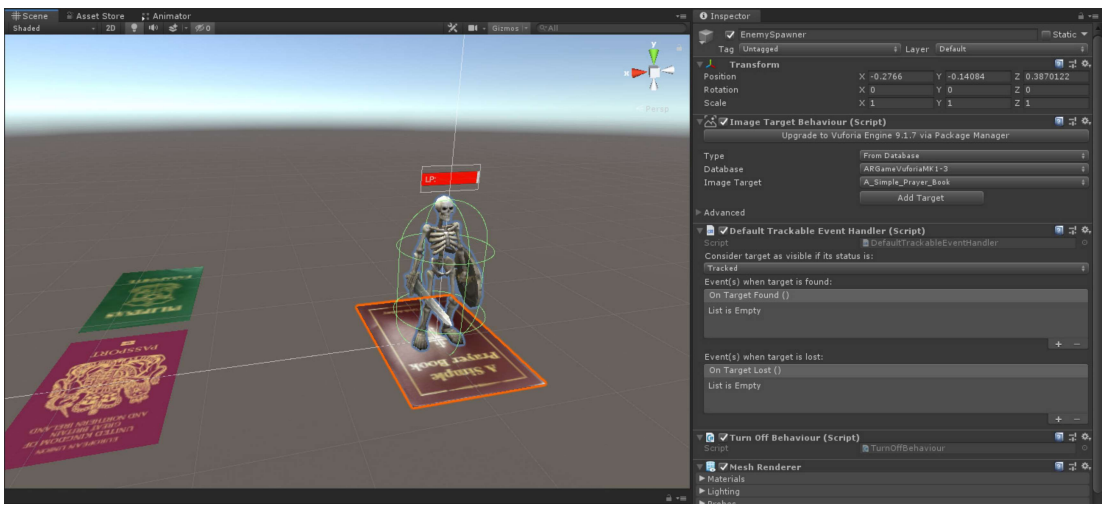

Fig. 4.2 The scene view, showing how an entity is linked to the image target. Look at image target behaviour.

The Vuforia Engine in comparison to the AR Foundation is more user friendly in terms of implementing the package within Unity. This is further supported by the design and approach method used for linking Vuforia Engine to Unity, as well as linking the image targets to the models. It uses an app license key to link Unity to your Vuforia account.

The realisation of implementing the database required for the image targets (being cards) is Vuforia Engine's stand out feature. It utilises a simple online website to associate the images targets required for the cards, which are then simply downloaded and linked to the Unity project.

Vuforia Engine Problems

The current build of Vuforia Engine 9.0 in conjunction with Unity 2019.2.21f1 caused some issues that unfortunately could not be resolved through multiple trial and error of different builds of both. The issue is that currently when the Unity project is built and tried to run on XCode for the iOS development, the build will break when opened on the iPhone. I came to the deduction that is an issue with Vuforia Engine 9.0 as I experimented by removing Vuforia Engine 9.0, adjusting different settings required such as what version of iOS to build for, as well as the different build settings data required such as permission for the iOS camera usage. Ultimately, I found out that the same Unity project would open perfectly on the iOS device if the Vuforia Engine 9.0 were removed.

4.4.2 AR Foundation

From the initial report, this project was intended to be built for a mobile device, as it is the more natural device to utilise augmented reality, due to it being built for a handheld smart-

phone. A smartphone development from throughout the past decade also works coherently due to all of the features: high-quality cameras that are constantly improving, touch controls, audio drivers and advanced gyroscopes for the movement of the device.

I, therefore, decided to try advert this issue with Vuforia Engine 9.0 and overcome it by developing scripts for AR Foundation. AR Foundation is a platform that is provided and recognised by Unity, but unlike Vuforia Engine it is not as easy to integrate. The current features that it possesses and how they are implemented are not as streamlined requiring more components and scripts to be utilised, therefore a higher learning curve was required.

I spent a week trying to develop my knowledge for AR Foundation, creating scripts that successfully worked in instantiate an object upon an image target. AR Foundation does not have a database, but instead a library where image targets are stored in an array, which I scripted to line up to an array of objects based on an index/ Refer to figure 26.

The script worked perfectly and managed to fully augment the object upon the image target. It also seemed to work in the sense that it was perfectly built for my iOS device, providing smooth and accurate image detection for the object models. However, there was one major issue, even though my images would instantiate the corresponding object based on index value – only 1 object would instantiate at once. It did not support image detection for multiple tracked objects, which was essential for this project.

4.4.3 Resolution between Vuforia Engine & AR Foundation

The augmented reality integration within this project went through multiple life-cycles as explained, trialling and comparing the current packages available. This was a portion of the implementation process which took a significant amount of time, as each trial run required the device to be built on my computer, then transferred to XCode and then fully deployed on my iPhone. The clarification on what package to use was ultimately based upon the overall requirements from the specification, as well as the unforeseen issue of COVID-19.

The device to be a working card game required multiple image tracking, so an object can be instantiated for both player and enemy. The discussion between me and my supervisor concluded this outweighed the concept of it being on an iOS device, as it is built within the Unity engine for this purpose.

For the future work when the Vuforia Engine problem is resolved this could simply be addressed, consequently it could be implemented for an iOS device with little work. The COVID-19 factor supports this decision as we expected the Project Viva to be demonstrated online, hence to able to testify my project it would be easier if it was displayed on my Windows device where it will be run in a simulator using a webcam. The webcam has its issue, due to the overall image quality being lower and lack of proper image stabilisation this

may cause some jittering and bugs. It does allow for the demonstration of proof of concept for the project.

The concluding compromise was to use Vuforia Engine for this project. Hypothetically as demonstrated in the UML diagram, the AR package could be either the Vuforia Engine, AR foundation or even an alternative one if all requirements were met, as it's just a module entity that uses its GameObjects from the package contents to collaborate with the other entities involved for the outcome.

4.5 Player and Enemy Entities

The player and enemy entities have a lot of similarities because of their overall functionality and requirements needed from each GameObject in terms of gameplay. The architecture for both is purposefully similar in terms of the structure of components including: an animator, rigidbody, collider and several custom scripts. This is all to account for the overall future expansion of the card database, as well as migrating the system to include PvP.

The development of each script was an emphasis on my aims and system architecture being modular, so I devised each different script to have a particular intention. The scripts were all designed to be universal when possible, however, it is not the case for the script that processes the player/enemy attack scheme simply as this is an expression of individuality.

4.5.1 Scriptable Objects (PlayerCard/EnemyCard)

The scriptable object data container was used for the creation of the cards for both player and enemy entities, due to the data container allowing independent class instances. The UML diagram showcases the base attribute the scriptable object is responsible for, containing information regarding the different card stats needed to make each card unique.

The scriptable object data container was used for the creation of the cards for both player and enemy entities, due to the data container allowing independent class instances. The UML diagram showcases the base attribute the scriptable object is responsible for, containing information regarding the different card stats needed to make each card unique.

The scriptable objects are a fine tool to use, as they can simply be created as an asset having their stats easily modified in the inspector. This demonstrates the ease of expansion if more cards were to be added to the system, creating a new card that is assigned for each entity, profiling it by filling out the attributes as the developer desires. For a gaming aspect, this is also crucial for balancing out the different cards in respect to one another, ensuring more

competitive gameplay is achieved. Each card should have its own strengths and weaknesses; a more tactical strategy is needed for choosing each card.

PlayerStats/Enemy

This is a simple data handler, reading the information from the assigned scriptable card object, creating a separate instance of all of the data attributes. The method of calling upon the data, using a handler to save each attribute is required so that manipulation can occur to these stats without overwriting the scriptable objects. The design allows any corresponding scriptable object to be applied with the corresponding class name, storing the information to be manipulated by the other classes. The script is also responsible for some basic operations such as setting the values of the cards on awake, taking damage/hitpoints, as well as destroying the GameObject when a condition has been met.

Animator

The animation system utilised within Unity is very rich and sophisticated for easy manipulation of the GameObject model. The figure 27 shows the different states and the relations between them, allowing a smooth flow of transition. The implementation of the system allows the different states to transition based on different parameters, triggers, and entry states. The animation for the objects allows them to be visually more appealing, conveying more information to the user.

Alternatively, different scripts can be called upon during different states of the animation transition state, an example of this would be the death animation calling upon the Dead() method. The utilisation of parameters, triggers and the calling of the method based on the animation state was the main implementation process for the movement and attack functions. The main problem with methodology calling upon the different methods during animation states is that it is dependent on the model having suitable animations provided. It would be an over-ambitious project aim for me to create a large quantity of player/enemy models with appropriate animations, as it is out of my jurisdiction and skill-set.

The implementation of the player/enemy methods calls is dependent on this design, as well as the fact I believed it showed a better understanding to have unique individualised GameObjects with a higher degree of functionality than mundane simpler 3D objects with similar functionality. I decided to use free assets that have the basic requirements needed, making the current player/enemy card models have a high level of functionality. The overall design scheme of this video-game, which is being constantly reiterated means that the database of cards can be easily expanded upon if more well-animated models were obtained.

4.5.2 Player Movement

Select Manager

The initial state of deciding which object is selected occurs during the BattleState.PREPLAYERBATTLEPHASE, where the GameObject 'SelectManager' becomes active.

The select manager uses ray casts to get the input of the user, which due to Unity API can be either a touch input intended for a smartphone or in this case a mouse button down interaction. The select manager uses ray cast to detect, which GameObject has been chosen for user control as prompted during the gameplay. It uses a custom shader to outline the chosen GameObject for ease of recognition where the assigned GameObject is stored in a static variable, allowing it to be manipulated by other scripts such as 'PlayerController'. During the BattleState.PLAYERBATTLEPHASE, the script is disabled making the selectObject static variable null, so that process can occur during the players next turn.

Player Controller

The player movement is managed through the 'PlayerController' script is responsible for the detection of the user input on the provided joystick that becomes active during the BattleState.PLAYERBATTLEPHASE. During the OnEnable() method call, the currently selected entity from the SelectManager static variable (refer to . . .) are passed into the PlayerController script. This is another design influence of it being modular, as the PlayerController enables during the PLAYERBATTLEPHASE accessing the relevant information, but then is cleared when it is disabled during any other BattleState. The modularity allows the joystick to control any Player GameObject that is selected, as well as changing the controller Player GameObject each turn, providing the user with more freedom on player control, invoking more tactical gameplay through choosing with Player GameObject to control each turn.

The player movement is conducted through the Rigidbody component, as it provides motion through the Unity physics engine. The implementation works by detecting the amount the joystick is displaced from the origin, providing a velocity based upon both the displacement and the moving Force (being the GameObject movement speed stat). If the joystick is displaced at all from the origin it invokes an animator trigger, causing the movement state to occur. It also uses unit quaternion to ensure that the object is facing the direction of the joystick displacement.

4.5.3 Enemy Movement

The enemy movement uses a similar process utilising both the rigidbody and animator component, however, due to it not being user-controlled a basic level of AI has been implemented. The figure 28, as the implementation of methods is unique to the state machine. The movement is instead controlled using state machine behaviour whenever the enemy GameObject is now in the corresponding movement state. The transition to the movement state is simply called upon within the BattleController, during the EnemyAttackPhase() coroutine, as the movement state called “Walk” has a Boolean trigger that is set to true.

4.5.4 Player/Enemy Attacks

The methodology of implementing the player attack schemes used a simple button layout with cooldown function, allowing control of the player GameObject skills. The enemy attack scheme is simply called at random, during the player ’Walk’ state machine behaviour once they get into range. The player/enemy attack script is independent for each assigned card player/enemy GameObject, as each player/enemy GameObject needs to be unique in terms of abilities.

The current video card game development provides an example for both a melee range and long-range player GameObject. The enemy GameObjects are both melee range, but the boss entity has different melee attacks, due to the increased amount of animation effects that the game object has. The scripts are both unique for each player/enemy GameObject, however, they do share some fundamental concepts. The different attack scripts utilise the fundamentals of a lot of the different Unity scripting APIs and the managers I have implemented.

Melee Range

The melee range attacks for both the player/enemy GameObjects are purposely designed to very similar, utilising layer masks to depict whether the GameObject is either a player or enemy layer. The cooldown (decided in the PlayerCard scriptable object) for the player buttons use an enumerator to make the buttons interactable or not, allowing the attack animation to occur. The enemy object as stated will randomly call upon the attack method when in range, during the state machine of “Walk”.

The enemy attack method is called, during the attack state animation of the Enemy GameObject. Utilising the flow of the animation allows the attack scheme to occur more naturally, the method can be called out the apex of the animation. An example of this would be a sword swing where the method can be called at the pinnacle point of contact that would

occur. However, since the player GameObject is user-controlled, the attack event can be called during any position, playing the assigned animation for the interacted button, although if not in the collision region no effect will occur. Each attack has a random chance of hitting, providing more dimensionality to the gameplay.

