Development of Strategic Game for Android Devices with AI

Hyun Seok Cho

April 29, 2019

Supervisor: Lazic Ranko S

Second assessor: Bhattacharya S

Dept. of Computer Science
University of Warwick

Contents

Chapter 1

Abstract

Accompanying the explosion of mobile technology such as smartphones and tablets, the mobile gaming industry has expanded significantly with several attempts at replicating the gaming experience of a PC game. Multiplayer Online Arena Battle (MOBA) genre has been one of the most actively played genres during the last decade. On the other hand, it has not taken off for mobile MOBA games. This report will tackle the process of creating a simple mobile MOBA game through a popular real-time engine Unity and the challenges that arise.

Chapter 2

Introduction

The Multiplayer Online Arena Battle (MOBA) games genre has been one of the leading genre's both in players and watchers. The genre has been able to maintain its position in the rankings thanks to games such as League of Legends and Dota2. These games are under the gaming companies Riot Games and Valve Corporation respectively. Both of these companies are private and therefore have no need to disclose their revenue values to the public, however estimates suggest an impressive revenue of 2.1 billion dollars for League of Legends [1] and 406 million dollars for Dota2 [2].
On the other hand, MOBA games have yet to earn a place in the standings in the mobile games industries. Recent charts of the mobile games ranking show that the leading genre is the 'Battle Royale' genre [3]. In fact, there is not a single game in the top 10 that are MOBA related. This report aims to reflect the current state of mobile MOBA games by producing a simplified version from scratch in order to provide a good understanding of mobile gaming and possible reasons to the disparity in popularity between MOBAs for PC and mobile devices.

2.1 Background

To unfold the reasons as to why the MOBA genre is not such a popular mobile game choice, a brief summary of why mobile gaming has increased in demands, what are MOBAs and why they have gained recognition is appropriate.

2.1.1 Smartphones and Mobile Games

Smart phones have been around since early 1990's with the old IBM Simon and attracted some users in the mid 2000's with the invention of Blackberry devices however it was not until the release of the iPhone on the 29th of June of 2007 that smartphones became the majority's choice of mobile communication. According to this graph in figure 2.1 provided by Asymco, smart phones started the Early Majority stage around the 2009-2010 and the expansion of smartphones has only been increasing exponentially ever since. As of july of 2013, the coverage of US smartphone penetration had well crossed the 50%, close to the 60% bar.
Samsung is another leading company that has helped spreading the usage smartphones across the world. This has inevitably resulted in many smartphone users trying out mobile games on their devices. Ever since the release of the iphone, third-party developers started creating applications non-stop. Apple and Samsung have only made it easier for these developers to program new applications by allowing the iOS and Android operating systems to be compatible with the app development tools created by these developers. Tools include Android Studio, Eclipse and Unity amongst many others.

A survey carried out by JITTA in 2004 shows a table with demographics of 714 respondents through a questionnaire involving 96 questions [4] that shows that 95 percent of them have played some form of mobile games before. This is a remarkable achievement considering that the survey was

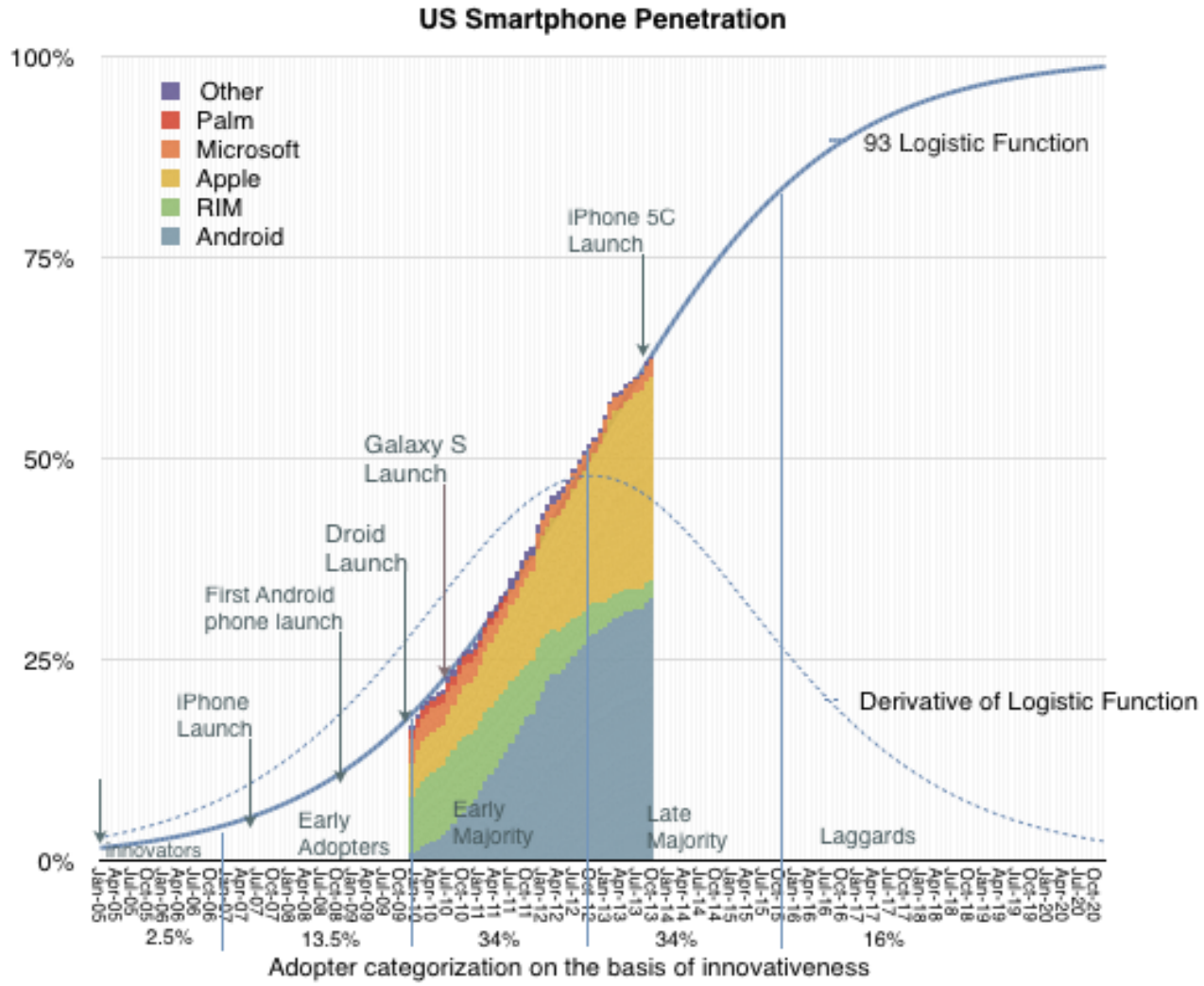


Figure 2.1: Smartphone Penetration in the US

carried out before the explosive global growth of smartphones. It is likely that there is an almost linear correlation between the increase in smartphone penetration throughout the years and the percentage of people having tried mobile games.
As of today, Asia Pacific has been the leading region of growth in mobile gaming in recent years ([5] and [6]) and some graphs released by Newzoo show that over 50% of 2018's global games market derives from Asia Pacific generating revenues of up to 71.4 billion dollars in figure 2.2. Many of the most well known PC and mobile MOBA games have been released by Asian gaming industries. The more well known games in their respective platforms are League of Legends for PC and Arena of Valor, both owned by Tencent. With the help of big companies like this, MOBA games has tried to integrate itself and find its footing on the mobile gaming industry. So why are MOBA games still not in the top rankings?

Furthermore, before contemporary PC gaming, consoles were the most prominent platform; however, as soon as 3D graphics libraries such as OpenGL and DirectX became available to the general public, its reputation in the market has drastically increased. This has helped in the development of mobile gaming as mobile devices' apps are compatible with PC operating systems such as Windows and Mac. Often well-funded mobile applications create PC versions to extend their audience and gain more popularity.

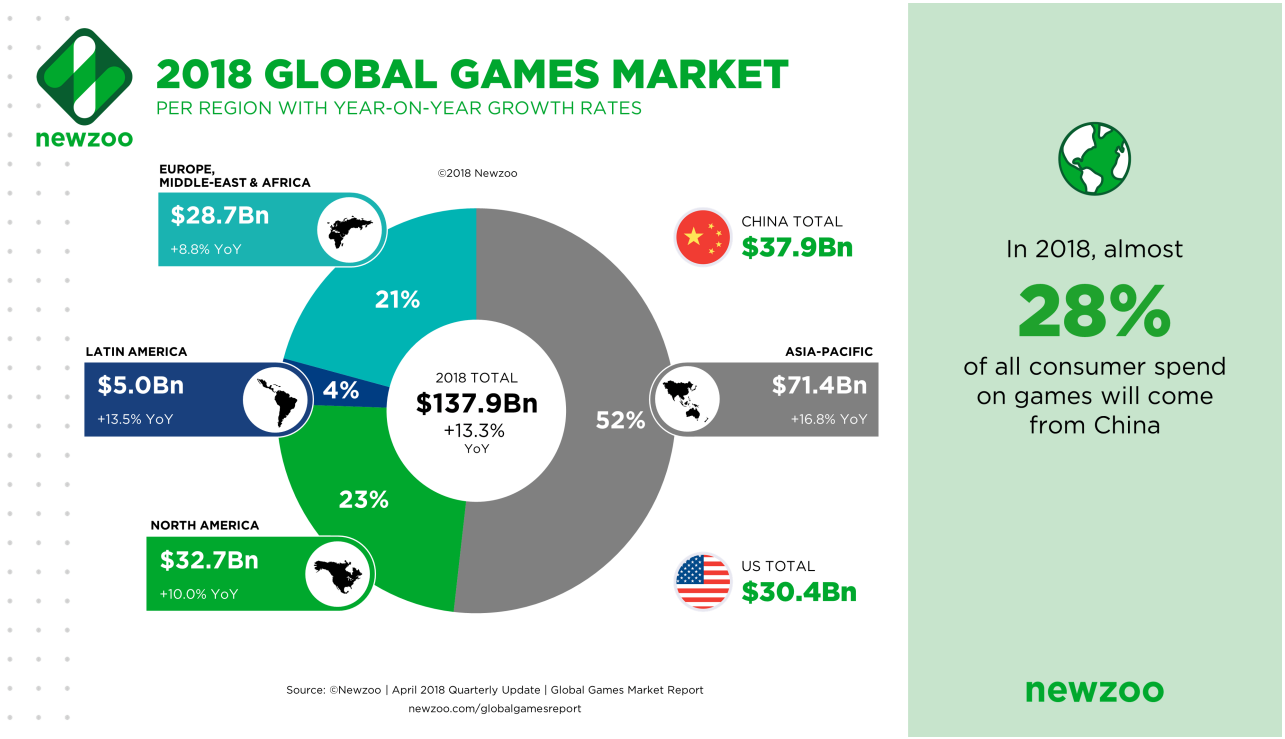


Figure 2.2: Global games market

2.1.2 ESports and Live Game Streaming

The main reason why MOBA's have become such a staple genre in PC gaming is because of eSports and their involvement with hosting competitive tournaments of different games. ESports usually involves organized competitions between professional video gamer individuals or teams. Popular genres associated are MOBAs, real-time strategy games (RTS), first-person shooters (FPS) or card games. This form of entertainment has become so popular in the recent years and caused quite a lot of controversy in terms of whether or not it should be considered a sport. Regardless of its category, its generated revenue of 325 million dollars in 2015 is of economic significance.