Each different object will have an assigned transform being the 'attackPoint', due to the various sizes of each model. The attackPoint will also have a range based on the player/enemy card stats called attackRange, which assigns a 3D physic collider that when overlaps with the assigned layer mask object collider region allow the attack method to be called. An example of the layer masks region is fundamentally the opposing GameObject entities, so a player attack layer mask will be enemy and visa versa.

A foreach method is then called for player GameObject for the collision occurrence, calling upon the assigned operators for the calculations of damage operations, setting HUDs, play SFX and Boolean operations to detect if a GameObject health is (or below) 0. The enemy GameObject works in a similar method to call upon the operation and play SFX, however, does not require the collision detection, as the attack method will only be called when in range of the target GameObject based upon the attackRange value. For basic AI implementation the number of hit points deducted from the player GameObject is random (explained further in results, as it showcases gameplay). The boss entity has a similar call to the main attack method, but has a 'SuperAttack()', which simply outputs more damage to the player GameObject and plays a different animation.

Range Attack

I decided to implement a larger variety of attacks, hence the decision for the mage player GameObject. The mage attack is fundamentally a spell that fires out of the mage GameObject location, providing a form of long-distance attack that is shot at the enemy.

When the attack button is pressed when the mage is currently the selected GameObject from the 'SelectManager'. The MageSpell prefab will then instantiate at the location of the transform attackPoint. I utilised an Awake() to play the according to SFX, retrieve the current mage stats (the stats might be affected by other objects, hence it is dynamically located), as well provide the object with velocity. Providing the object velocity during the Awake() method call allows it to move, this ultimately allows the prefab to be shot, which in this case is in the direction the mage is facing.

Similarly to how the player melee attacks works, it uses collision detection through the OnTriggerEnter(Collider hitinfo), allowing the instantiate prefab to detect if it has hit the object with a collider. When the object collides with the enemy object it allows the corresponding operations to occur that are similar to the attack method calls.

I have also implemented an if statement in case the object is shot at a player GameObject, hence no operations should occur. I chose to make the prefab instantiate in a forward direction with velocity, as it provides more interactive gameplay in terms of having to manoeuvre the mage GameObject in the correct orientation instead of just simply having it act homing skill.

4.5.5 Player Skills

To further increase the individuality aspect of each player card and their GameObject I have implemented a special skill button for each that have their full fledge affect, including both an animation and SFXs. The skill button also follows the premise of having a cooldown, meaning that it cannot be spammed each turn. The skills are designed to be very unique for each given GameObject, allowing the whole premise of the video card game to more intuitive. An example of the current development is that the melee player character (Arthur_knight) boosts the attack and defence stats, whereas the range player character (Arthur_mage) has a heal function.

It would be over-ambitious yet again to be able to include a lot of skill-based effects, due to each one being unique, modifying the scripts for each individual one. They also require the player asset to have suitable Fx_Prefabs, hence the redundancy of time that would be involved.

Both the player and enemy object are the heart of the gameplay system, this being the visual augmented reality objects that are instantiate upon the real-world image targets. The system architecture for each object went through many iterations to ensure that they had a modular design, so they could be easily expanded upon in the future. A lot of the scripts were modified, due to my knowledge of Unity expanding such as the state machine behaviours through animations. I also took care in the methods I went about implementing the scripts, so that the foundation of scripts could easily be applied to the expansion of the database for both player and enemy cards. A large amount of time went into reducing the number of bugs between player and enemy GameObjects, due to the components that were implemented. A simple example would be when 2 objects collided the velocity of each would make them repel, flying across the screen.

4.6 NPC Object

The NPC object was a recommendation by my supervisor Balin Deng to provide another layer of gameplay mechanics. It was a suggestion that was implemented during the last week

of the video-game development, however, it went pretty smoothly due to it being the current prime of my Unity knowledge and understanding.

The NPC object is a GameObject that is instantiated upon the image target during the player battle phase. It uses a detection region similar to the ones implemented in the melee attack range to cause a switch case statement for a random effect to occur when the player enters the collision region. The effects are both buffs and de-buffs that are randomly chosen, thus providing another mechanic to the battle phase, as the user could be both rewarded or punished.

4.7 Audio Manager

The implementation of my audio manager is based upon the Unity scripting API: UnityEngine.Audio. The original method of adding audio within the Unity project is similar to the 'Main Menu' scene where an audio listener (usually on the camera) and audio sources are added to each GameObject. The 'Main Menu' scene is applicable for this method, as it only utilises one audio source and audio mixer. However, this causes a lot of redundancies as for the 'Game' scene, each separate GameObject would need an audio source added for the functionality I required, which is not efficient due to the sheer quantity of audio sources currently required. The inefficiency would just increase throughout the expansion of the video game where more audio sources are required. I devised a scheme that would solve this issue, which is shown on the UML diagram being the scripts 'Sound' and 'Audio Manager'.

4.7.1 Sound

The custom class Sound is derived from utilising the different attributes found within the Audio Source class, however, it has been streamlined and only includes the attributes I believe to be necessary. The included figure shows that some of these certain attributes required modifications such as including a [System.Serializable] and providing ranges. This utility allows it to be more user friendly within the inspector, allowing for modifications to occur such as simply changing the base volume of the audio clip.

Another implementation that is based upon the actual user interaction with the video game is the AudioMixerGroup attribute. This allows the developer to assign each different 'Sound' to a different AudioMixerGroup based on category (master, background and SFX), meaning that they can be changed within the GUI elements through sliders. This gives the user more control of the video game and is a feature that is normality in modern-day video games, hence the implementation.

4.7.2 Audio Manager Class

The audio manager script is the main handler for the custom class of Sounds, utilising an array that size can be set in the inspector, allowing for sounds to be easily added or removed. The sound class can then be easily modified based on the attributes included.

The method for calling the desired sound is now a lot more streamlined, as you just simply have to call the Play(string name) or alternatively Stop(string name). The name is a string attribute that is added when a new sound is added to the array, allowing this method to be easily called within other scripts (example of how to call). The method is also accountable for an easy typo mistake that might occur, which will now simply play nothing instead of breaking the video game.

The initial redundancy of the current method of adding audio within Unity has now been amended successfully so that it follows my more modular system architecture. It is now a GameObject that is more efficient for future expansion purposes because to further iterate, a new audio source can simply be added/removed from the array located within the AudioManager and then be easily called upon using the required method. Instead of Unity methodology of having to add a separate audio source as a component to each different GameObject and then calling upon the method.

4.8 COVID-19

The current circumstance in the world has as expected had an effect on the overall implementation of this project, due to COVID-19. It has affected the overall the whole implementation of the report through the current laws that have been set in motion by the government. An example of this is that the development of Unity was currently being developed on my main computer and I went home with my Apple MacBookPro, consequently due to the uncertainty of being able to transport and the risk of being in contact with my university housemates. I went a 1.5 week period of time where I was not able to physically work on my main computer, causing me to try to perform work at home trying to co-align with what has already been completed.

Overall COVID-19 has had a major effect on the implementation of the project. From the minor changes such as the meetings having to be performed online instead of in-person and the change in the working environment. The transition of the working environment and the daily occurrences that have to be adjusted to compensate for the current regulation all have had an effect in way one, altering my the mental state of this project subconsciously.

Chapter 5

Results and Evaluation

The overall system's success in terms of analysing the extent of me achieving my goals is based upon the specification and design. This can best be demonstrated through the overall walk-through of the gameplay. The gameplay will help visually aid the overall implementation process, providing more context in the different components listed within the UML diagram. As repeatedly stated, I do believe I have achieved the overall idea of this video game being modular in terms of the system architecture, the point has been fully supported during the implementation process, hence this segment will focus more on other key areas. The key areas are shown within the gameplay portion of the figure.

5.1 State System

The gameplay of the video game clearly shows how it iterates between different states within the system where different operations are performed in terms of requirements from the gameplay. GUI elements are displayed on the screen to help indicate the user, which also provide any required information. This main requirement is successful as the system enumerates between the main required states in terms of the BattleStates, but also the WaveStates from the level manager being able to cycle through the UML diagram until a certain requirement is met either the user winning or losing.

The gameplay from the state system follows the basic principles found in alternative video card games [19], however the defining point of the gameplay is not the state system, but what it does during each state. The overall gameplay is unique to my video game from the premises of how to win/lose and the player controls that occur during each stage, which I hope provides an intuitive experience.

5.2 Augmented Reality Detection

The augmented reality detection after scaling both the model and the attached billboard HUD works as intended, indicating the required GUI stats to help the user have a concept of progression through the battle. The models are also instantiated upon the assigned image target correctly, which is a fundamental requirement for the system, therefore it is a huge success. The gameplay image clearly shows a blend of the real world and the augmented models, enhancing the overall gameplay experience being able to successfully interact with the PlayerController. The models for the enemy are also a success, targeting a random player model, moving towards them, and performing the correct animations and operations.

I am confident of the overall implementation of the augmented reality system, due to it successfully fulfilling the main requirement. However, with any augmented reality system, some bug fixes were required to smooth out the system and gameplay. The first change is to only set the assigned model GameObject to active when the image target is found, removing some of the collision bugs, and targeting issues. Throughout the whole implementation, constantly checking to make sure the augmented models worked as required a lot of informal bug testing was performed. If it were possible to do user testing this could have found out, as more bugs within the augmented reality system and the entire system as a whole, because each user would approach scenarios differently. This is explained further in Initial Report - User testing.

5.3 Database

The database is the main goal that sadly could not be properly completed. This is mentioned in the implementation section, due to the limitation of finding appropriate models and the redundancy of time required to make complex individual systems versus similar mundane ones, i.e. the models currently in the database all have the same functionality with slightly different variables. From this point of view, the database was not successful. However as pointed out in my modular design, I do believe that the system works from a more comprehensible summary, as its major strength is the ease of ability to expand with more time.

5.4 Final Details

The final success from the current specification is the finer details within the gameplay that enhance the overall experience through both visual and audio cues. The gameplay has

extensive usage of visual cues through animation and SFX, increasing the overall experience compared to a static 3D object movement that is not natural. Both the animator and SFX are a huge success in the system, and overall were a smart move to implement during the final stages of development. The SFX are to help provide visual aids to the special effects, instantiating an object to help indicate when a special effect occurs, making each model feel more individualised.

The audio manager is another huge success, due to my implementation methods and operations that it performs. My implemented system is a lot more efficient than compared to the current Unity scripting API method of producing audio, therefore this is a goal that has been greatly achieved. Further tests to see how this performs would be increasing the overall array size to see if any implication occurs, as well as the overall performance when multiple audio sources are currently called at once. The current system may only have 3 audio cues being called at once.

5.5 Testing

5.5.1 Gameplay performance testing

The overall implementation of the different scripts has been done intentionally to reduce the overall stress in terms of computational power required. This is a key requirement to ensure that when deployed to a smartphone device that it can run smoothly, providing the intended gameplay. A method to try to adhere to this is that a lot of components could be checking for conditions through the use of the Update() function, which will, in turn, uses a lot more computational power. The UML diagram showcases that I have tried to limit the use of this method by instead using BattleStates and other conditions to trigger the required components. A lot of informal front-end testing occurred, making sure it performed as intended, reducing the overall number of bugs.

An improvement that could occur for the overall performance would be back-end through the Unity Performer (a feature I found out late in development), which provides real-time feedback on usage within both the CPU and GPU. If the feature was understood earlier in development I would be able to tell if the overall implementation of algorithm structure versus abusing the Update() method call has a true impact in performance through analytical data. The same data test could occur for future development through the expansion of the overall system architecture and volume of entities, safeguarding against any potential performance issues.

5.5.2 Initial Report - User testing

The initial report project aims and objectives overlap with the current ones mentioned within this final report. However, there is one key aim within the initial report that could not be adhered to, this is due to a combination of the unprecedented times and software limitations. A '... sense of fun' and 'A medium between ease of use for new users but have enough strategy to develop a skill gap.' The main feedback regarding the overall initial report is that time needed to be allocated for evaluation and user testing.

The premise of a game being 'fun' and having the dependency of it requiring skill is subjective. It is an opinion that would require user testing to receive feedback on areas of the gameplay they enjoyed and areas that are lacking, as well a consensus of if they believe the game is 'fun'. This is the main critical test that would be required for the overall success of the video card game, which can be further supported through beta testing that is common with other large game development companies. The feedback would allow changes to be made to better cater for the intended stakeholders. The changes could be minor to fixing some bugs - to major changes like re-configuring the whole gameplay mechanic to provide better ease for new users, but having enough strategy to develop a skill gap.

Unfortunately a combination of the stated Vuforia Engine 9.0 issue with iOS deployment, that is further reinforced by the fact getting an application deployed on the iOS app store requires a paid-for developer account, that also needs time to be passed through the iOS team before it reaches the app store [2].

The alternative option that would have been done and was my original method of resolving this could not be conducted. This was going to be anonymous user testing by allowing them to experience the gameplay through my computer, gathering qualitative data-sets that would help analyse the success of my gameplay. The appropriate ethical rules and procedures would be adhered to. The main alternative approach could not occur, due to COVID-19 restrictions, as this option would require users to be physically at my computer system, hence I was not able to gather a large enough volume of data to truly justify whether the gameplay was 'fun' and make the viable changes needed.

In hindsight evaluating my own work methodology in terms of time management; this issue may have been adverted if user testing was conducted before the increased prevalence of COVID-19, having most of the video game being developed before the said period. However ultimately, I did not worry about this aspect as there are more important events occurring at this moment in time and safety is the main priority.

5.6 Evaluate methodology and programming languages

5.6.1 Unity Game Engine

The Unity game engine is a vital instrument in terms of the overall performance of the project. The prior research ensured that Unity game engine was a justifiable decision, due to all the functionality meeting my system requirements. The main one is adhering to the augmented reality aspect of the gameplay and allowing for cross-platform development with multiple devices. Unity is a very intuitive game engine that is fundamental to this project, which I already had an idea it would perfectly be able to, due to AAA game titles being developed on Unity. Although I cannot definitively state that Unity is the best possible game engine for this video game, as I have not experienced any other game engines to provide a conflicting argument, outlining the strengths and weaknesses of each, the choice to use Unity was a success.

The Unity game engine community allowed the development of the database, due to the free assets available on the assets store, allowing simply queries to search for the packages required. I was, of course, limited by the fact that I had to acquire 'free assets', that meant hunting packages that filled my system requirements whereas a more commercialised video game could simply buy the assets required if their skill set did not meet the needs: 3D models, GUI artwork, sprites etc.

The Unity game engine has a magnitude of features that are readily available within the GUI, making it initially programmatic in the sense that it can overwhelm an inexperienced user who does not know where to start. I have already stated that creating a video game without prior knowledge to the game engine could have been a slightly over-ambitious aim. An example of this is that over time I developed knowledge on certain features that would make some implementation aspects more efficient or allow for better testing such as the Unity Profiler stated. However, I do believe that overall all it was a remarkable decision to use Unity game engine, due to it being a success for the overall project as a whole and the future development that could proceed.

5.6.2 C#

Is the programming language used for the scripts implemented for each of the different GameObject that provide the functionality to the video game. The experience of using previous programming languages such as Java and C made understanding C# very intuitive, consequently not limiting my overall performance in this decision. This is supplemented by the fact it is a high-level programming language that can be easily read, also it is intended

for cross-platform applications being desktop and mobile [17]. Many of the complex tasks are abstracted by the language, so when developing I did not have to worry about memory management or garbage collection. Learning a new language does have the basic issues involved, this can simply be through the syntax or the different approaches that were more optimised for C# coding.

The overall methodology I believe is therefore justified and success, as I wanted to use the programming language that is a normality for Unity development. This occurred with much success with the only main inefficient use of C# stemming from the niche method calls within the Unity game engine, deriving from MonoBehviour allowing for the following functions: Start(), Update(), FixedUpdate(), LateUpdate(), OnGUI(), OnDisable() and OnEnable(). The functions through repetitive use became readily used, utilising them as a part of my arsenal through the development process.

Chapter 6

Future Work

The future work area focuses on current implementation areas that do not currently fulfil my overall system aims and concepts. The section can be divided into main areas: the first one being the improvement of the current implementation method, finishing any areas that are not currently fully integrated in the manner I desire either due to time restraints or external factors such as Vuforia Engine 9.0 having software issues. The second area would be over the entirety of creating the video game the understanding of the development process, as well as new ideas have generated future concepts that I would want to implement.

I have personally got involved in this game project, envisioning a lot of ideas I would want to implement, as the execution is simply limited due to time and creativity. The overall development within Unity game engine is a clear reflection of Hofstadter's Law, due to the over ambitious aims of learning a new system engine and not completely accounting for areas that require more time.

6.1 Current Implementation

6.1.1 User Testing

If I had more time I would have been able to conduct my user testing, as well as being able to obey the current laws that have been set in place, due to COVID-19. The first area this project would need to cover is extensive user-testing, allowing game testers to play the current state of the video game, so I would be able to gather analytical feedback. The qualitative judgement of these game testers would be the key area of future work, allowing me to cycle through the game's creation by implementing features recommended by the stakeholders. This would provide a quantitative means of measuring the judgement of if this video game is 'fun', which I believe is the main key area to focus on.

The overall user testing would also help towards bug fixes and performance issues that may occur. Having multiple users approach the overall gameplay mechanics at once, the bigger volume could draw more attention to this potential problem. It would also give me a better indicator if any areas of the came are too intensive for the user's device, hence the implementation of the scripts can be best optimised if required. Sentence feels a bit incomplete if you had more time what?

6.1.2 More Levels

From the initial user testing, I would be able to design more enemy GameObject and levels that are enjoyable based on the user's feedback. The current concept only features 1 proof of concept level, but with more time and the access to more unique 3D models, this can easily be expanded, due to the current implementation methodology. The levels would be built upon the current mechanics through the gameplay, adding upon the PvE aspect of the game. I would also if I had more time to overhaul the entire PvE system, provide a tutorial and some form of story aspect, utilising the said levels. The levels would also be able to provide more feedback to the current implementation and what changes need to be made. An idea for this would be a point system, in-game currency, experience etc.

6.2 Future Game Development

6.2.1 Artistic Implementation

The next aspect of the video game would be making a fully-fledged game, deploying it on the iOS store or the Android app store. This would allow the video game to be classified as an indie-game, which most importantly makes it accessible for other users with the passion for card games. This would inherently allow for more user testing, due to the nature of both app stores providing user reviews.

The ability to deploy it fully to a mobile device would require some minor changes, which firstly, would be dependent on the bugs being fixed within Vuforia Engine 9.0 package by its development team. The second one would be making the screen overlay more dynamic, allowing the different GUI elements to compensate for the multitude of the different resolution displays of modern smartphones, as the current development is built for portrait orientation on an iPhone X screen (1125x2436).

6.2.2 Team Expansion

For the overall development of the gameplay, it would require the sourcing of graphics artists that skill set is suitable for GUI designs and 3D GameObject models, creating the animations that go along. The sourcing of music and sound effects are also required. All the current assets are simply royalty-free, which have been mixed and matched together to help demonstrate the purpose. The expansion of the team and acquiring the needed skill sets would help towards the overall commercialisation of the game, due to it proving the video game with a unique design overall in terms of aesthetics. This is key in modern-day video games, making it distinguishable from competitors and not infringing any copyright laws.

6.2.3 Database

The expansion of designers would largely be involved in the database of all the cards and all the information that is retained. The image targets (being the cards) could be designed in a unique manner like the current physical cards (reference), providing information that can be read. This would help with the blend of the physical interaction of real-world cards, which is enhanced by the GUI interface. The cards information and overall design in terms of characteristics, abilities, movement speed through the different attributes. This then can be correlated to a 3D GameObject model with the animation and SFX needed through the current components and new scripts that would need to be implemented.

This is a current limitation of the present concept due to the database volume being minimal, but as mentioned in the implementation, it can be expanded easily. The different 3D GameObjects and models with time would allow the overall expansion of the database, which could change the whole dynamics of the gameplay from its initial implementation. The user feedback would help design the cards in a way that best suits the intended stakeholders, adding new cards to balance ones that are currently overpowered. The overall implementation of the database would allow card decks to be formed, being unique to each individual user, limiting them on the number of cards they can have in each deck. The card decks were not implemented currently, as it requires a login system and a larger database. This would then provide a unique gameplay aspect, as they would need to pick the correct combination of cards to approach each scenario either in PvE or PvP subjectively, providing a more strategic game

6.2.4 Multiplayer

After the development of the PvE aspect of the card game; the next aspect as stated would be PvP. This was an original aim for the gameplay, however, this was too ambitious because of my lack of Unity knowledge, as well as hardware limitations as 2 devices would be required.

Login

The first aspect would be a basic login system, allowing the user data to be stored and to be able to add other users based on their unique ID, as well as making the account unique to their instance. This would be used to allow a save state within the user's instance of the video game.

The login system would need to be developed with great care to adhere to security issues and how the data is manipulated, stored, and retained [25]. The precision of the development of the login system would need to be fully explored, limiting the amount of information required, which would need to be in balance with the aspects required in terms of the functionality. A simple example of the login system would be a basic sign-up form, collecting data needed, but being aware of the privacy concerns and the ethical method of doing so.

User instance

Each individual user instance would allow the individuality of each user's gameplay experience. It allows them to have a unique experience where their current levels are stored, their progression as well as the unique cards assigned to their account.

To further iterate on this point, I have been so passionate about the development and evolution of the game I have thought of ideas based on how the database would be unique to each individual user. Even after this project has been officially submitted; I will be spending my free time developing the video game I have created. Each individual user would have unique cards assigned to their account, which can be unlocked through gameplay or the integration of in-game transactions. The cards they have unlocked would be linked to their user account data from the login system, allowing them to print off the cards at home, providing them with the necessary image target. If they tried to augment a card that they do not have access to as it is not unlocked on their account then nothing would occur.

6.2.5 PVP

From the unique cards that they have unlocked and the creation of their independent card decks. The user would then like with most video card games (reference) be able to play against their friends and other human beings through a PvP aspect. A gameplay aspect that would provide a whole new dynamic within the game, which could then be furthered elevated in a more competitive aspect through the balancing of cards and a leader board system. The PvP system would take normality found in modern-day PvP systems.

The implementation of the PvP system can occur through Unity, however, it would be necessary for the game to require internet access, therefore the development would need to be aware of the security issues that can arise. This would also require the user's permission, which can all be implemented in the method of a sign-up form including the terms and conditions. This is all ethical information that needs to be formally generated properly. In Unity, some form of network manager would need to be implemented to allow the transmission of data between each user's instance, so that they can smoothly interact with each other. Further research is needed for this overall concept to be developed. The gameplay aspect would be like the current PvE aspect, allowing the interaction of augmented targets, which are then controlled. The user would need to be in proximity like a real-world card game, however, the augmented objects would be instantiated on both user's screens.

Chapter 7

Conclusions

The key aim of this project was to help a childhood vision come to reality, making a card game where models would instantiate in the real-world. The overall objective as defined is creating a working augmented reality card game, showing proof of a concept that could be expanded upon. The video game has the inherent concept of it being a card game, fulfilling those conditions. It is built on the foundation of being able to expand, so that more cards can be added to the video game, developing the database, models, levels, and other features.

I used an individualised Sprint Agile scheme to develop the project, using the Unity game engine, implementing packages such as Vuforia Engine and an abundance of components with the use of scripting API, as well as using tools from the asset store that are beyond my skill-set. This development process took up most of the time. In hindsight, if I had a better understanding of Unity, having a clearer path and direction towards the whole overall system would be built. I would have used a more formal approach like TDD, which would also account for user testing.

I was sadly not able to cater to an important goal of the gameplay being ‘fun’, due to the reasons outlined through the report. I do believe that instead, I have implemented the game in a modular fashion that would allow an ethical procedure of user testing to easily occur when appropriate to do so when the external limitations are removed. I believe I have used my time frame appropriately, as areas, where I could not do as originally intended, have been negated by the finer details that were not a part of the original specification. I made sure that I could constantly adhere to the overall development of the video game, striving for it to be better.

With more development time and having a larger development team in term of skill-sets from an artistic creation or learning them myself, I would be able to fully deploy the project. Like with any video game I would be able to increase the overall functionality of the game, expanding the database, altering the gameplay, constantly bug testing. The whole video game

is to make something I am proud of and enjoy. In doing so I hope other stakeholders enjoy it as well.

Overall, considering the project has had a lot of uncertainties during development and in being developed through an unprecedented time, as well as using a game engine for the first time that is built upon a foreign programming language being C#, I would say this project is an overall success. I have successfully created a working augmented reality card-game that shows a proof of concept, but more importantly allows for the most important factor of video game development, catering to the consumer by constantly improving. The video game can be constantly evolved, having no limitation in overall development through a cycle of user testing and development.

Chapter 8

Reflection

Throughout this project, I have developed a lot of explicit and tacit skills that reflect this module being the sheer pinnacle of my course being Computer Science (BSc). I have learnt a diverse number and range of skills, techniques and implementation approach through the guidance of my supervisor, but more importantly the independence of handling such a large project. More significantly, the tacit knowledge gained regarding me as an individual and how I handle the pressures of developing such a large project will be of more importance in the future and the projects that will follow. It has shaped my mannerisms in understanding how to better approach a project and fully understanding that I should take these life experience constructively. It provides insight into my future career path after I leave Cardiff University.

Firstly, it has provided insight on my ability to learn a complex skill-set like the Unity game engine and new programming language C# in a short period, having an idea on how I may approach future learning differently. This applies to the niche skills that I may have to grasp in my future career paths. I understand that I need to allocate more timing for the overall general learning and that the project development may not go as intended, due to my knowledge, the complexity or time limitations. I would also be more prepared in the future so that if I have prior knowledge of the unique software needed, I can build a stronger foundation before starting.