As eSports has gained popularity it has even surpassed the viewership of traditional sports [7]. This is partly due to the promotion of live game streaming and the different platforms backing up this activity. The rapidly growing availability of online streaming media platforms such as Twitch.tv and Youtube have sprung eSports to easily become one of the most popular and accessible forms of entertainment nowadays. Chart figure 2.3 created by Medium, an online publishing platform, shows the eSports ecosystem and its interaction with these media platforms and other third parties.

Most of the games hosted in eSports competitions correspond to the games in the top rankings of viewership on the Twitch.tv [8]. However, none of the top games in the list are mobile games. This is one of the reasons why even though PC MOBA games are popular, mobile MOBA games are not. Even though there are ways to stream the contents of your phone onto the platform, the player base pool is much smaller than popular PC games. It simply cannot compete against the big brands.

Another reason why the viewewship is so low for these mobile games is because of the graphics. Graphics play a big role in how good your content quality is. Even if the platform supports 1080p60 resolution, there is little point to it if the graphics built into the game itself are mediocre. Generating impressive visual gameplay for a mobile game can be very difficult specially when you have to balance the tradeoff between performance and appearance. Mobile phones have improved considerably in the last decade, however, the fastest phones nowadays are approximately half as fast as a good PC in terms of CPU performance and they have about half the RAM as well. The GPU of a

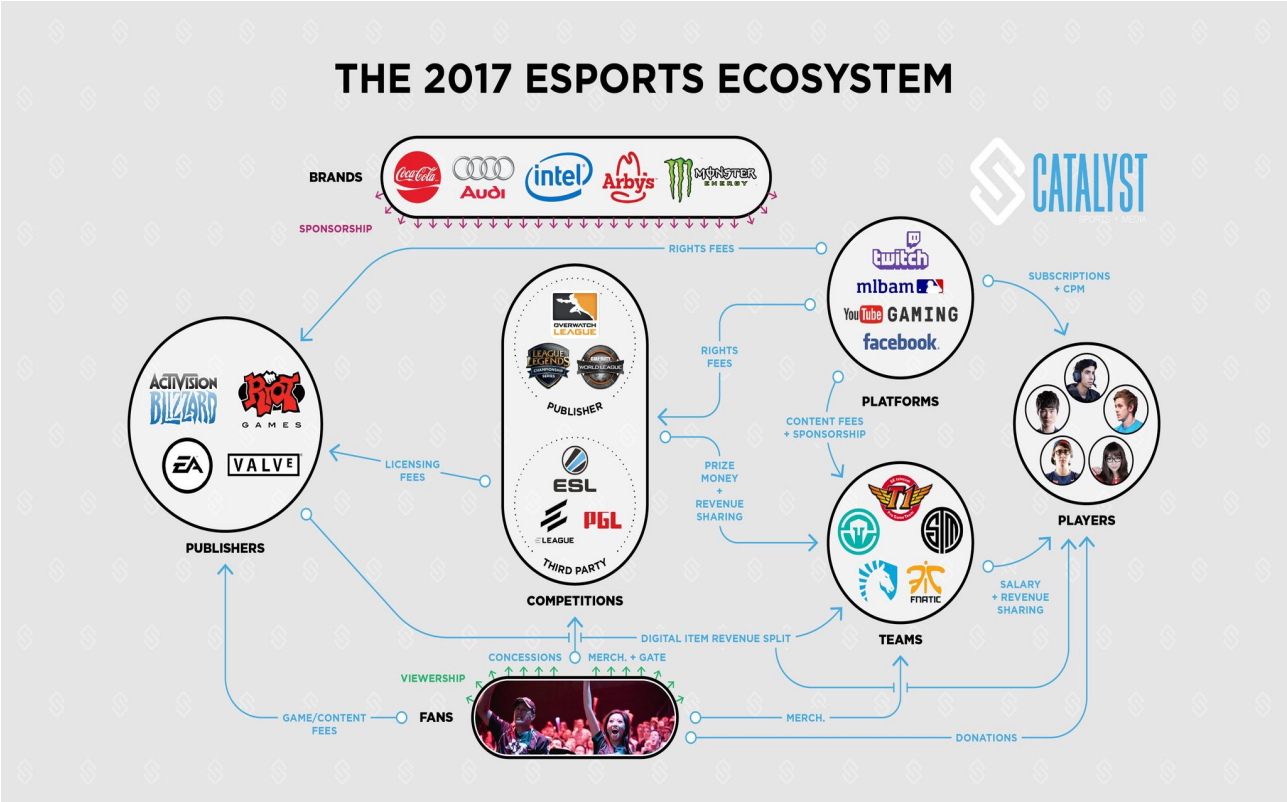


Figure 2.3: ESports Ecosystem and interaction with third parties

high end phone is actually fairly fast compared to an intel integrated GPU however there is a bottleneck in the GPU as there are difficulties in manufacturing and designing smaller and smaller silicon die sizes.
Note though that although high viewership on live streaming platforms helps promote a game, the passive viewers of a game will not necessarily play the game themselves active as suggested in literatures such as 'The Esports Market' [9]. The article makes an observation and a reference to a Newzoo report that states that 40% of all viewers do no longer play the eSports associated games thus rejecting the congruency. This shows that even if the games which are highlighted as the stars of eSports, they are not necessarily the most and most definitely are not the only games being played by the viewers.

2.1.3 MOBA

To fundamentally understand the stated difference in popularity, an understanding of what MOBA is and why PC gamers enjoy MOBAs is crucial. MOBA games is a type of real-time strategy game where two teams, usually consisting of five players each, attempt to destroy their opponents base structure by controlling a character and defend their own. The winner is the team that manages to accomplish this task the quickest. The difference between other strategy games and MOBAs is that typically the former involves minimal or non-existent cooperative team play to win the game and the latter requires the team to act as a unit and communicate ideas and strategies between players to win. As intuitive as it might seem many strategy games are multiplayer, they are not necessarily grouped into teams. Usually, it's a single player against multiple other single players.

MOBA games have a characteristic and quite distinct map that most of them typically follow. This map consists of a base, positioned at the bottom left hand corner and top right hand corner for each team, and three straight lanes extending out of the base called top, middle and bottom. These lanes connect to form three pathways from base to base. These lanes are equipped with turrets and players must destroy turrets in descending order of distance from the base. A generic map can be seen

in figure 2.4.

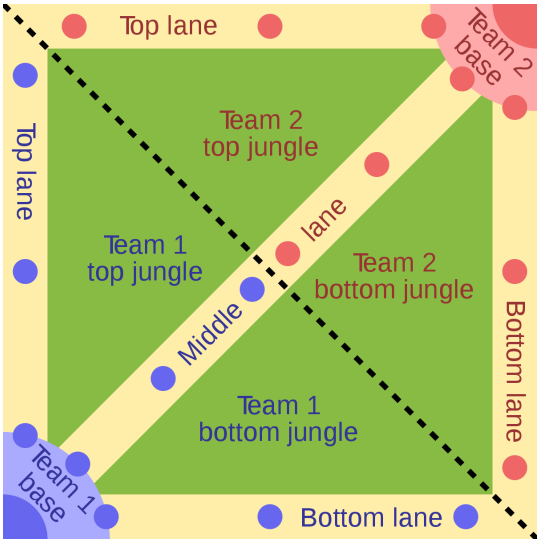


Figure 2.4: Classic layout of MOBA maps

Many of these real-time strategy games involve some sort of form of income generation and collection. MOBA games also have this trait and it is usually in the form of destroying enemy units. These units can come in multiple forms such as artificial intelligent mobs, structures such as turrets, the characters the opponents are controlling, and/or neutral mobs that favour no team in particular. This aspect of the game allows more experienced and skillful players to perform better than the opponents thus earning bigger rewards and introducing the term 'carry' which involves a player outperforming the opponents to an extent where it benefits other members of the team.
Players can then use this income to buy items and customize their own character as the game progresses. These items improve character's statistics such as defense, attack or utility numbers which all help make the character stronger in one way or another. MOBAs have a strong emphasis on what is known as 'builds'. This term is used for itemization choices. After the release of the game or a patch of the game, builds are still relatively blank and players tend to buy any items they want, however, after a period of time, more experienced players are able to distinguish what item purchases yield better results in term of performance and what gives them a better chance at winning the game. Unlike MMORPG games, most items are available for any character disregarding the character's ideal function which opens up the possibilities for the players and allows their creativity to be part of the game.

Chapter 3

Motivation and Problem Statement

Although current MOBA games are at the pinnacle of PC game genres, its neighbouring mobile games have yet to see much growth in their fan base. The reason for this is due to the limited experience a MOBA mobile game can provide. Even with the latest upgrades in processing power and graphics, the smaller screen on smartphones makes these types of action real-time strategy games harder to play. As of 2019, there are no top MOBA games which introduce a third-person view which is what distinguishes the action type obtained from FPS team games and MOBA games. The only third-person view MOBA game out in the gaming industry that has gained some recognition by its peers is SMITE.

SMITE was released by Hi-Rez Studios in March 25, 2014. The game's theme was based on mythodology which was a unique concept at the time. The game has also managed to enter the eSports scene and has continued to hold competitions annually, up to six seasons this year. On the year of release, the game was so popular that the prize pool for the first ever SMITE World Championship, held by Hi-Rez Studios, was a total of 2.6 million dollars, only behind Dota2 and even higher than the current MOBA leader, League of Legends. Steam charts, figure 3.1, reveal that the player base follows a fairly unpredictable growth in between years. Nonetheless, the number of concurrent players does seem like it has been decreasing slowly throughout the last 4 years, starting from 20000 players at any one time and ending at 16000 players. About 20% of the player base has stopped playing or play less frequently. This is a significant number and the reasons are simply that the game has reached a stagnant point where the player base is not really growing and thus it is slowly decaying as is the natural life cycle of video games.

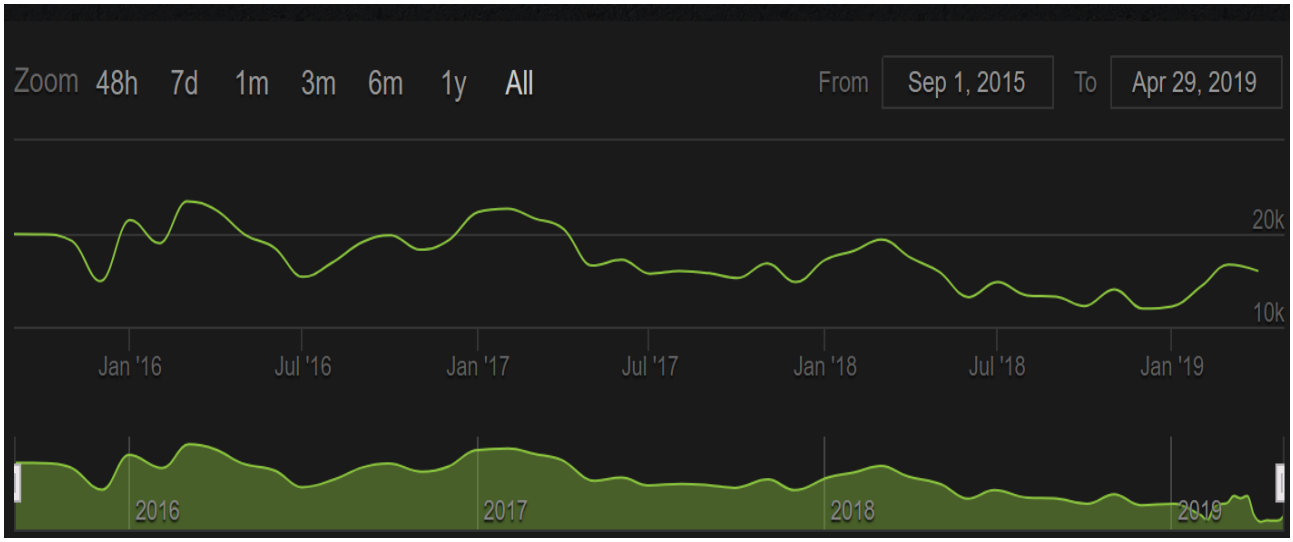


Figure 3.1: Chart of number of concurrent SMITE players between September 2015 and April 2019

More elaborate reasons as to why MOBA games are not as popular is because mobile games them-

selves are not attracting as much of a fan base as other forms of video gaming. With the rapid growth and expansion of technology globally, players have developed interests in hardware computing. This has created a spurt of players dedicating time to building personal computers specially because some hardware components are getting cheaper and better over time. This has lead players to spend more and more time playing PC games rather than mobile games. Data provided by Stone Temple shows that even though mobile devices are used more than desktops in the US, the gaming industry category is only used by 47% of people [10]. This verifies the fact that the limiting factor is not the mobile usage of customers but rather their involvement with the gaming industry.

A technical problem that arises due to the screen size is that the user's control over the game mechanics (character movement, abilities, and overall awareness of the surrounding) is far less precise than when playing with a mouse and keyboard. Utilizing a mouse gives you a much larger range of motion and the resolution of a desktop screen is so much larger than that of a mobile device that the displacement of mouse to character can be kept relatively small. This smaller movement scales give rise to precision and control over the game which allow players to excel at the game and outperform their opponents. In mobile devices, actions are limited by the space provided as acceptable UI and the game visual itself must be kept in a very small screen, which means you are limiting the player on the methods he/she can use to outmaneuver the enemy. This type of constraint pushes players away from mobile games as they end up bored of repeating monotonous actions over and over again.

Chapter 4

Objectives

The objective of the project was to learn about the difficulties, risks and development of mobile games in general and ultimately create a simplistic MOBA mobile game for Android called 'Glyn'. Along the journey, any difficulties were recorded and any suitable amendments were researched and implemented.

The main features of the completed product should include the following:

1. A visually appealing world environment for the player
2. A set of controllable player characters
3. A controllable and intuitive third-person view camera setting
4. Artificially intelligent objects
5. Appropriate user interface
6. A multiplayer gameplay

Chapter 5

Methodology

5.1 Unity3D

Unity is a real-time engine created by Unity Technologies in June 2005. The engine supports a variety of platforms and operating systems including:

- iOS.
- Android.
- Windows desktop.
- Mac OS desktop.
- Linux desktop.
- WebGL.
- Unity Web Player

The engine allows users to create 2D and 3D games. The main scripting API is C# which is very similar to Java as they are both object-oriented languages thus the transfer of knowledge from one to another is not a major difficulty.

Unity contains two built-in physics engines which handle the physical simulations for the user [11]. The physics engines ensure that the game's components move realistically and react appropriately to the environment they are exposed to. What this entails is a more immersive experience for the player. Furthermore, this engine's physics properties' can be edited allowing the users to create fictional environments if desired. A common example would be to reduce the gravitational forces applied on a given object to make objects float.
Moreover, the physics engines eliminates one of the most expensive computations in gaming: rendering. The engine supports multiple rendering techniques however the default technique is 'Forward Rendering'. Forward rendering is a quick and cheap method of simulating light on objects. Each object is rendered depending on the number of light sources affecting it. This methodology has an apparent disadvantage of a 'render cost on a per-light basis' [12]. This can, in the long-run, be computationally expensive as more light sources are added to the game, however, for this project's aim a singular light source will suffice and as such the computations required to render the objects are not as significant. Using the default rendering technique also saves time as no research nor coding is required for other possible techniques.

The engine's user interface is easily comprehensible and its drag-and-drop features make programming faster. Additionally, the engine provides a real-time code refreshing system which allows users to make quick changes to variables and observe the changes in the game-play immediately without waiting for compilation and rendering time.

Another useful debugging method is the developer mode in Android devices. This mode is compatible with Unity and it allows users to test their applications by connecting their devices to the Unity engine via USB. Unity then automatically downloads all the necessary packages and assets required for the application to run. This method of testing is mostly used after an entire component is integrated into the game. Testing the applications on mobile devices is crucial as the resolutions are different than on personal computers.

5.2 World Environment

The world environment is composed of two separate sections:

1. Map Layout: The map layout refers to the shape of the map and the basic overlaying design of the paths in the world.
2. Terrain: The terrain is the actual shape of the floor and how it is manipulated to match the map's layout and give a clear indication to the player of what the map layout is like.

5.2.1 Map Layout

The map layout is, as mentioned in the background of MOBAs, a diagonally symmetrical three-routed map. Each route, known as lanes, has its own set of objects however, this does not mean lanes are independent of each other. They are all connected through subpaths between the lanes which is known as 'jungle'. The reason why the lanes are connected is because otherwise the game would no longer be a team vs team game and rather just a player vs player game. Having the jungle gives players the option to freely move around the map to aid allies or defeat enemies and come up with creative strategies to defeat the enemies as a team or simply stay in their respective lanes and focus on their individual performance.

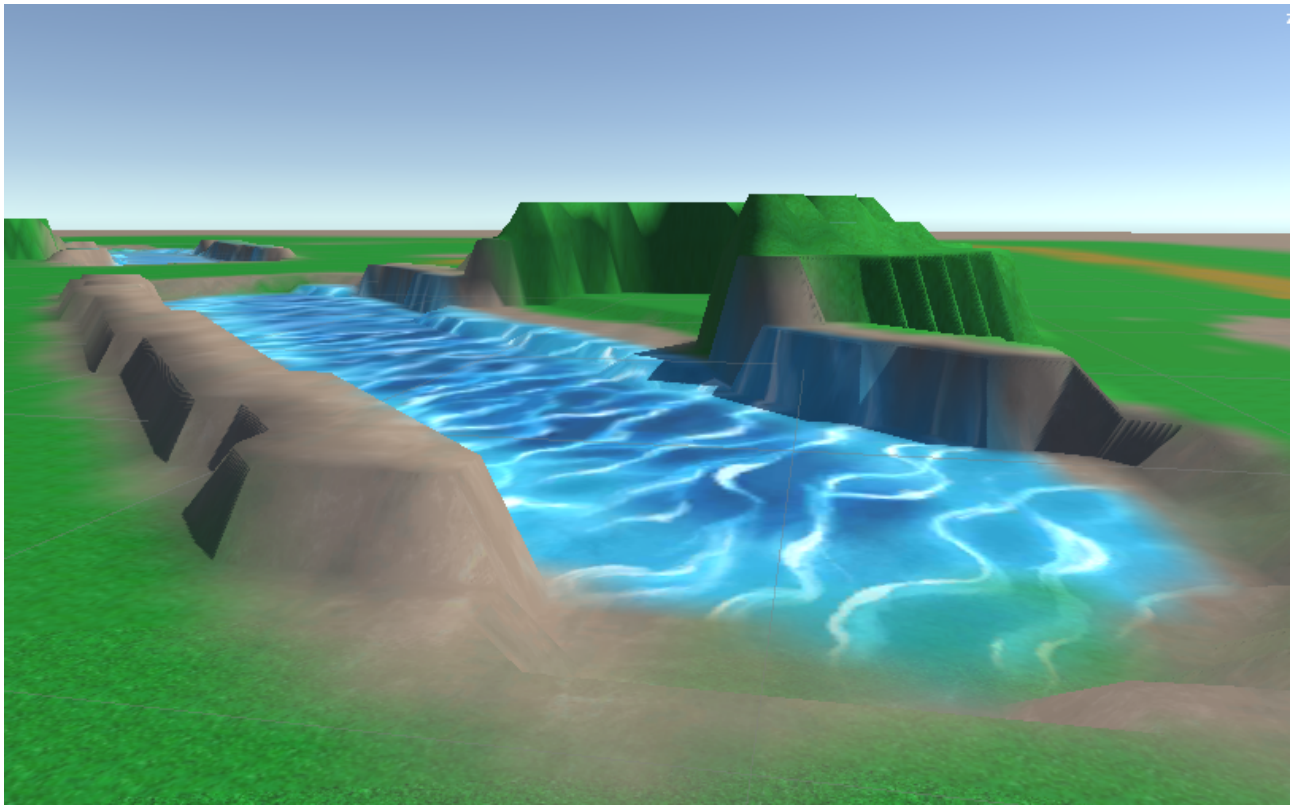


Figure 5.1: Glyn's Textured Terrain

5.2.2 Terrain

Unity includes a feature called 'terrain' which allows the programmer to create a flat blank terrain on an axis (horizontal axis in this case) which is included in your project as an asset. The asset contains a set of four tools which are all very useful for designing and decorating your terrain [13]. The brush tool was used to make elevations in the terrain and simulate mountains and hills. A disadvantage of this method is that the brush tool only allows the terrain to be raised, not lowered. Consequently, any depressions that are desired cannot be formed. As a countermeasure, the height of the terrain was set to -2 in the y-axis, after which we use the brush tool to bring the majority of the terrain to a height of 0; allowing us to form depressions using the range -2 to 0. An example of this type of terrain is shown in figure 5.1
Another component of the terrain asset is the terrain collider. This component ensures that the terrain contains the physical properties of the Unity physics engine thus making it solid and interactable [14].