Secondly, even though I used an approach of Sprints during this method I am aware of its limitations. The project system architecture was based upon the needs of fulfilling the system aims and goals, however, it was not blueprinted as neatly as in the appendixes provided. This approach worked due to this project being an individual one, however, this may not work in future team projects. I believe a more concrete layout of time management should have occurred, following stricter guidelines such as following my initial report's time frame to a better a degree. I do believe in following a more formal approach, but in unison with my approach of not being anxious in the development project not having to follow the weekly

guidelines word for word. This allows problems to better be accounted for, providing more time to areas that may have not run as smoothly.

Regarding the report, I believe that I should have allocated more time to it along with creating the supplementary material throughout the development process so that it conveys more closely to my thought process at the given time. This could have been documented where all the references were added whilst they were being used or simply logging my overall development process. I believe due to the sheer nature of learning a new game engine and the problems involved, as well as the fact this project is a video game, meaning that the development of builds can be considered endless meant the report was started too late. In hindsight I should have allocated more time for the report, ensuring that it is a better reflection of the workload done during this semester. I have also learnt a great deal regarding a more professional typesetting system, being LaTeXto generate reports. It was difficult to understand the design features when creating a project of this size, but ultimately it has broadened my skill set through spending the time to formally lay out the information in a design scheme I am proud of.

Finally, the overall unique experience of handling external factors such as COVID-19, which even though may have affected the outcome negatively, I believe will help me in my future endeavours. Ultimately, I do believe that my overall project may have been to a better standard then I would have hoped. This experience has helped mould me into a person where my mindset is that I can still adhere to my main goals and aims, however, I can also recognise that the journey there may not be as smooth and that compromises might occur. What we do during these hardships and having the ability to still perform at a high standard is most important to me, to learn from these experiences, applying them in the future endeavours. I do believe that overall, I am grateful to have experienced a project of this magnitude, as it allows to me have a better grasp of my strengths and weaknesses, developing me as an individual.

Glossary

Agile A project management approach based on delivering requirements iteratively and incrementally throughout the life cycle. [1]

Agile Development An umbrella term specifically for iterative software development methodologies. Popular methods include Scrum, Lean, DSDM and eXtreme Programming (XP). [1]

Augmented Reality is the blending of interactive digital elements – like dazzling visual overlays, buzzy haptic feedback, or other sensory projections – into our real-world environments. [20]

Monobehaviour Is the base class from which every Unity script derives. When you use C#, you must explicitly derive from MonoBehaviour. [10]

Prefab Is a system that allows you to create, configure, and store a GameObject complete with all its components, property values, and child GameObjects as a reusable Asset. The Prefab Asset acts as a template from which you can create new Prefab instances in the Scene. [5]

Scriptable Object Is a data container that you can use to save large amounts of data, independent of class instances. One of the main use cases for ScriptableObjects is to reduce your Project's memory usage by avoiding copies of values. [6]

Sprints A short development phase within a larger project defined by available time ('timeboxes') and resources. [1]

Virtual Reality Is using computer technology to create a simulated, three-dimensional world that a user can manipulate and explore while feeling as if he were in that world. [29]

Yu-Gi-Oh! Is an exciting universe based on a card game played with Monsters, Spells, and Traps. The Yu-Gi-Oh! franchise includes manga series, television series, several video games, the Yu-Gi-Oh! TRADING CARD GAME, and more! [15]

Monobehaviour common methods

The unique methods that are widely used within C# development, however may not be common to a standard user.
Awake() Is used to initialise any variables or game state before the game starts. Awake is called only once during the lifetime of the script instance. [14]

Start() Like the Awake function, Start is called exactly once in the lifetime of the script. However, Awake is called when the script object is initialised, regardless of whether or not the script is enabled. Start may not be called on the same frame as Awake if the script is not enabled at initialisation time. [12]

Update() An update is called every frame, if the MonoBehaviour is enabled. [11]

FixedUpdate() Has the frequency of the physics system; it is called every fixed frame-rate frame. [13]

LateUpdate() LateUpdate is called after all Update functions have been called. [9]

OnEnable() This function is called when the object becomes enabled and active. [8]

OnDisable() This function is called when the behaviour becomes disabled. This is also called when the object is destroyed and can be used for any cleanup code. [7]

Table of Abbreviations

Acronyms

AAA All About Appearance

AR Augmented Reality

FBX Filmbox

GUI Graphical user interface

IDE Integrated development environment

PvE Player versus Environment

PvP Player versus Player

R&D Research and Development

SFX Special effects

UML Unified Modeling Language

The initial sketches for the Game

Appendix A contains the initial sketches are based upon the specification. I chose to manually draw some basic designs, so I would be able to visually have a better understanding on how I wanted each section to perform.

The initial sketches for the Game

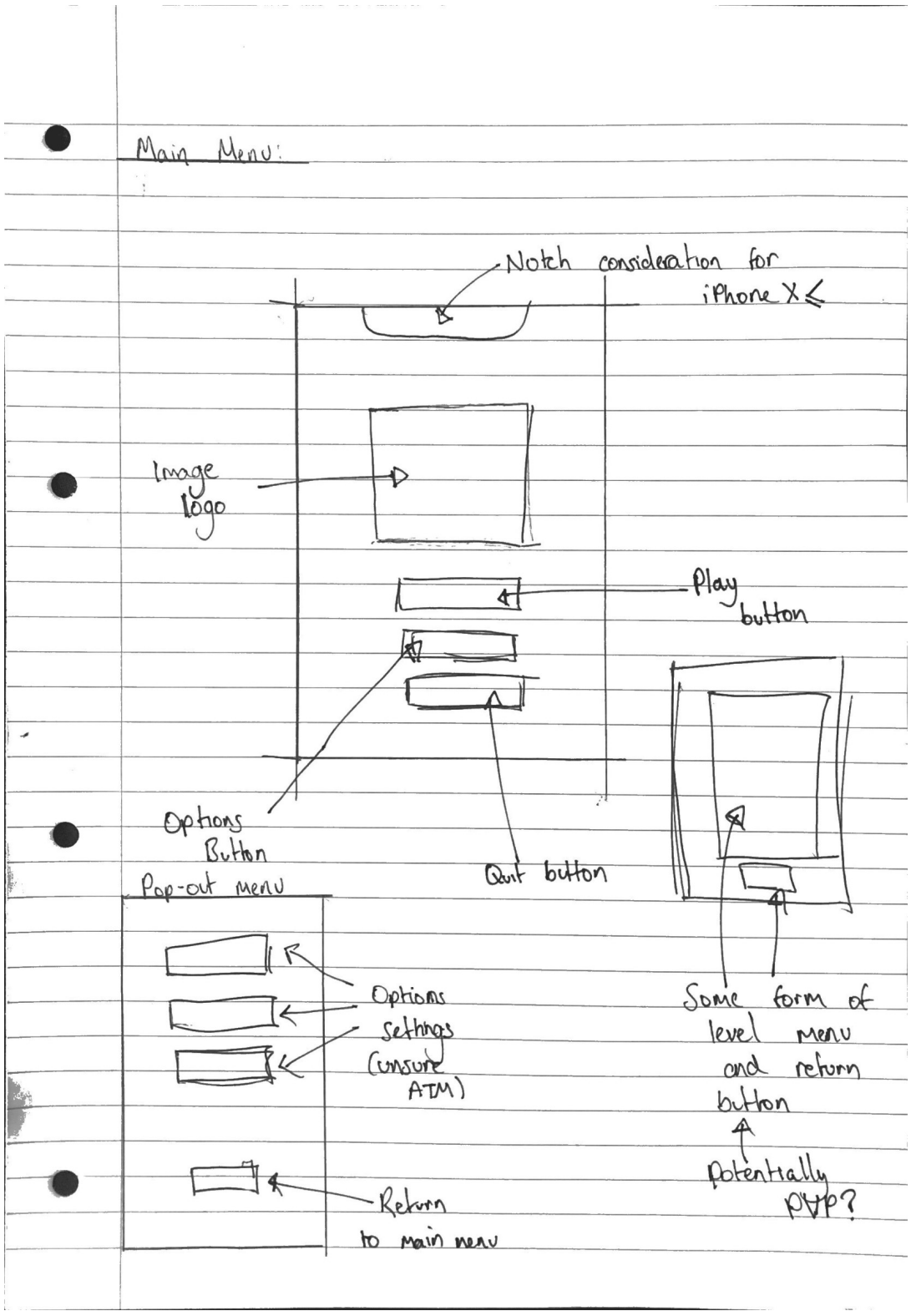

Fig. 1 The initial designs for the main menu

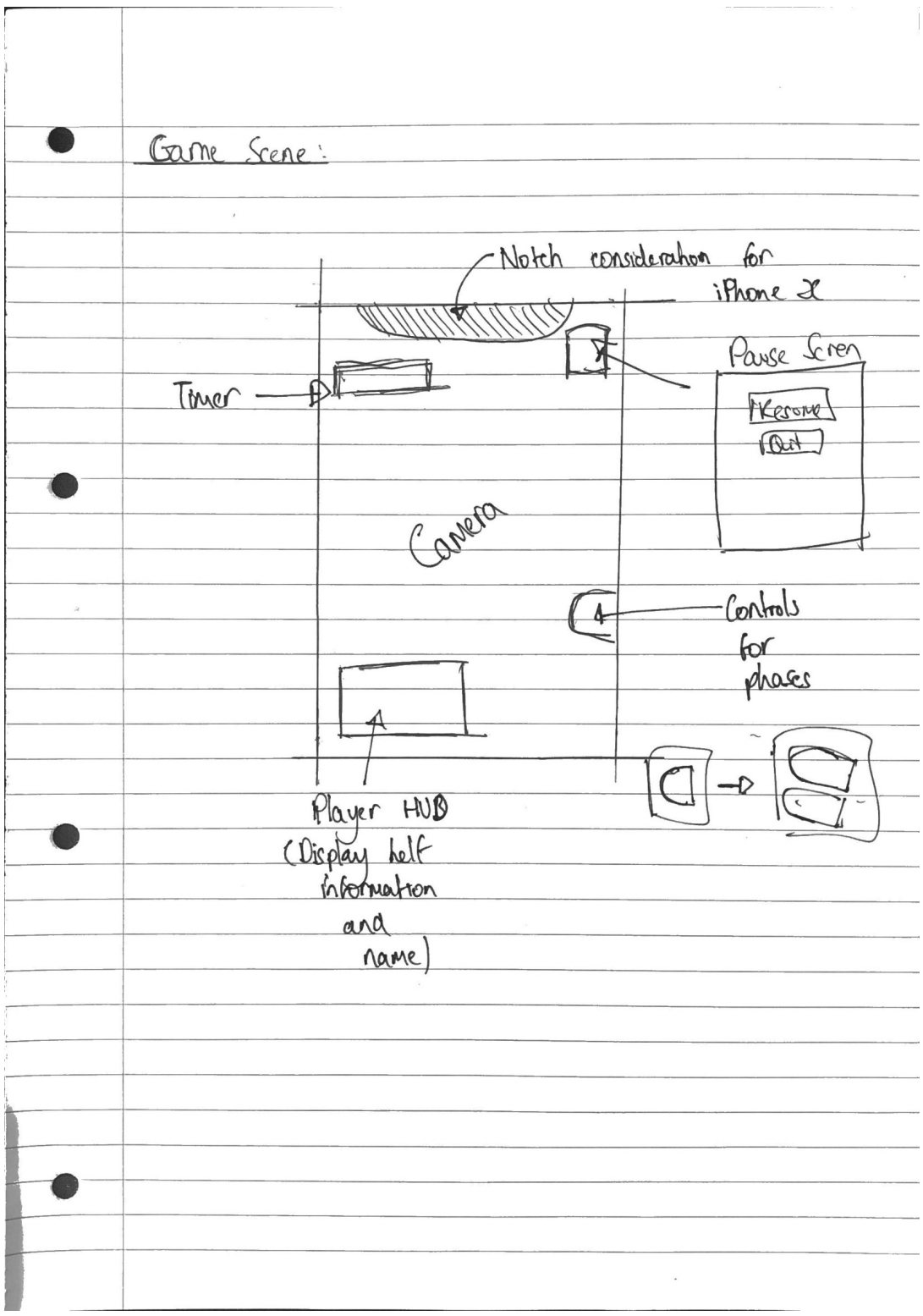

Fig. 2 The initial designs for the game scene

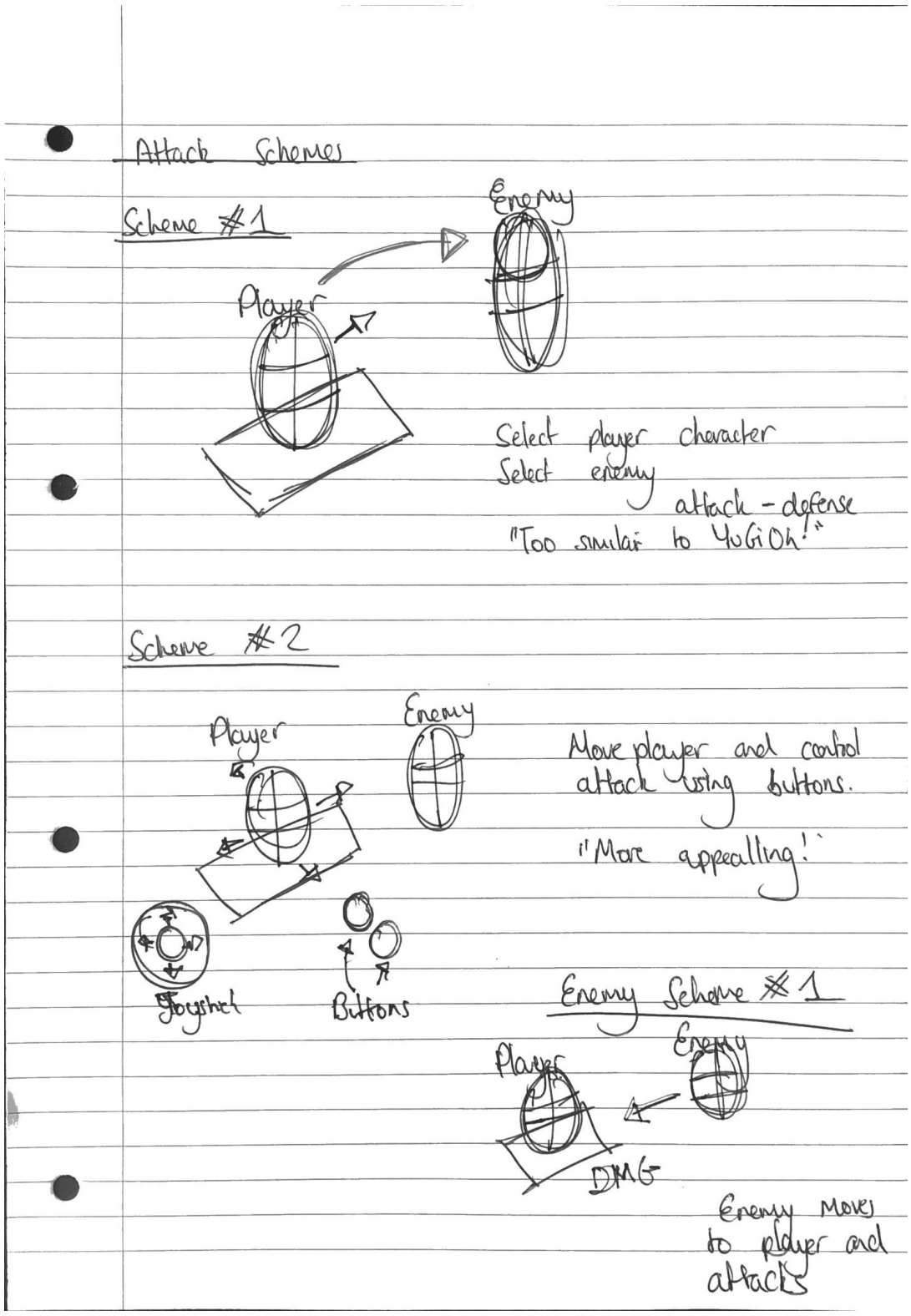

Fig. 3 The initial designs for the battle system

UML Diagrams

Appendix B show the 3 UML diagrams to help show how the different components within the game engine interact with each other. It shows how the different scripts interact for both the 'Main Menu' and 'Game' scene, showing the class relationships. The 'Game' scene features both a basic level, showing the initial components stated within Game Architecture and the more developed advanced level from what was required from the implementation. (Note: I have also uploaded the UML diagrams with this report, due to the limitations of the size, hence it should be treated as support documentation.)

UML Diagrams

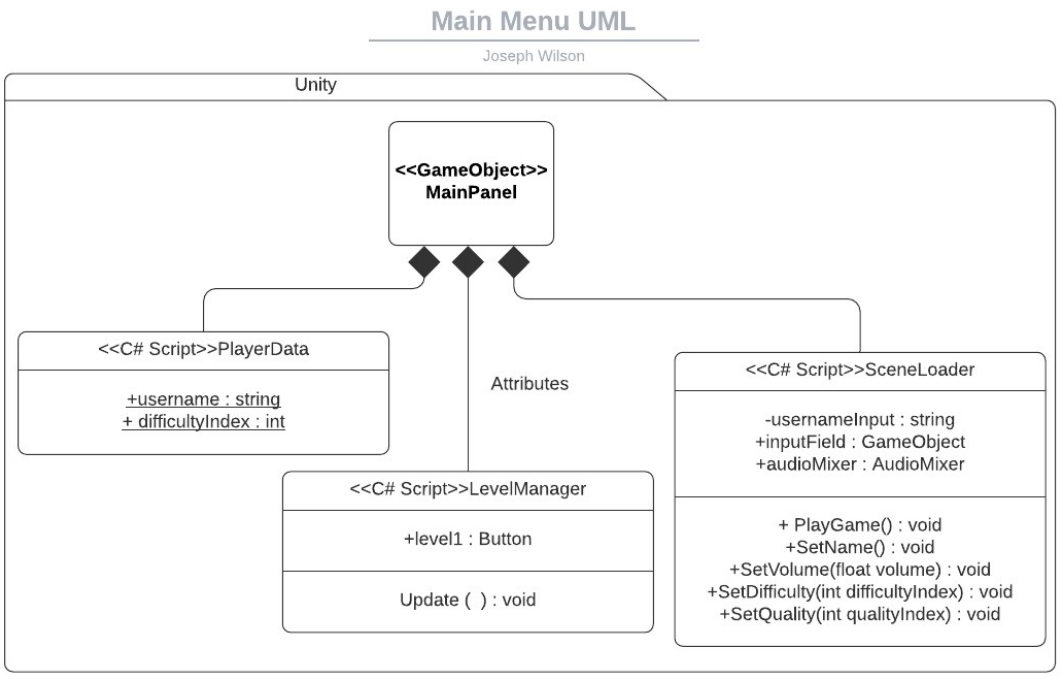

Fig. 4 This showcases the interactions between the components in the ’Main Menu’ scene

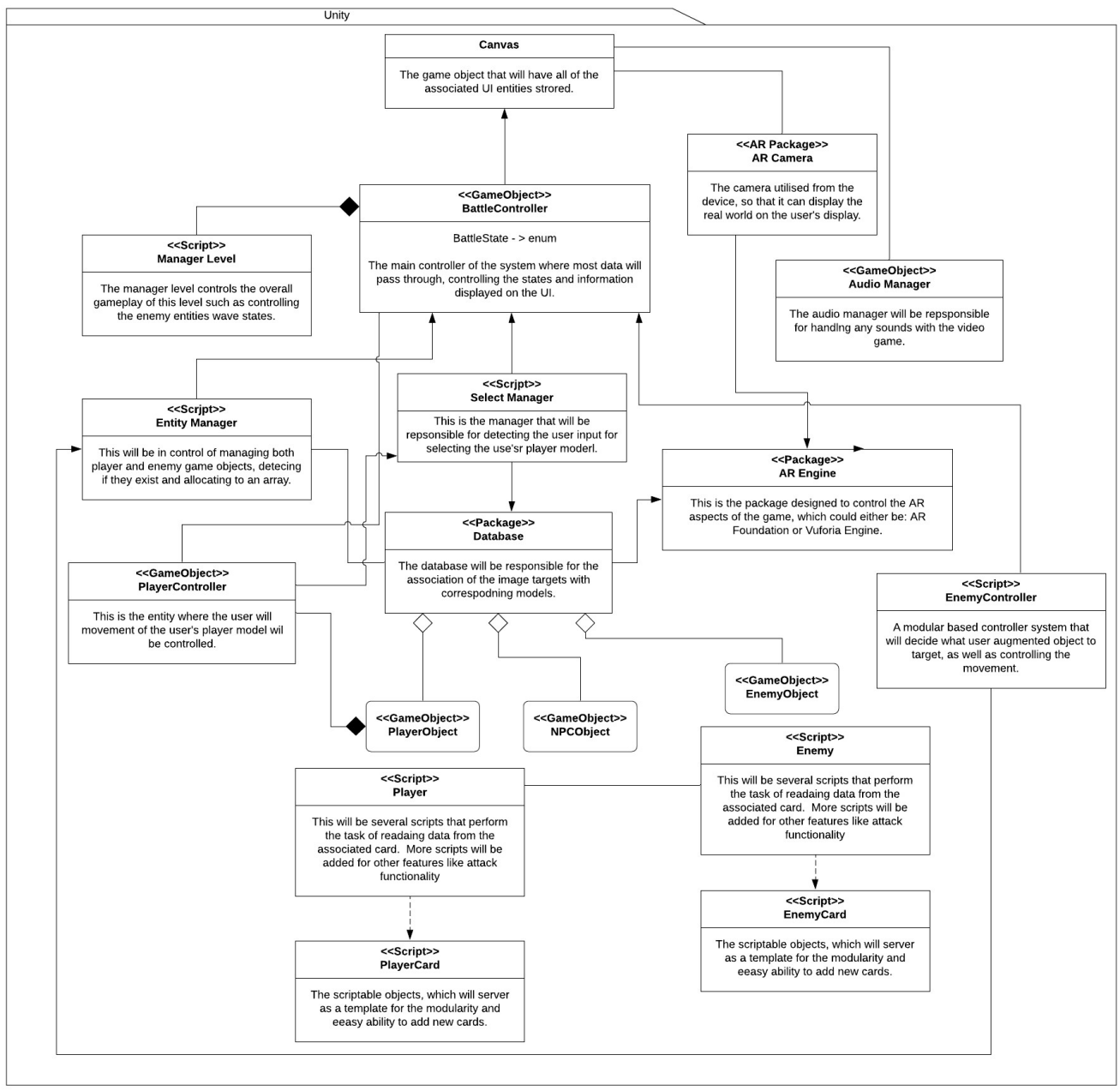

Fig. 5 This showcases the interactions between the components in the ’Game’ scene based from the initial system architecture

UML Diagrams

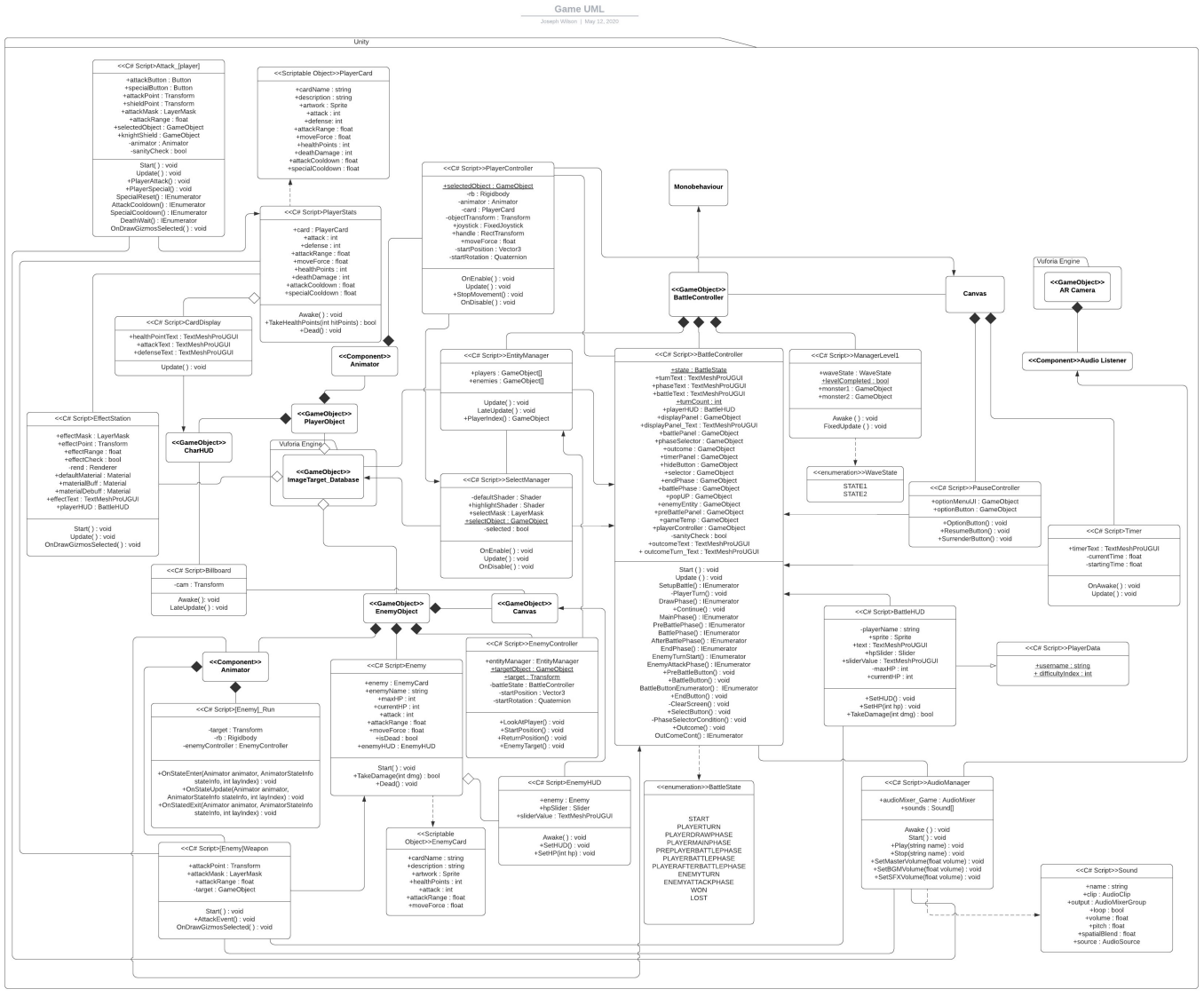

Fig. 6 This showcases the interactions between the components in the ’Game’ scene after the implementation process

An overview of the gameplay

Appendix C contains the gameplay, which is shown using the Unity game view within the game engine. It show firstly the main menu scenes. Then after the gameplay that occurs once you use pick a level from the main menu. The game view window is set in the resolution designed for a portrait orientation of an iPhone X/XS. The Unity game engine has a magnitude of presets in terms of resolution to accommodate different iOS devices.