5.3 World Decoration

This section is mostly a design question and most, but not all, of the decisions are subjective. Some decorations are objects or terrains in the battlefield which obstruct or interact with the characters in any manner. Other decorations have no effect on the gameplay and their sole purpose is to provide an aesthetically pleasing visual.

The largest and most noticeable decoration are the mountainous rocky structures which encase the map. These have been made to prevent players from going out-of-bounds and accessing restricted areas which might cause bugs in the game.
Other structures that affect the movement of characters are the elevated terrains inside of the map. The ground has been elevated by a height greater than 2 in several areas to ease the recognition of pathways in the jungle (keep in mind, that the total minimum height of these terrains must be 4 units as the terrain itself is already at a height of 2 units as mentioned in the World Environment section). The raised areas are not traversable as the maximum height difference possible between the terrain and the character is 1 unit. Other areas have been lowered by heights of 0.5 and 1 to represent depressions and mimic the formation of rivers. Note that these rivers are indeed crossable since they are meant to be pathways for the players. As mentioned earlier, these types of restructuring are done with the brush tool on the terrain asset.
Moving onto visual decorations, the game includes:

- The terrain has been textured differently for specific areas to represent their section. Due to time constraints the textures had to be downloaded from an asset package which were prebuilt. The land has been textured with a green grass texture to mimic a field. The river has a blue turbulent water texture. Patches around the turrets have been textured with some muddy textures to make them look realistic and not just structures that come out of nowhere. And finally, the main lanes have been textured with a sand rocky texture to represent footpath eroded areas [15].
- Ground covered in grass-looking assets as seen in figure 5.2. These objects do not interact with the players and they are able to walk through them without collisions.

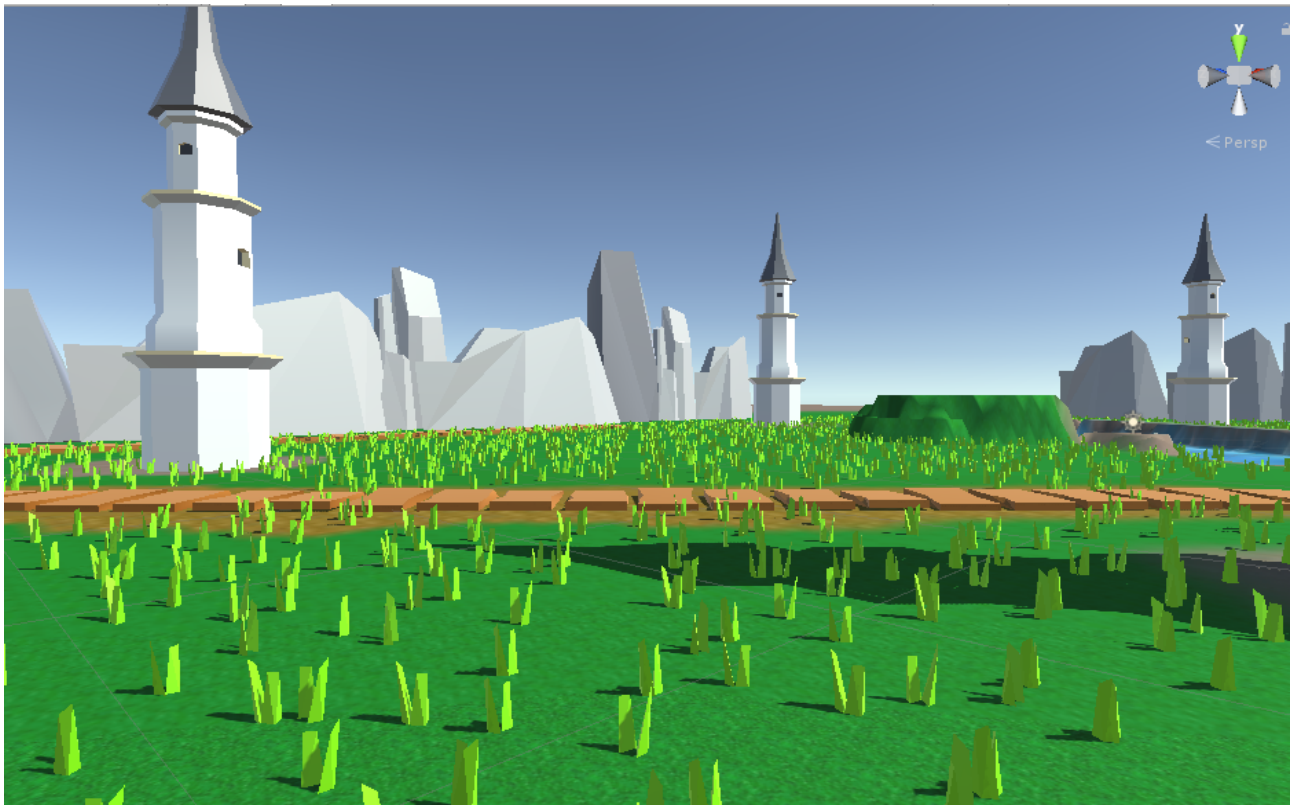


Figure 5.2: Glyn after some world decoration

5.4 Player

The player object is a simple capsule object observable in figure 5.3. The reason why this particular shape was chosen was because of its 2 unit height and its 1 unit width. This is the most humanoid shape. The aim of the project is not to create stunning visuals but rather learn the about the process of developing a MOBA gameplay, therefore any character visuals are left out in this project.
This also helps when it comes to gameplay as making the game recognize each individual extremity of a character as part of the character hitbox can be very expensive. To overcome this barrier, a 3D shape asset is chosen to represent the character. The animated character is then added as a component and the asset's mesh renderer is turned off to avoid overlapping images.
Just like the terrain, the capsule has a capsule collider component almost identical to the terrain collider. This collider is required because otherwise the player object would simply fall off the world without resistance from the terrain.

5.4.1 Rigidbody vs Character Controller

There are two main components that can be assigned to a player character and those are the rigidbody and the character controller components.
The rigidbody component sets the object's movement under the influence of Unity's physics engine [16]. This implies that the body would experience gravity, momentum and other physical properties that a real-life physical body might have. On top of that, other rigidbodies that collide with this entity would also experience forces on them which make them move.

Contrarily, the character controller does not use Unity's physics engine and instead allows the user to determine the different physical properties that are attributed to the object [17]. These properties can be adjusted to make the character move the way the user visualizes it and not necessarily in a realistic manner if that is not the goal in the first place. In this project, the player object benefits more from using the character controller because of the collisions. If the player was a rigidbody,

any collisions with the environment would cause the body to react and thus the camera would also move. Furthermore, the environment part that collided with the player would also experience a force which displaces it. Albeit having some displaceable objects in the game, the majority of the map should not be constantly changing position as the players come into contact with them.

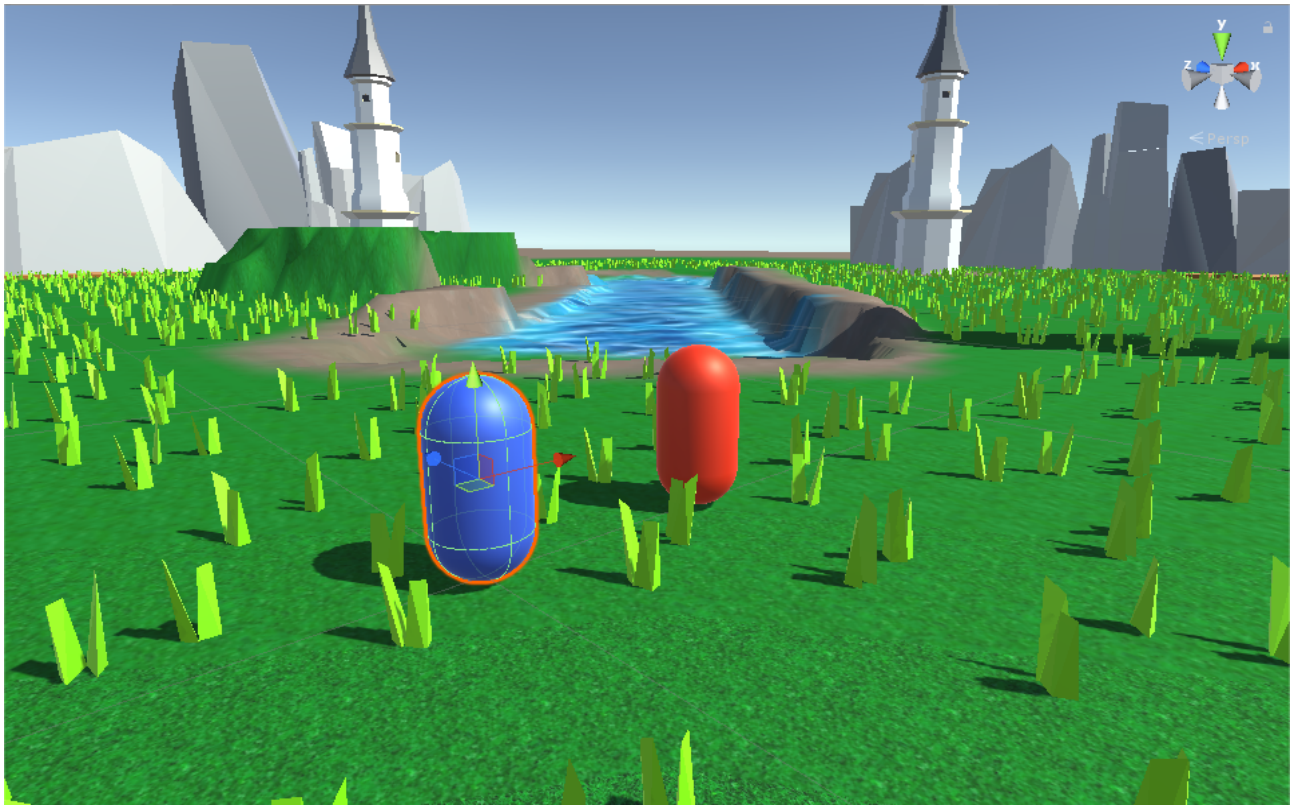


Figure 5.3: Players capsule

5.5 Virtual Joystick

Generally, games that require the user to move their own character on mobile devices use virtual joysticks. These input devices report the angle or direction that the user desires the character to move towards. For the purpose of mimicking a realistic person's ability to displace and rotate at the same time, the player movement and the camera rotation have been assigned to separate joysticks. The left joystick controls the player movement and the right joystick controls the camera rotation. Since the joystick is part of the UI, although it is an advanced mechanism, their objects must be represented in the 2D graphical user interface. This can be visualized in figure 5.4.