An overview of the gameplay

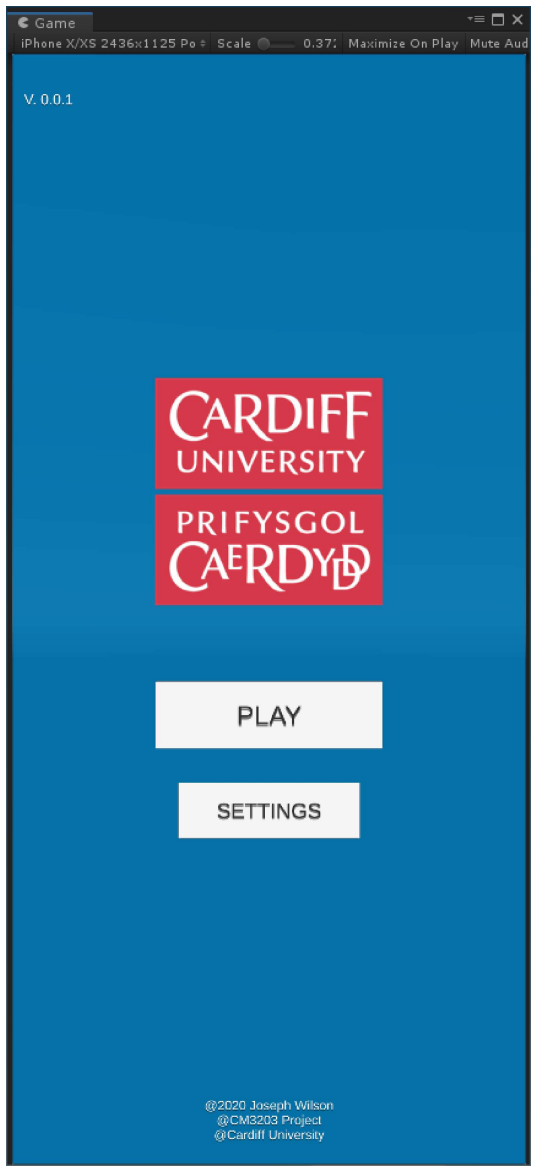

Fig. 7 The main menu - starting screen

The initial main menu screen that is presented to the user after the splash loading screen occurs.

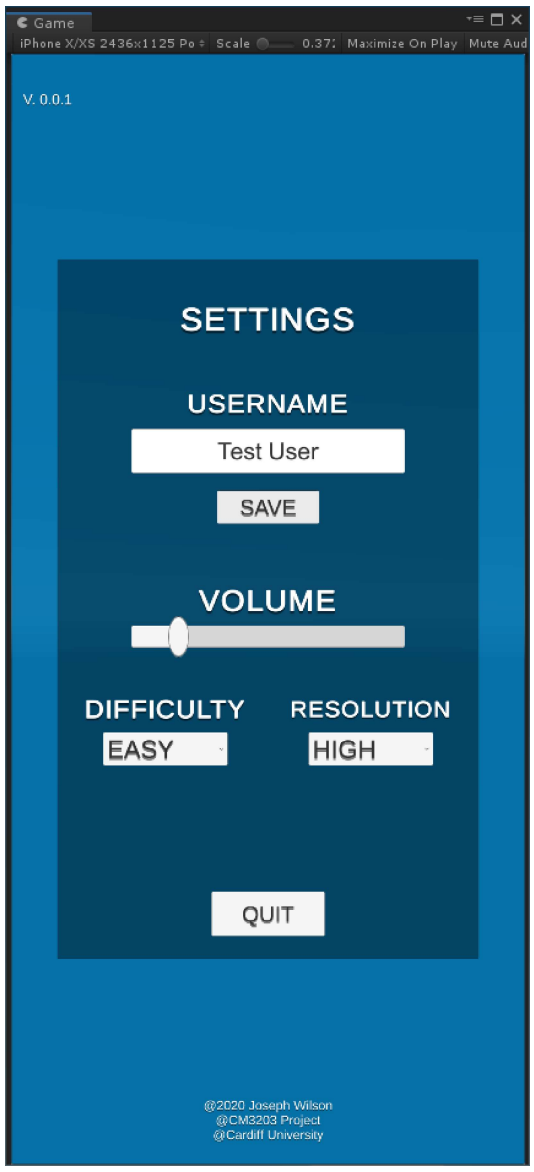

Fig. 8 The main menu - settings screen

The settings menu allows the user to select the various options that are available to manipulate the in game settings. A username is required before they can play a level.

An overview of the gameplay

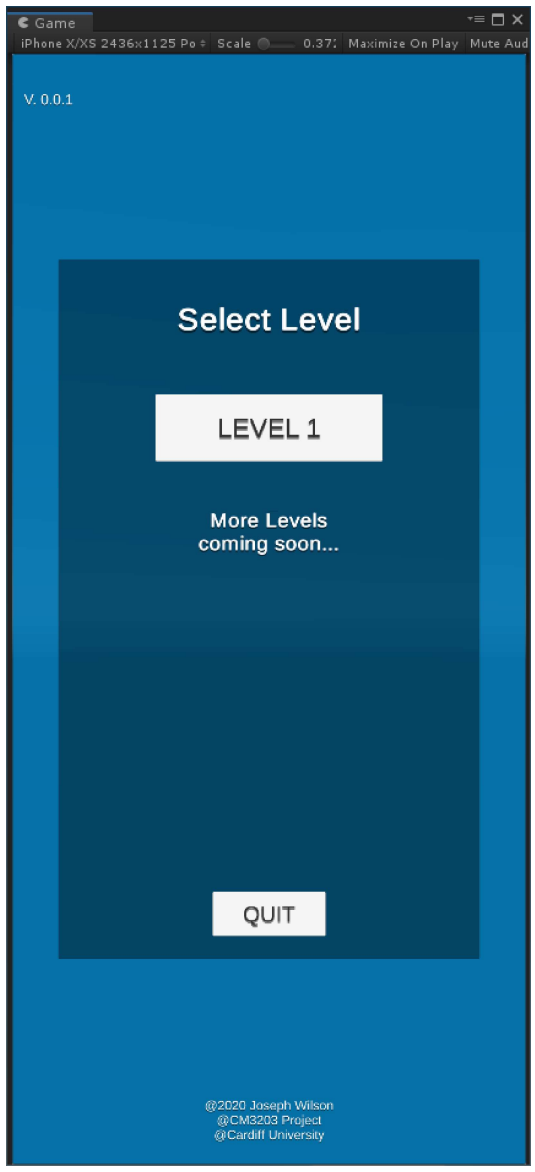

Fig. 9 The main menu - level screen

The level menu showcases the different levels the user is able to select.

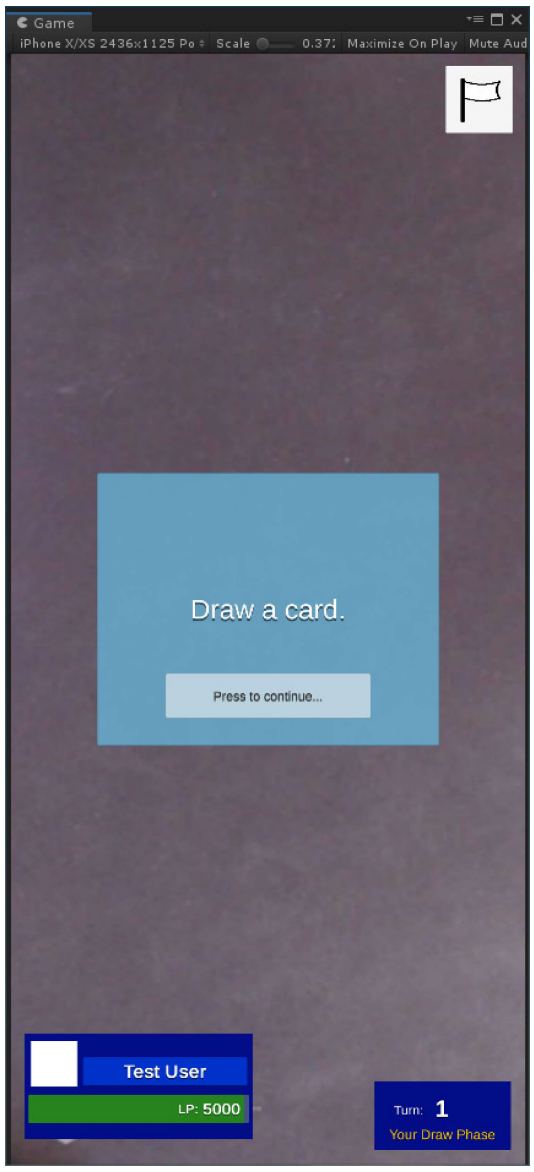

Fig. 10 The game scene - draw indicator

After the initial state of DUEL appears. The game transitions to the first interactive screen where the user draws a card (getting prepared to play). The display HUD clearly shows this, as well as the turn and current battle state. The difficulty is based on the main menu settings, which affect the overall HP of the player and strength of the enemy entities.

An overview of the gameplay

Fig. 11 The game scene - player main phase

The main menu is where the player places their image target to be instantiated, as well as other NPC objects. It also allows them to transition to different scenes through the settings wheel. The settings wheel is orientated to adhere for a user's thumb placement.

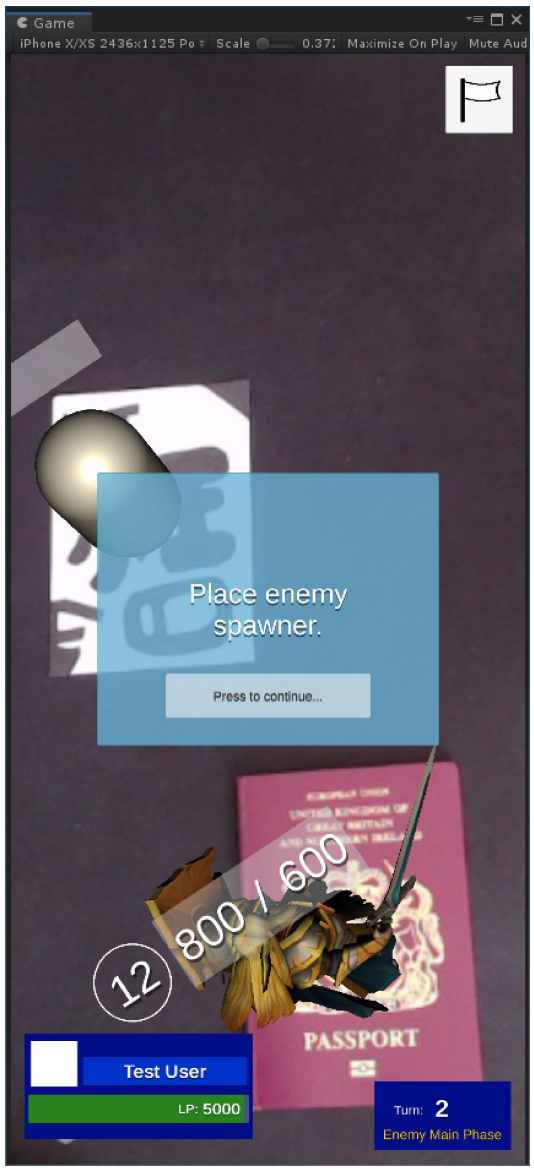

Fig. 12 The game scene - enemy main phase (Wave 1)

The game has transitioned to the enemy turn through the user's settings wheel. An indicator to place the enemy 'spawner' object for PvE has appeared. The settings wheel is orientated to adhere for a user's thumb placement.

An overview of the gameplay

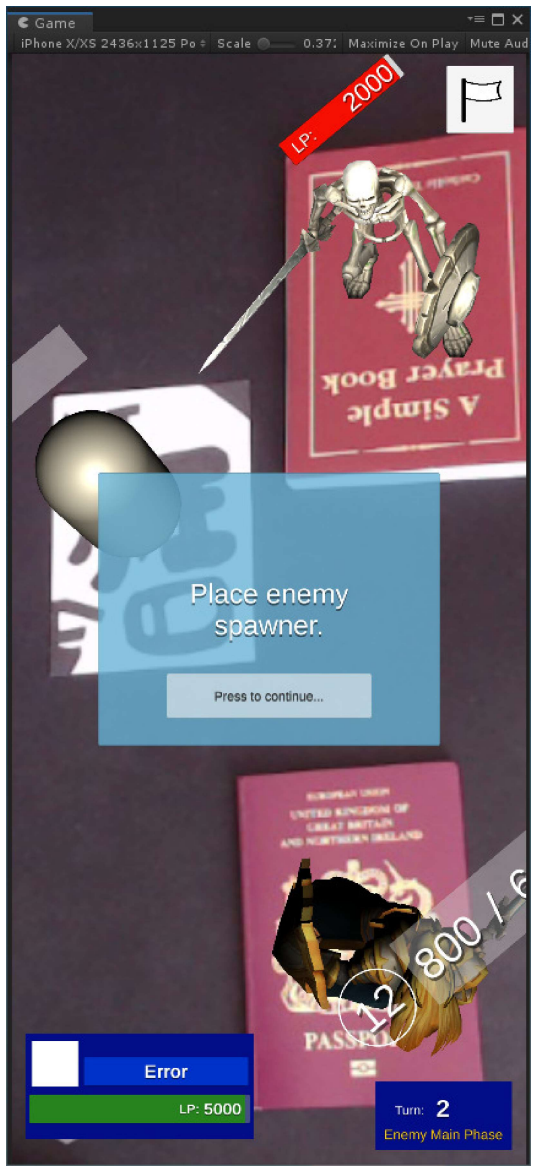

Fig. 13 The game scene - enemy main phase cont.

The enemy game object instantiated, showing its current HP. Both the player and enemy use a billboard script for the HUD, ensuring the information is always displayed facing the user.

Fig. 14 The game scene - enemy attack phase

The enemy object moves towards the player target and attacks the object, performing the assigned operations.

An overview of the gameplay

Fig. 15 The game scene - second main phase

An example of the next main phase where the user may instantiate another image target. The objects look the same, however, they are both different as shown through their stats. Further detail of what happens when you press the settings wheel, showing the various options that pop up.

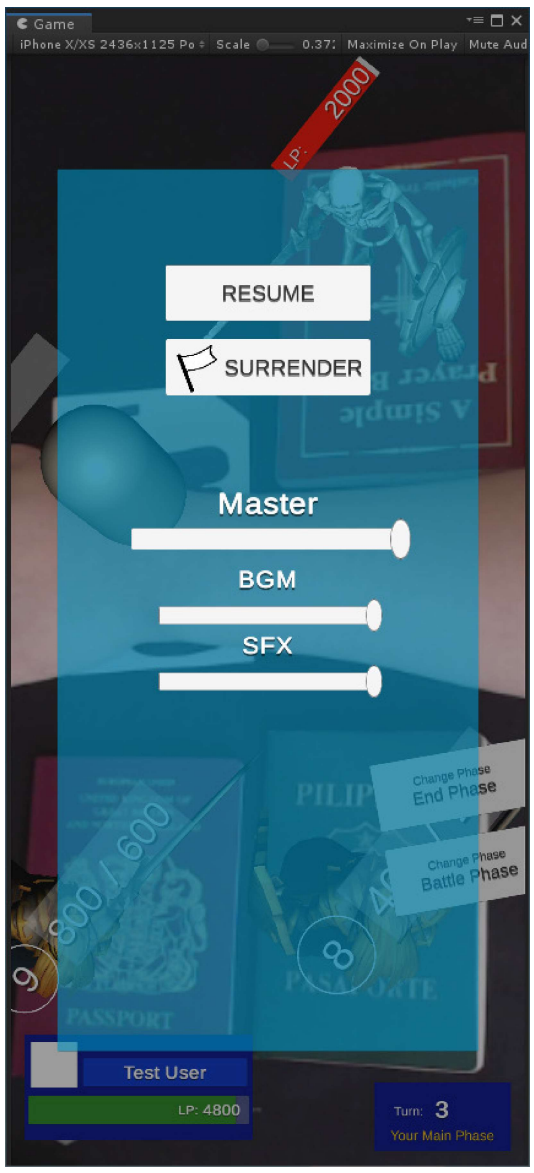

Fig. 16 The game scene - pause menu

When the surrender flag is pressed it acts as a pause menu, freezing time for all the enumerations and game flow. The different options are also present to allow the user to manipulate the presented settings.

An overview of the gameplay

Fig. 17 The game scene - player select phase

The pre-battle phase where the select manager scripts occur. The user is able to select the object they want to control (you can see an orange outline on the left entity).

Fig. 18 The game scene - player battle phase

The player controller and button appear, and you can see the player has now moved to the enemy object.

An overview of the gameplay

Fig. 19 The game scene - player battle phase cont.

The user clicks on one of the buttons and the attack has occurred, reducing the enemy object HP.

Fig. 20 The game scene - player battle phase (effect)

Alternatively, the user may move to the NPC GameObject. It chooses a random effect, which is indicated through the nameplate above and the colour. It this example a positive effect has occurred.

An overview of the gameplay

Fig. 21 The game scene - Wave 2

After the first enemy within the wave is dead the second one spawns. You can see the new stronger creature move towards the player entity and has reduced more of the HP.

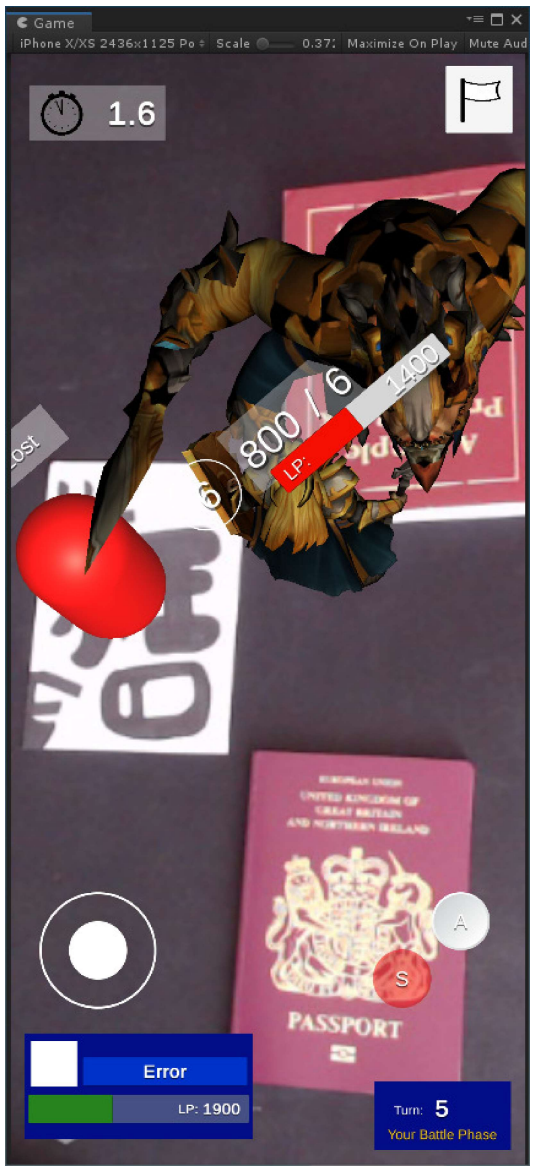

Fig. 22 The game scene - more examples of gameplay

A screen of the gameplay showing more information. This displays more information regarding the 10 seconds timer the user has to controller their object. It also shows the negative effect of moving towards the NPC object, the reduction of enemy model HP through attacks, the cooldown of using an attack/special, which is indicated through the button being red.

An overview of the gameplay

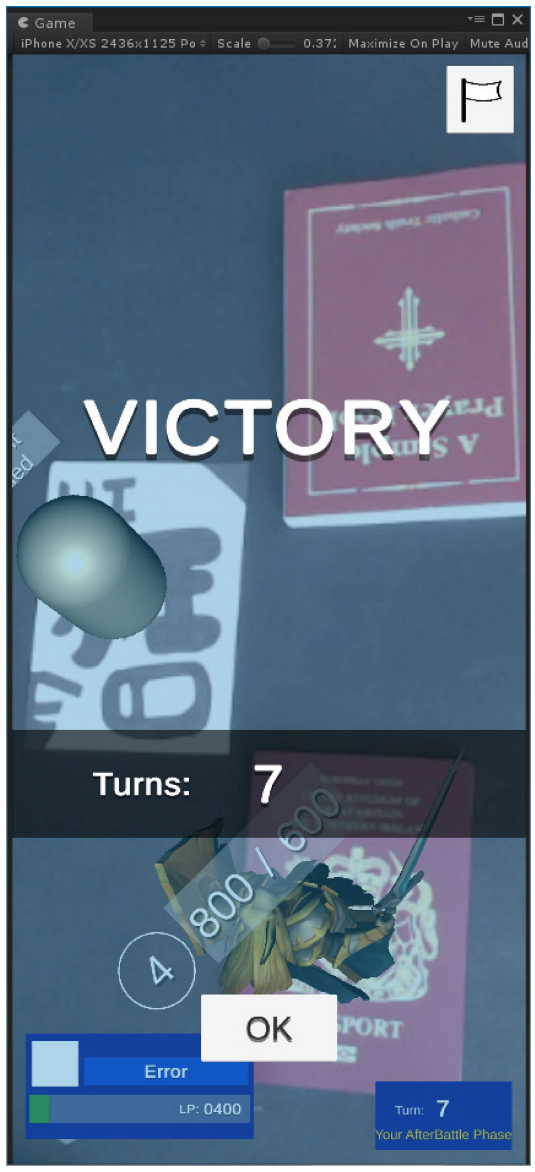

Fig. 23 The game scene - Victory menu

The state screen of declaring the user has won when all enemy entities have been destroyed. The background audio will change.

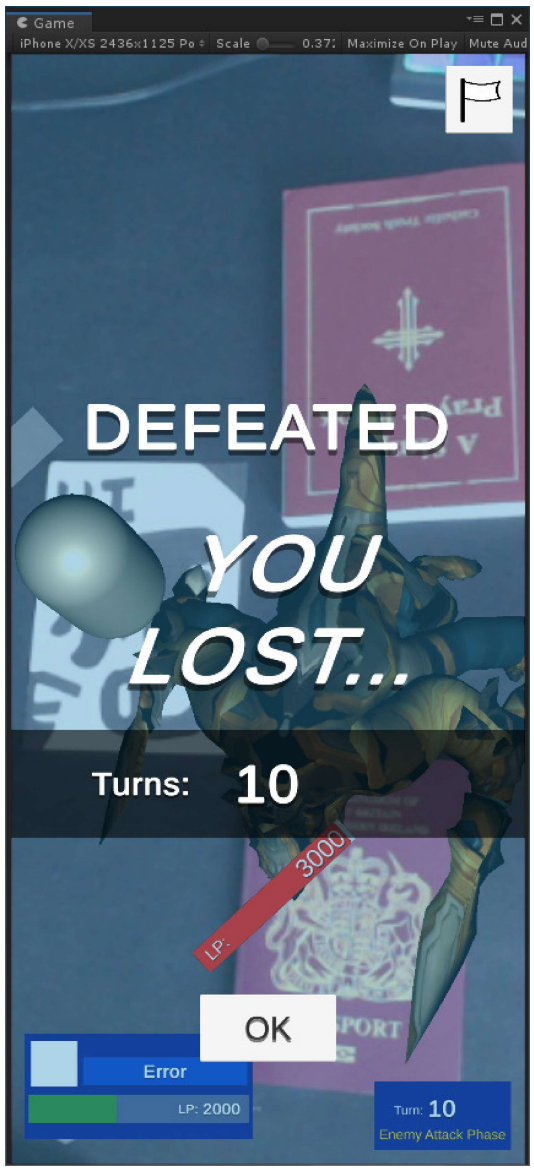

Fig. 24 The game scene - Defeat menu

The state screen of declaring the user has lost when all player entities have been destroyed or when the HP reaches 0 or. The background audio will change.

Supporting Code & Other Material

Appendix D contains supporting material regarding the code used within AR Foundation (this code is no longer apart of the current build). It is also used to highlight any unique Unity features.