There are several assets available at Unity's Asset Store. A free popular package is the 'Joystick Pack' which contains multiple types of joysticks that can be implemented into the canvas of the screen with a 'simple drag and drop'. Each joystick contains a customized script which calls the event trigger system called 'OnDrag()' which is fired every time the pointer is moved during dragging [18]. The function of this event is to obtain the current position (x and y axis on the screen) of the joystick on the screen and find the difference against the joystick's fixed original position to determine which direction the user is dragging the joystick ball towards. These 2D direction vectors are stored in variables ready to be used for movement or rotation.

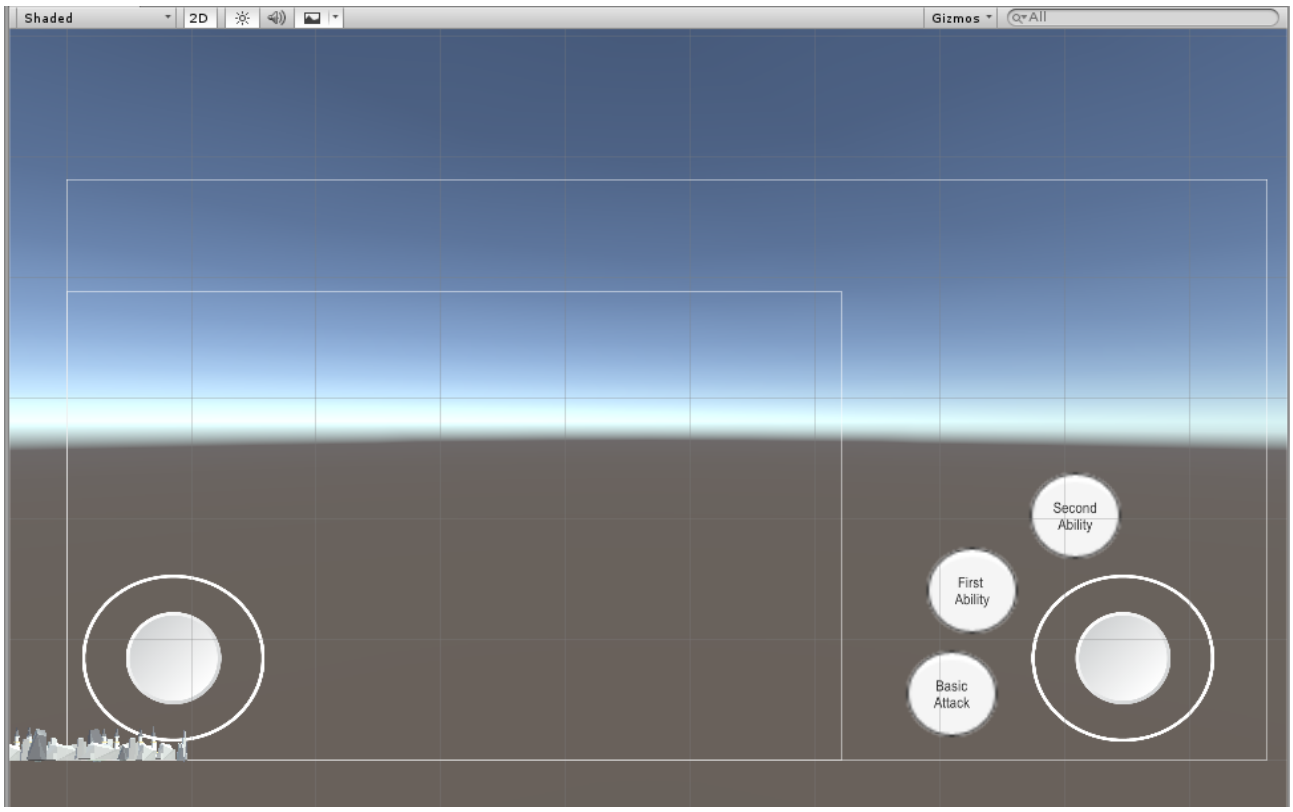


Figure 5.4: Joysticks in the 2D plane

5.6 Camera

MOBA games have customarily adopted a bird's eye perspective which allows a much better view of the surroundings. In our case, Glyn has opted for a third-person view. The reason behind this decision is that a third-person view restricts a player's awareness of the surrounding at any given moment. Although the game allows a 360 degree rotation around the y-axis, the player still only sees what is in front of them and a little bit of the back. This limiting factor adds tension to the game as sneak attacks are made possible which make the player's be more attentive that they would if they could observe the entire map. Third-person view also adds originality to the game as almost no MOBA games have tried this method except for SMITE.

Moving on to the technical aspects of implementing the camera, we have a camera asset which is initialized for each player upon starting the game. This camera always contains a camera controller script which performs the calculations for the camera. Before anything, the camera contains a public variable of type transform called target, which is assigned as the player's character; this is so that each camera only obtains information about one player's character and not all others.

After this assignment, the character becomes the pivot point for the camera, around which it rotates as shown in figure 5.5. The rotation computation is performed in a 'LateUpdate()' function as suggested by the Unity documentation [19] because this accounts for any movement that the character could have performed inside of 'Update()'. The function's primary task is to obtain the horizontal and vertical positions of the camera joystick (right joystick) on the screen much like it was done for player movement. These values are then multiplied by the rotation speed factor, which has been assigned as 2 after some testing, to give us a horizontal rotation float value and a vertical rotation float value. Now, to rotate the player, we use the target transform 'rotate()' function which takes in 3 float values (x, y and z), and assign the rotation value to the y-axis and the other two axes as 0. Now that the character has rotated the player's camera must follow and rotate with it. To accomplish this, the current rotation of the target (the character) is obtained in the form of Euler angles.

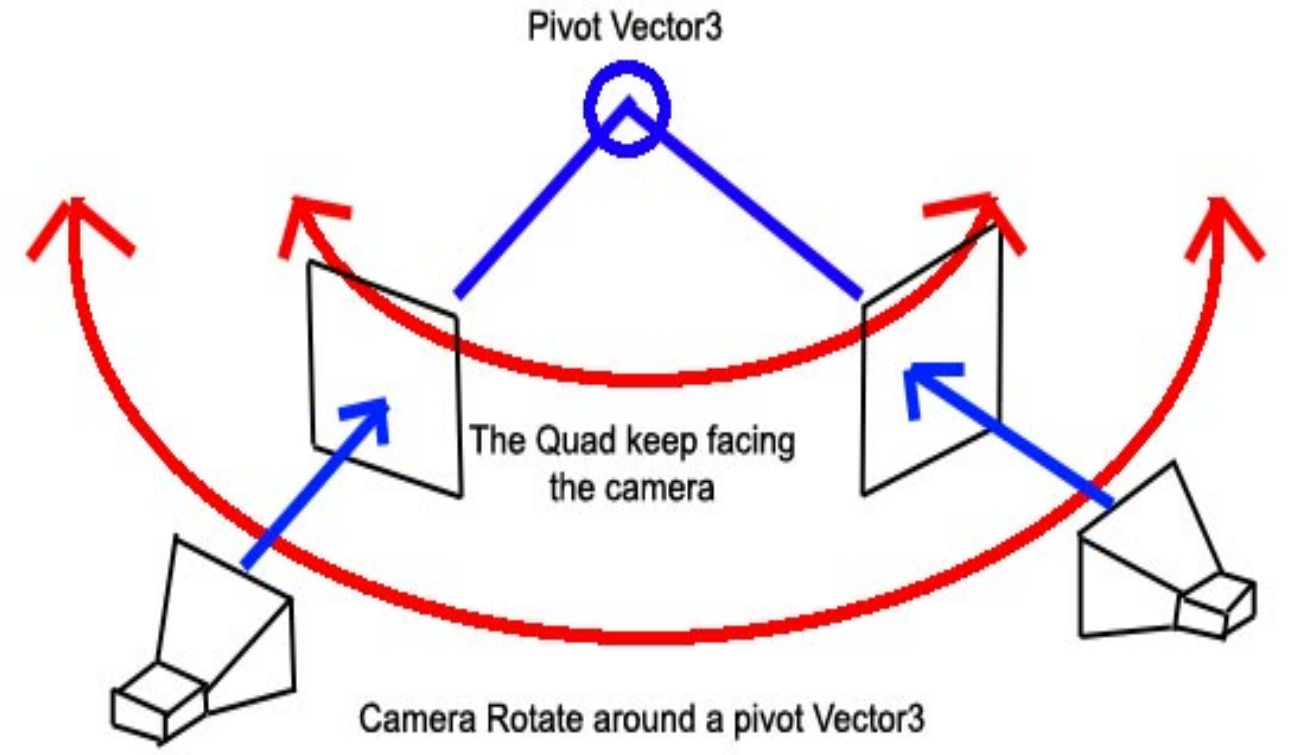


Figure 5.5: Diagram representing the camera rotation

'Euler angles provide a way to represent the 3D orientation of an object using a combination of three rotations about different axes' [20]. The article also explains Euler angles' limiting factor known as 'Gimbal Lock'. This problem arises when a degree of freedom/rotation is lost due to two or three gimbals being shifted into a 'parallel configuration' [21]. For this exact reason, the Euler angle values are converted into a quaternion.

Quaternions are the number system that Unity uses internally for all the rotations. Quaternions do not suffer from 'Gimbal Lock' and provide some advantages when it comes to smoother interpolation [22]. However, they are more complicated to understand intuitively. For the purposes of the camera rotation, the position of the camera will be changed to the target's position minus the quaternion calculated.
Now that the camera has been displaced, the character is longer centered. To fix this, a simple 'transform.LookAt()' built-in function with the character as its parameter is enough.

5.7 Player Movement

The player movement was achieved by creating a script which accessed the character controller component of the player object. The code was built upon the tutorial provided by Unity on moving the player using the character controller [23]. This in-built function called 'move' moves the character in a given direction. This direction of movement must be in the form of a 3D vector [24]. In order to calculate this 3D vector, all 3 axes must be analyzed to obtain the correct direction of movement.

- The forwards and backwards movement (movement in the z-axis): This is represented by the movement joystick's (left joystick) vertical component value multiplied by the player's forward vector. The vertical component contains information about the device user's vertical finger displacement and the player's forward vector applies a force in the z-axis considering

its rotation. What this implies is that no matter the rotation of the player, the forward vector is always be relative to the direction the character is facing.

- The sideways movement (movement in the x-axis): This movement is represented similar to the forwards and backwards movement but in this occasion, the movement joystick's horizontal component value is multiplied by the player's right vector. The horizontal component contains information about the device user's horizontal finger displacement and the player's right vector applies a force in the x-axis considering its rotation.
- The up and down movement (movement in the y-axis): For this particular game, no jumping or crouching is implemented so the only thing that is required is the falling mechanic. This mechanic is implemented by utilizing the physics gravity's y component [25]. This might seem confusing and misleading as the player object is a character controller is not a rigidbody as mentioned earlier. This implies that physics is not applied to this body, and gravity is no exception. However, in order to recreate the falling mechanic that would otherwise be there if the player was a rigidbody, the script must apply a downwards force on the player object and this value is being calculated using the gravity's y component multiplies by the gravity scale (which is just a public variable that decides on how strong the gravitational forces are) and Time.deltaTime.

5.8 Touch Events

Touch events are events which are triggered through the use of buttons. These buttons are User Interface objects which are placed on the screen and not on the scene and can be 'clicked to trigger an event' [26]. The player characters all have a script called 'playerShoot' which take in a button and listen to it for any incoming messages. This sets up a communication system between the player and the Unity system activated by clicking that button (note that this clicking works for PC's through a mouse click and for mobile devices through a touch). Upon clicking, the character fires a normal projectile with its specific settings such as speed, damage and time to live.

5.9 Turrets

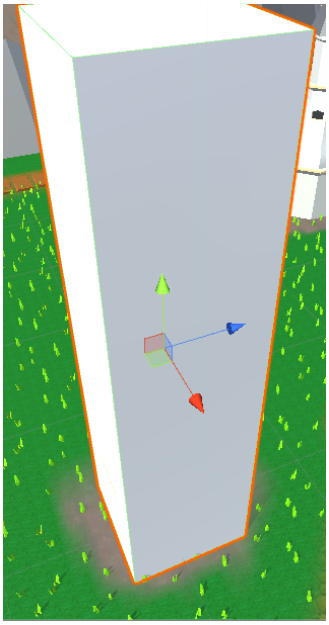
The main objective of the game is to destroy the opposing team's main structure whilst defending your own. In order to improve such a one dimensional concept, turrets are usually placed in each of the 3 lanes for each team which make the games last longer. The usual number of turrets per lane is 3, however, considering the audience and the platform the game is meant to be played on are mobile devices, the turret number has been reduced to 2 per lane to make the games faster paced and less drudgery.
All turrets are composed of 3 primary components: a turret structure, it's corresponding projectile spawn point and the turret's hitbox. The structure is the physical object which is visualized in the scene; the spawn point is a simply a point within the structure from which the projectiles are supposed to fire from; the hitbox is a cuboid surrounding the turret structure which has been created to ease recognition of collisions (the hitbox is explained in further detail later on). The difference between the turret's structure and the hitbox is easily observable when comparing between subfigures 5.6a and 5.6b.

5.9.1 Turret AI

The turrets have two main scripts:

1. Tracking System: Once the target has been identified, its world position is obtained using the transform.position attribute and compares it to the previous position of the last target. If the positions match, then no rotation is needed since the turret is already orientated towards it.

(a) Turret's structure with hitbox mesh renderer: OFF



(b) Turret's hitbox mesh renderer: ON

Figure 5.6: Difference between turret structure and turret hitbox

In the case the current and previous positions do not match, then the new rotation is calculated by utilizing the built-in function of the quaternion type 'quaternion.LookRotation()'. This function calculates the rotation between two world coordinates aka 3D vectors.

Once the rotation required is obtained, the rotation is checked against the turret's rotation. If the turret is already facing the correct angle, then no change is made. Else, the turret is rotated using 'quaternion.RotateTowards()'.

2. Shooting System: The shooting script instantiates projectiles at a projectile spawn point depending on whether the target is within the turret's range or not. The 'Vector3.Distance()' function is used to calculate the distance between two world coordinates. The script also implements a fire rate and a fire timer in order to threshold the number of projectiles fired within a time frame.

5.10 Projectiles

Projectiles are used to attack and deal damage to other game objects with health. This has been one of the most basic forms of attack in many MOBA games because it is such a dynamic weapon. Projectiles can have an immense range of variables meaning that new different types of attacks can be created easily and quickly.

5.10.1 Projectile

The normal projectiles are projectiles with the simplest mechanic. Upon firing, they follow a linear trajectory until it is destroyed. These projectiles are used for player basic attacks since they are plain and straightforward without any special abilities. They have three public variables: a damage variable, a speed variable and a time to live variable. The first two are self explanatory. The time to live variable is a float value which determines the time the projectile has before it is destroyed. The way this idea is implemented is simply by using the 'Destroy' function which does exactly that. Another useful effect of the time to live variable is that it also acts as a range variable since depending on the speed variable the projectile can only displace itself by a specific distance.

5.10.2 Tracking/Homing Projectile

Tracking or homing projectiles are projectiles that once fired, they follow the target until collision. Unlike the normal projectiles, these projectiles are used for turret attacks. They also differ in the sense that they do not have a time to live therefore they are undodgeable. The reasoning behind this design was so that the players were forced to leverage the turret's targeting through minions however the creation of automated minions was never completed (this topic is discussed further in the last chapter).
The tracking projectile also has a target variable which is obtained from the tracking system mentioned in the Turrets section. This target's world position is acquired and stored. By utilizing the Vector3.MoveTowards() function, the position of the projectile can be redirected towards the position of the target. Due to this code being in the Update() function, the position of the target and the projectile is recalculated every frame and thus generating a seemingly homing projectile that tracks the enemies.

5.10.3 Collisions

Collisions so far have not resulted in any changes in the environment. Nonetheless, this time around the projectiles must be destroyed upon collision. There exists an in-built method known as 'OnCollisionEnter(Collision)' which is an event which is triggered whenever a collision between the parent collider and another collider occurs during gameplay. This method is passed the 'Collision' class which contains information about the collider in question. This information can be used for the detection of specific game objects and differentiating collisions between inanimate objects such as terrain and mountains and animated objects such as players and turrets.

As explained previously, the normal projectiles belong to the basic attacks of characters meaning that any collisions before the time to live limit will cause the projectile to be destroyed. On the other hand, the tracking projectiles that turrets shoot will only be destroyed if the collision occurs between itself and the targeted character. To distinguish between characters, the target's game object tag is compared with the collision's tag, if they are equivalent, the projectile is destroyed and damage is dealt, otherwise the projectile continues its course.

5.11 Health

Health is a must-have component in almost any game which involves killing. The health system is used on any game objects which can die or be destroyed through attacks. A game object with health will contain a health system and its corresponding health bar. The health system is what controls and regulates the health of the character whilst the health bar is just a visual representation of the character's health. These components are dependent on one another. Since health is such a widely used concept in so many games there is similar software available all over the Internet. For this project, the health system was based off of a Unity3D video released by EYEimaginary [27], a youtube channel dedicated to creating Unity3D tutorials and teaching beginners and advanced developers.

5.11.1 Health Bar Controller

The health bar controller's main function is to keep a register of all the health bars active in the game. Upon starting the game, the health bar controller is instantiated and it initially consists of an empty key-value dictionary of health systems as keys and health bars as values. The reason why some type of list is required is because we want to be able to apply changes to health bars independently. As an extreme example, lowering the health of one character should only lower its corresponding health bar, not everyone elses.
Furthermore, it is not advisable to keep health bars which are no longer being used. Ideally, whenever a game starts, any game objects with the health system script should instantiate a new health

bar for that character and add it to the dictionary. These entries must be unique in order to be able to retrieve them individually. Upon destroying a game object, its corresponding entry is deleted from the dictionary. By reducing cache memory, the system's performance can improve significantly.

5.11.2 Health System

The health system script contains two major functions: Damage() and Heal().

The Damage() function takes in a damage integer parameter which is subtracted from the current health when the subject is dealt damage. For obvious reasons, this operation is only performed if the game object's current health is greater than 0.

The Heal() function takes in a heal integer parameter which is added to the current health when the subject is healed. This operation is only performed if the game object's current health is greater than 0 (aka is alive) and if it is less than the maximum health. In the case that healing the character exceeds the maximum health, the current health is set back to the maximum health.

Whenever the health is changed a public event is triggered to notify the health bar that it needs to update its visual to show the correct health.