Supporting Code & Other Material

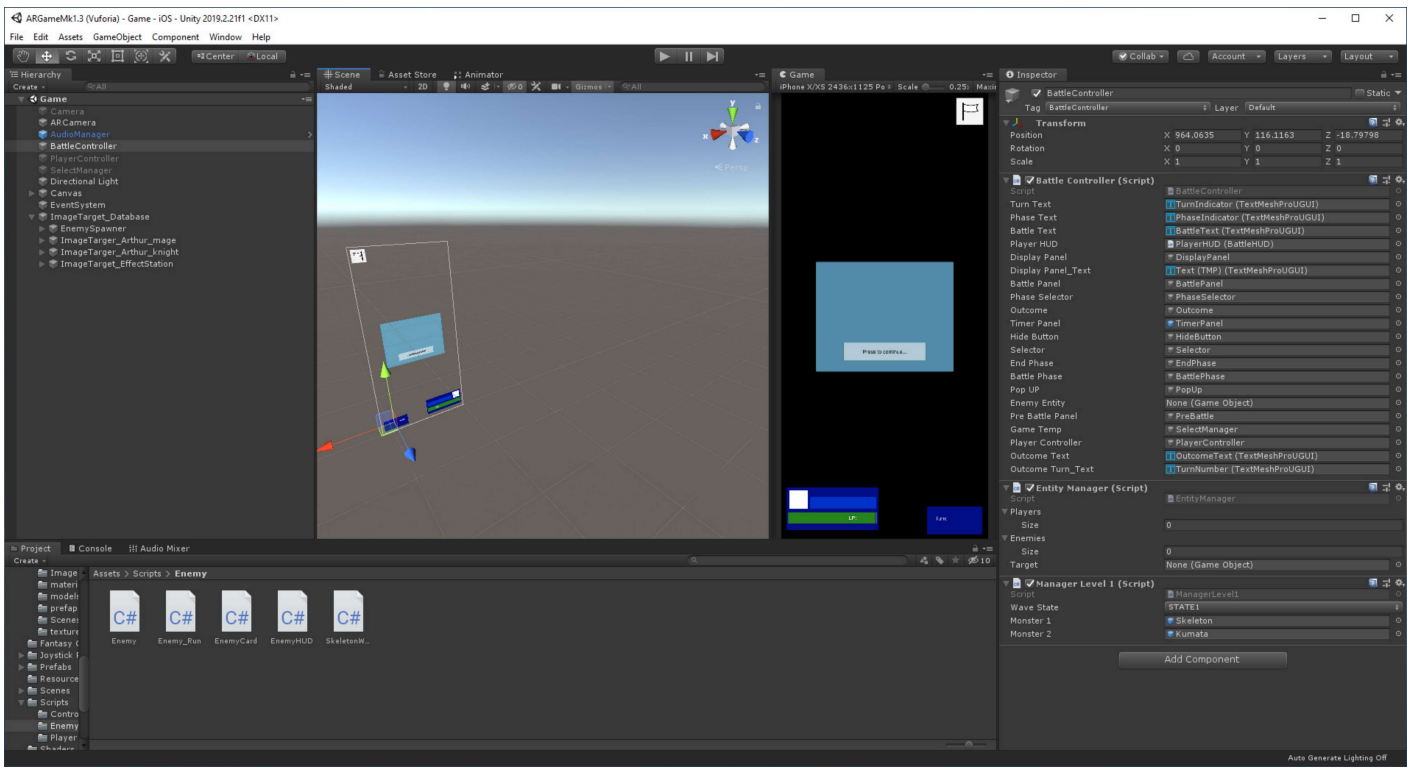

Fig. 25 This figure shows the general working environment, whilst using Unity game engine.

```
void Awake()
{
    dismissButton.onClick.AddListener(Dismiss);
    imageManagerAR = GetComponent<ARTrackedImageManager>();
}

void OnEnable()
{
    imageManagerAR.trackedImagesChanged += OnTrackedImagesChanged;
}

void OnDisable()
{
    imageManagerAR.trackedImagesChanged -= OnTrackedImagesChanged;
}


void OnTrackedImagesChanged(ARTrackedImagesChangedEventArgs eventArgs)
{
    foreach (ARTrackedImage trackedImage in eventArgs.added)
    {
        // Display the name of the tracked image in the canvas
        imageTrackedText.text = trackedImage.referenceImage.name;
        // Give the initial image a reasonable default scale
        trackedImage.transform.localScale =
            new Vector3(-trackedImage.referenceImage.size.x, 0.005f, -trackedImage.referenceImage.size.y);
    }

    foreach (ARTrackedImage trackedImage in eventArgs.updated)
    {
        // Display the name of the tracked image in the canvas
        imageTrackedText.text = trackedImage.referenceImage.name;
        // Give the initial image a reasonable default scale
        trackedImage.transform.localScale =
            new Vector3(-trackedImage.referenceImage.size.x, 0.005f, -trackedImage.referenceImage.size.y);
    }
}
```

Fig. 26 This figure shows original code used for AR Foundation for the prototype of multiple image tracking

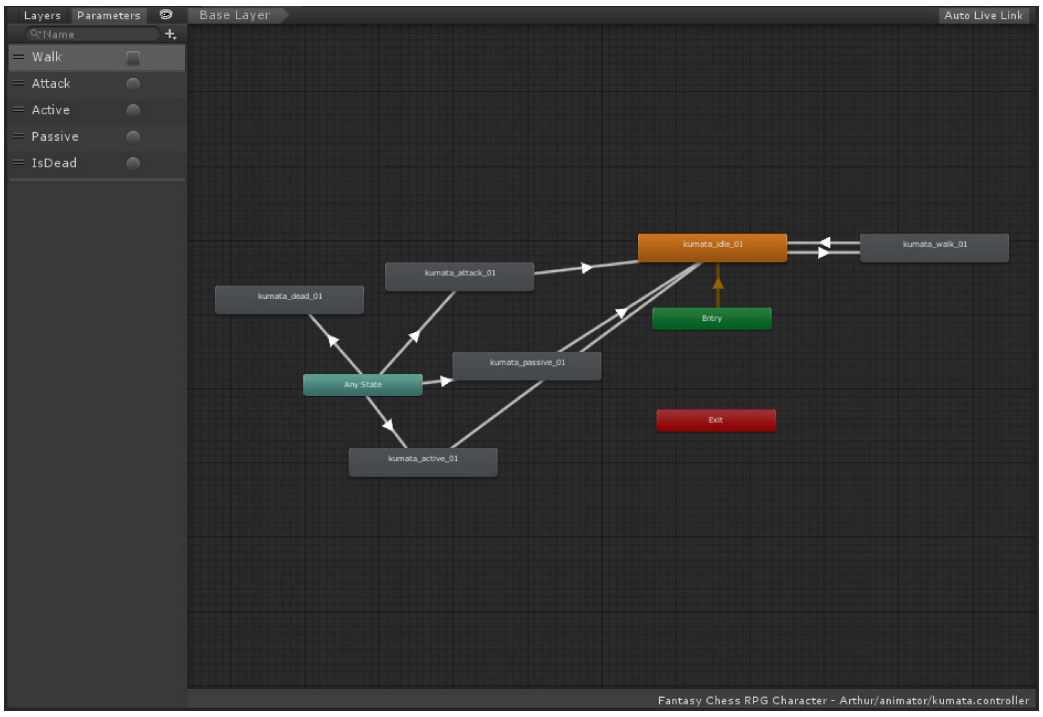

Fig. 27 This figure shows the animator window and how the different states interact.

Supporting Code & Other Material

```
5   public class Enemy_Run : StateMachineBehaviour
6   {
7       Transform target;
8       Rigidbody rb;
9
10      EnemyController enemyController;
11
12      public float moveForce;
13      public float attackRange;
14
15
16      // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
17      override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
18      {
19          moveForce = animator.GetComponent<Enemy>().moveForce;
20          attackRange = animator.GetComponent<Enemy>().attackRange;
21
22          enemyController = animator.GetComponent<EnemyController>();
23          enemyController.EnemyTarget();
24          target = EnemyController.target;
25
26          rb = animator.GetComponent<Rigidbody>();
27
28      }
29
30      // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
31      override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
32      {
33          enemyController.LookAtPlayer();
34
35          Vector3 newPos = Vector3.MoveTowards(rb.position, target.position, moveForce * Time.fixedDeltaTime);
36
37          if (Vector3.Distance(target.position, rb.position) >= attackRange)
38          {
39              rb.MovePosition(newPos);
40          }
41          //Attack Transition
42          if (Vector3.Distance(target.position, rb.position) <= attackRange)
43          {
44              animator.SetTrigger("Attack");
45          }
46
47      }
48
49      // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
50      override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
51      {
52          rb.velocity = Vector3.zero;
53          rb.angularVelocity = Vector3.zero;
54          //Make sure to reset trigger
55          animator.ResetTrigger("Attack");
56      }
57
58  }
```

Fig. 28 This figure shows the different methods implemented for the basic AI of an enemy GameObject.

References

[1] Agile project management glossary | apm. https://www.apm.org.uk/resources/find-a-resource/agile-project-management/glossary/. (Accessed on 05/13/2020).

[2] Apple developer program - apple developer. https://developer.apple.com/programs/. (Accessed on 05/11/2020).

[3] Entity component system - unity learn. https://learn.unity.com/tutorial/entity-component-system#5c7f8528edbc2a002053b679. (Accessed on 05/07/2020).

[4] Lego® nexo knights™:merlok 2.0. https://engine.vuforia.com/case-studies/lego-nexo-knights.html. (Accessed on 05/12/2020).

[5] Unity - manual: Prefabs. https://docs.unity3d.com/Manual/Prefabs.html, . (Accessed on 05/13/2020).

[6] Unity - manual: Scriptableobject. https://docs.unity3d.com/Manual/class-ScriptableObject.html, . (Accessed on 05/13/2020).

[7] Unity - scripting api: Monobehaviour.ondisable(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDisable.html, . (Accessed on 05/13/2020).

[8] Unity - scripting api: Monobehaviour.onenable(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnEnable.html, . (Accessed on 05/13/2020).

[9] Unity - scripting api: Monobehaviour.lateupdate(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html, . (Accessed on 05/13/2020).

[10] Unity - scripting api: Monobehaviour. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html, . (Accessed on 05/13/2020).

[11] Unity - scripting api: Monobehaviour.update(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html, . (Accessed on 05/13/2020).

[12] Unity - scripting api: Monobehaviour.start(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html, . (Accessed on 05/13/2020).

[13] Unity - scripting api: Monobehaviour.fixedupdate(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html, . (Accessed on 05/13/2020).

[14] Unity - scripting api: Monobehaviour.awake(). https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html, . (Accessed on 05/13/2020).

References

[15] Yu-gi-oh! trading card game. https://www.yugioh-card.com/en/about/newto.html. (Accessed on 05/13/2020).

[16] About ar foundation | ar foundation | 2.2.0-preview.6. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html. (Accessed on 05/07/2020).

[17] Cross-platform mobile development in visual studio - visual studio | microsoft docs. https://docs.microsoft.com/en-us/visualstudio/cross-platform/cross-platform-mobile-development-in-visual-studio?view=vs-2019, October 2019. (Accessed on 05/11/2020).

[18] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *IEEE computer graphics and applications*, 21(6):34–47, 2001.

[19] Brackeys. Turn-based combat in unity - youtube. https://www.youtube.com/watch?v=_1pz_ohupPs, November 2019. (Accessed on 05/10/2020).

[20] Kevin Bonsor Nathan Chandler. How augmented reality works | howstuffworks. https://computer.howstuffworks.com/augmented-reality.htm, February 2001. (Accessed on 05/15/2020).

[21] Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J Merelo. Evolutionary deckbuilding in hearthstone. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[22] Christina Gough. • u.s. average age of video gamers 2019 | statista. https://www.statista.com/statistics/189582/age-of-us-video-game-players-since-2010/, September 2019. (Accessed on 05/06/2020).

[23] Mizuko Ito. Technologies of the childhood imagination: Yugioh, media mixes, and everyday cultural production. *Structures of participation in digital culture*, pages 88–111, 2005.

[24] Belinda S Lange, Philip Requejo, Sheryl M Flynn, Albert A Rizzo, FJ Valero-Cuevas, Laura Baker, and Carolee Winstein. The potential of virtual reality and gaming to assist successful aging with disability. *Physical Medicine and Rehabilitation Clinics*, 21(2):339–356, 2010.

[25] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. Coconut: An ide plugin for developing privacy-friendly apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(4), December 2018. doi: 10.1145/3287056. URL https://doi.org/10.1145/3287056.

[26] Xinqi Liu11, Young-Ho Sohn, and Dong-Won Park. Application development with augmented reality technique using unity 3d and vuforia. *International Journal of Applied Engineering Research*, 13(21):15068–15071, 2018.

[27] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[28] Maeve Serino, Kyla Cordrey, Laura McLaughlin, and Ruth L Milanaik. Pokémon go and augmented virtual reality games: a cautionary commentary for parents and pediatricians. *Current opinion in pediatrics*, 28(5):673–677, 2016.

[29] Jonathan Strickland. How virtual reality works | howstuffworks. https://electronics.howstuffworks.com/gadgets/other-gadgets/virtual-reality.htm, May 2020. (Accessed on 05/13/2020).