5.11.3 Health Bar

The health bar is a combination of two images. The background image is just a grey horizontal cylinder-like image which will act as the missing health. The foreground image consists of a smaller green version of the background image but it's type is no longer 'simple'. Instead, the type 'filled' has been selected with the horizontal fill method. This is a feature provided by Unity which displays only a portion of the image by only colouring a section of the Sprite and leaving the rest transparent. We are then able to access this foreground image in the health bar script and manipulate the fill amount property which is a percentage representation of the displayed image between the values 0 and 1. An example of what this health bar looks like is presented in figure 5.7

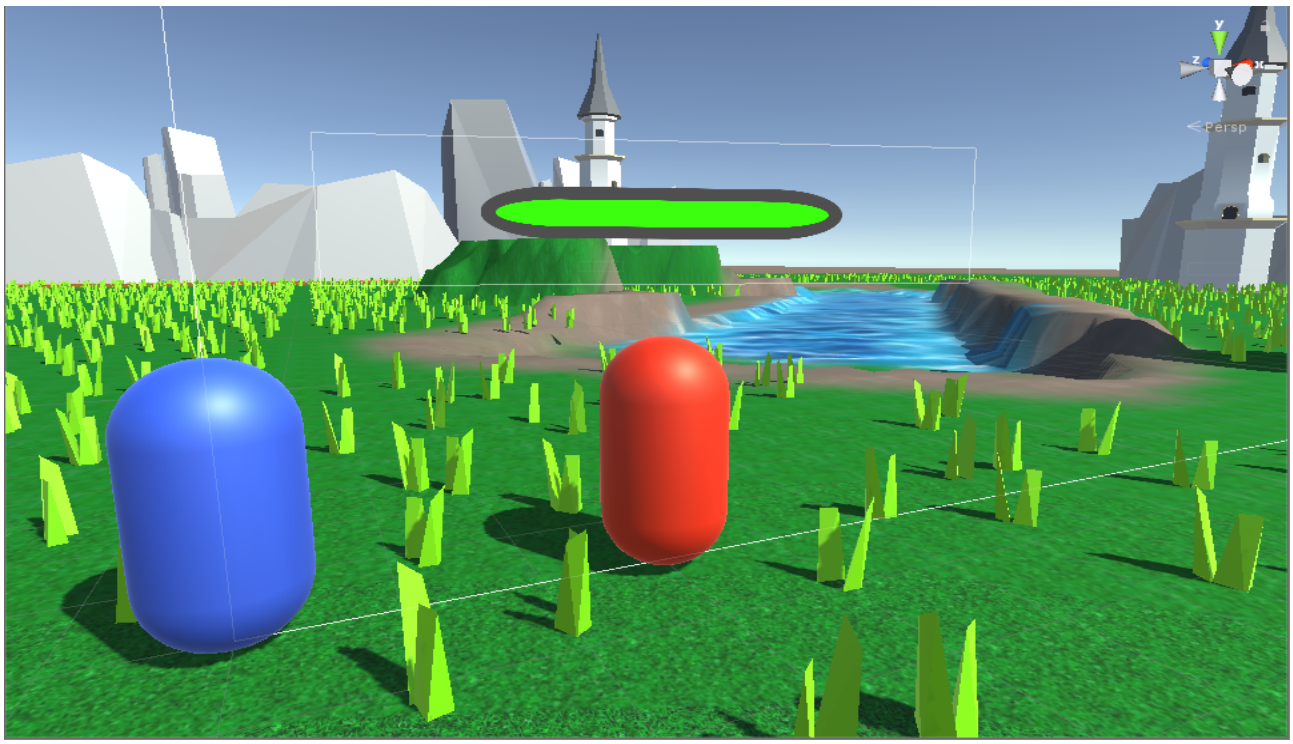


Figure 5.7: Health bar example

The health bar contains the function ChangeToPct() which has been assigned as a coroutine. 'A coroutine is like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame' [28]. This ensures that the character's health bar drops gradually as a procedural animation rather over the course of several frames, rather than instantly disappearing.

The health bar orientation and size also updates depending on the player's camera perspective so that it remains facing towards the camera at all times and the size is the appears to be the same even if the player gets closer or distances from the target. A visual example of what it would look like without this updating can be appreciated when you leave the game mode in the Unity Editor and move the main camera **??**

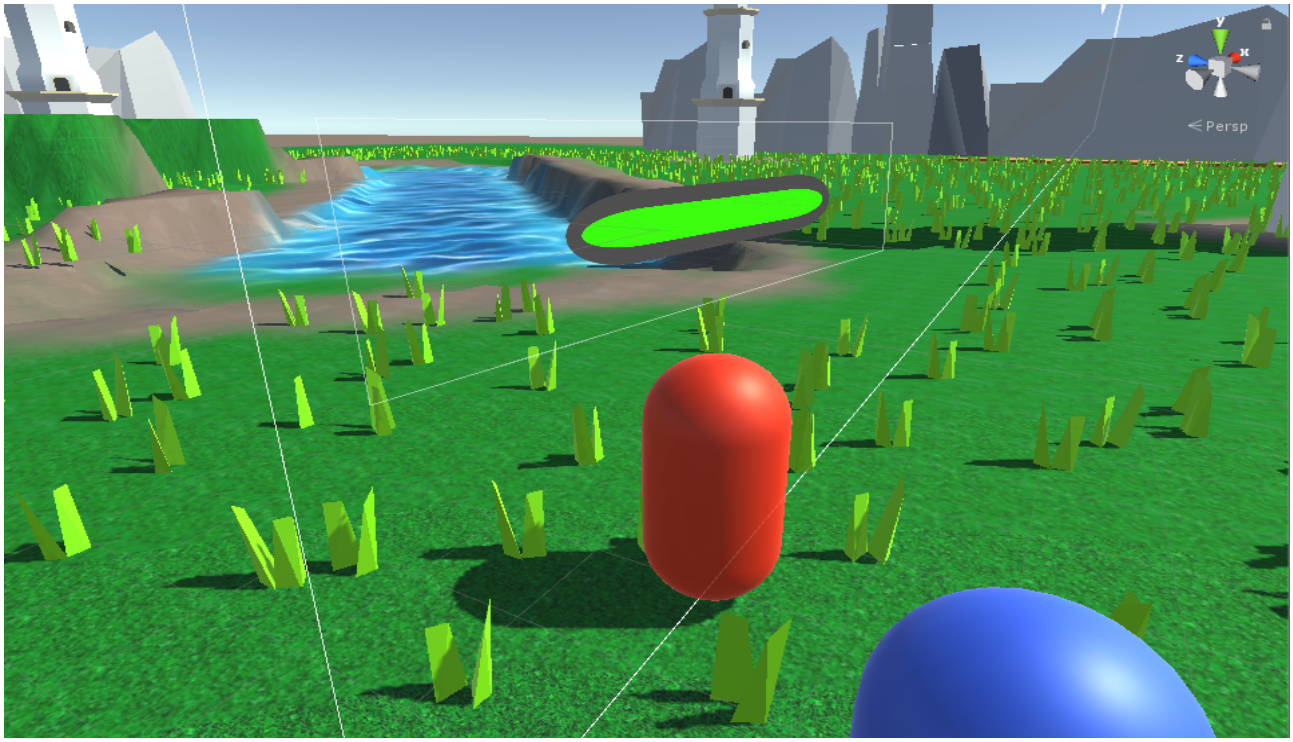


Figure 5.8: Health bar without orientation update and camera rotated by approximately 45 degrees

5.12 Item System

Items are usually implemented into games to add diversity to the game since playing with the same characters with the same statistics can become fairly monotone over time. Items allow players to customize their characters within a match which can bring out creativity and strategy.

A personalized item class has been created, each consisting of 9 attributes:

- ID
- Name
- Cost
- Description
- Icon
- Power
- Armor
- Extra health
- Extra mana

For clarification, the item icon consists of a 64x64 pixel RGBA compressed ETC2 8 bits image no filter that represents the item. Since artistic skills are not the main focus of this project, these icons have been obtained from asset packages available at the Unity Asset Store [29] and [30]. Many more attributes can be added, however, for the purpose of demonstrating the capability of the system these 9 attributes seemed more than enough. All of these items are stored in an item database which is a list of item objects. The list of items and their status can be viewed in table 5.1. Note that these items have not been balanced and are still just a prototype. These items still need require separate testing to check their power levels considering their prices and player interactions.

5.12.1 Shop

Items must be acquired by visiting a shop in the player's respective base. Said shop is a cube which upon contact with the player, activates a shop panel containing all the items in the item database and some information about them such as their name, their cost and their description 5.9. The items are displayed in a button format so that the players are able to tap the item they want to purchase and Unity can trigger the onClick() event which will in turn equip the item. Furthermore, the benefits of using a panel for the shop are that the opacity of the panel is lower than other UI which means that you can actually see through the panel, allowing you to keep attention on your surroundings if need be. Additionally, the shop is great for using the layout group script components because one can layout the numerours buttons of the shop in a clean and organized manner.
All players have an item inventory which consists of 5 slots. These slots are initially empty and are only used when an item is purchased from the store. The screen canvas has five more rectangular buttons anchored to the bottom which can be clicked to use said items which will immediately delete the icon and deleting the entry from the player's inventory.
One obstacle that was not overcome was the automatic adjustment of the inventory icon dimensions to match the resolution of the device being used. The purpose of this was to have an inventory icon for when the item slot was empty, however, even though the buttons resize, the custom icon does not because it is a fixed size 32x32 bit image. A solution to this would be to create a script that manually stretches the icon image to the same dimensions of the item buttons.

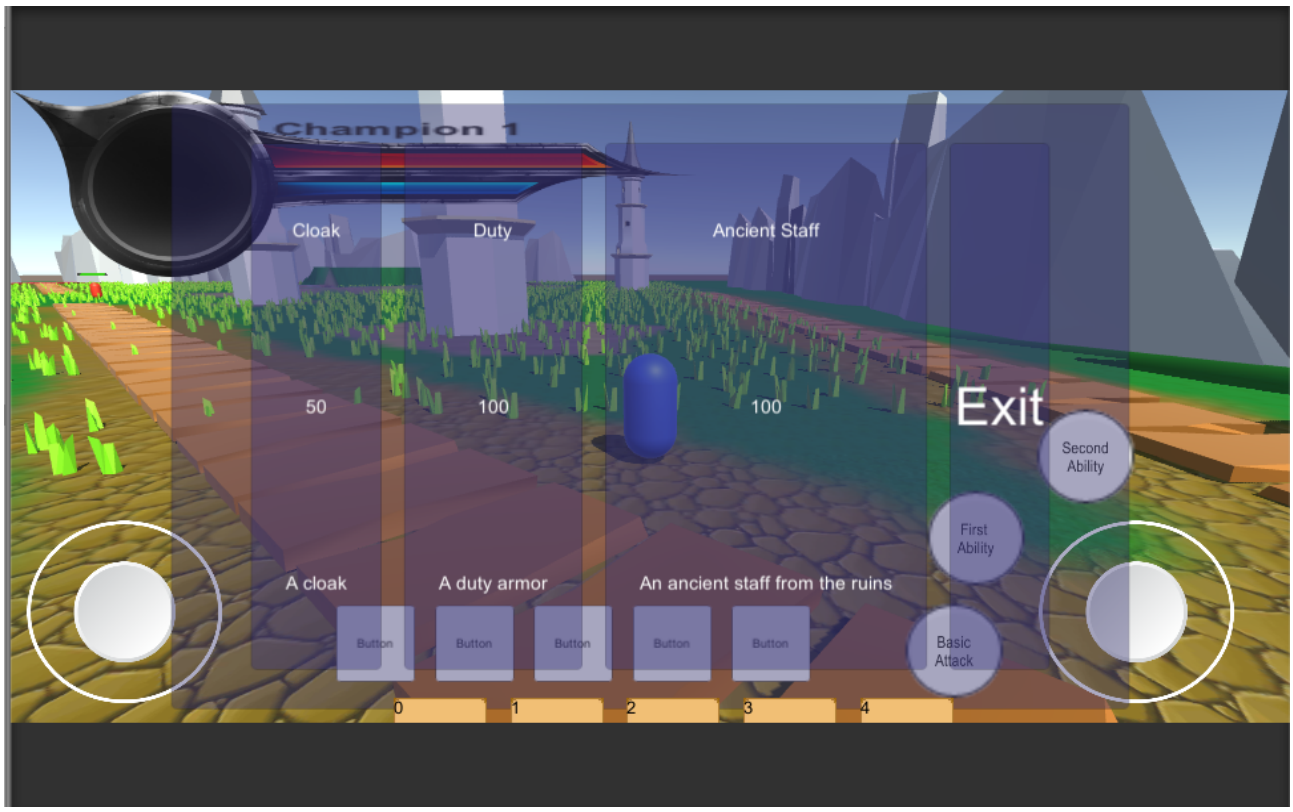


Figure 5.9: Shop panel open

5.13 User Interface

User interface is a section of game development which its importance is often overlooked. The user interface is a way for humans and computer to interact with each other. In our case, its a way for players to interact with the game and receive information on the consequences their actions have had on the game. It is very important that the user interface is well developed as a game with poor user interface can result in the player not understanding what their actions are doing and thus feeling unsatisfied. For example, whenever a player shoots and successfully lands a hit on an enemy, the game's UI should notify in some way that the player that the enemy has been damaged.
Many forms of UI have already been discussed throughout this report such as the health bar changes or the items being used. For that reason, this section is focused on the graphical user interface that is used in Glyn to represent the current player's health and statistics.
There are two main types of UI that game developers have been using to overcome the difficulty of displaying the game and the UI, all in the same 2D plane. These two types are 'Non-Diegetic' and 'Diegetic'.
Non-Diegetic user interfaces are those which are on 2D graphical user interface. These are usually overlaid onto the game screen, so it appears flat and in front of the action of the game; it is not part of the game world. Diegetic user interfaces are part of the game world and have some affiliation with the story line of the game. The problem with these is that they are not present at all times and some action might be required to access them such as having your character move to a certain location or maybe coming into contact with something; for example the shop. However, the player's own health bar and statistics is something that the player should have access to at all times. Furthermore, having a health bar on top of your own character at all times in a diegetic manner can be disruptive as it can block the view of the player since the camera always has the character as its pivot point. These lead to the removal of the player's character health bar and the implementation of a 2D GUI health bar only viewable by the corresponding player 5.10.
The design of the GUI has been obtained from an RPG-style Diegetic user interface available at the Unity Asset Store [31].

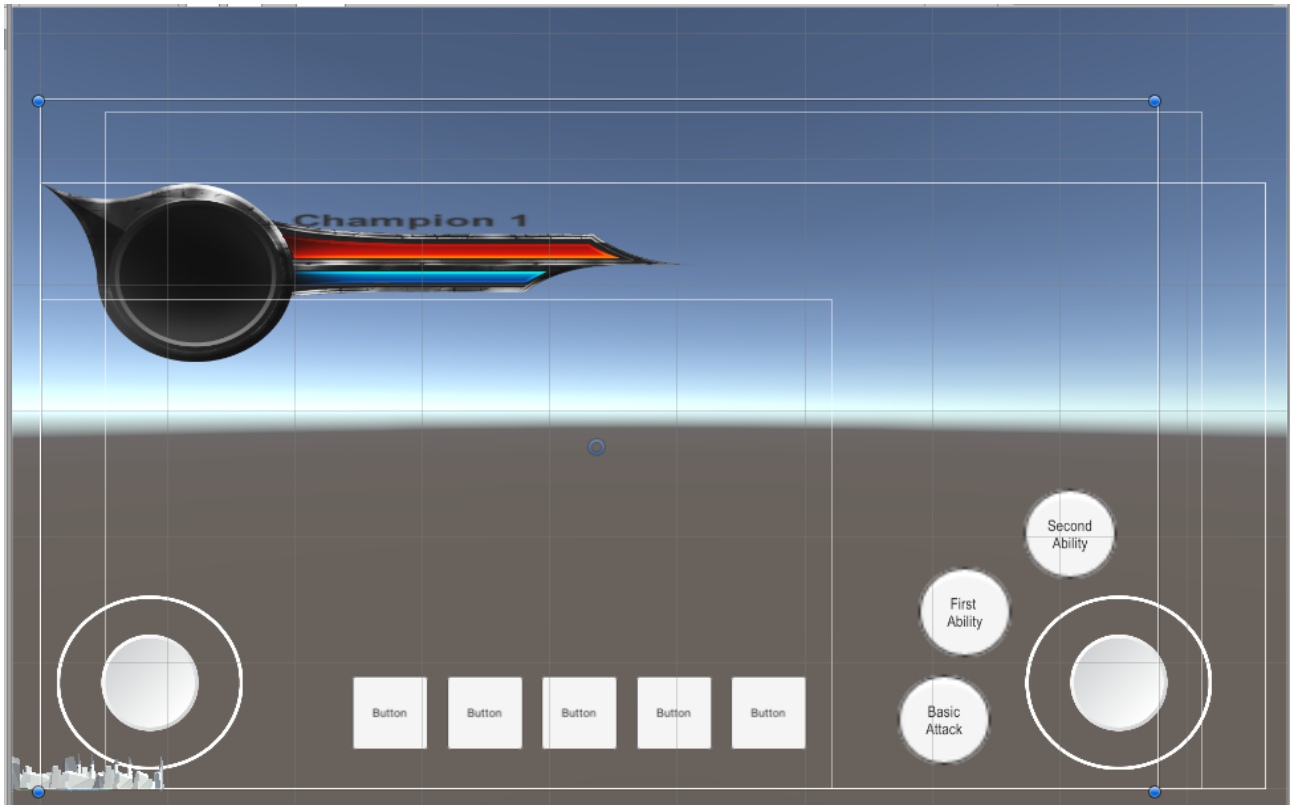


Figure 5.10: GUI Health bar integrated into the 2D UI Interface

Table 5.1: Database of items

Item ID	Name	Cost	Description	Power	Armor	Extra health	Extra mana
0	Cloak	50	A cloak	0	5	0	5
1	Duty	100	A duty armor	0	5	20	0
2	Ancient Staff	100	An ancient staff from the ruins	5	0	0	20
3	Axe	100	A one-handed axe crafted by dwarfs	10	0	0	0
4	Chainvest	75	A chainvest to protect from blades	0	10	20	0
5	Chestplate	150	Prepared for battle	5	30	0	0
6	Crimson Armor	200	The highest knight class armor	20	30	20	0
7	Dagger	25	Small but deadly	5	0	0	0
8	Demonic Armor	250	Found in the Nether	20	20	0	30
9	DoubleHanded Axe	150	Swing swing to win win	40	-5	0	0
10	Flail	150	Wing your weapon and deal huge damage	30	0	0	10
11	Hatchet	50	A quick buy	10	0	0	0
12	Iron Chestplate	100	Crafted with the finest iron	0	10	10	0
13	Iron Hammer	100	Simple but effective	10	0	10	0
14	Knight Sword	200	Enforce justice with this sword	30	0	0	20
15	Knight Armor	200	No wonder would dare attack a knight!	0	40	0	20
16	Light Armor	75	Be agile in the battlefield	5	10	0	0
17	Longsword	125	Classic sword	20	0	0	0
18	Metallic Stick	100	What the barbarians used	20	0	0	-5
19	Mystic Staff	200	A staff made by Mother Nature	30	0	-10	30
20	Persuader	50	Old but effective	10	-5	-5	0
21	Robe	100	What the wizards would use	0	5	0	20
22	Scepter	175	Paladins love this weapon!	15	0	5	10
23	Silver Armor	125	Shiny and light	0	10	0	30
24	Sledgehammer	125	Brute force	30	-10	-5	-10
25	Sleeveless Armor	75	Feeling hot?	0	5	5	0
26	Spiked Mace	125	The spikes are not decoration	25	-10	0	0
27	Steel Hammer	150	An improvement of previous hammers	15	0	15	0
28	Sword	75	Simple sword	15	0	0	0
29	Heavy Armor	150	Become a tank	-20	30	30	-5
30	Weak Armor	50	Any armor is better than none	0	10	0	-5
31	Wooden Bat	25	Just a stick	5	0	0	0

Chapter 6

Project Management

Given that the project consists of producing a simple mobile MOBA type game using Unity, time was required for both research of Unity and MOBAs, as well as time for the actual software development. In order to ensure adequate progress that will lead to the completion of the project with no delays, a detailed Gantt chart was created at the time of project specification. Gantt charts are useful for organizing large number of activities that need to be completed on schedule. This methodology also helps in including workaround time in case problems arise with any of the components. This chapter will present the final version of the timetable 6.1 along with the explanations about why certain decisions were made when planning the project and any adjustments that might have been made throughout the academic year.

Most of the activities that are listed in the Gantt Chart are unit components of the game which were allocated specific time periods and deadlines for completion. This period of time was dependent on the difficulty of the task at hand and the extensiveness of knowledge required to implement such component (i.e. how much research time I estimated to need). Following this planning structure, the first semester of the year (first week of October to first week of December) was designated to creating the world space and creating some of the basics in video gaming such as camera control and player movement. Logically, the project requires some insight and getting used to Unity's working environment and functions which is appropriate to assign to the early stages of the project.

The month of December contains a longer period of work on two of the units: 'Long touch events' and 'Short touch events'. This allocated time is unusually longer than other items, even though they are relatively simple to implement, because the work is to be done during holiday break. Personal issues could arise making my personal computer unavailable for a long period of time, as was the case.

Starting the second semester of the year, the focus of the project shifted to implementing individual systems such as the health system and the item system. Object oriented systems was a familiar topic which is why not much further research was required. A noteworthy section in term two is the 'Health System Debugging' activity that is included after 'Item System'. The original planned chart did not include this part. As explained earlier, Gantt Charts usually help the organizer to allocate some time as a margin of error and this is typically long enough to not delay the progress of the project. Notwithstanding, sometimes integration of entire systems into an already existing project can be challenging. The accident in this exact scenario was that the health system was not getting implemented correctly and the engine was not initializing neither the health bar nor the health system for many of the identities in the scene that had the health system script due to an error in script execution order. Analyzing the error, specially detecting the causality, and researching for a solution on this matter took longer than anticipated which is why a new activity was added in the halfway through the project. For reference, the solution was to access Unity's script execution layer ordering and manually configuring the engine to instantiate all the objects with health first before attempting to add a health bars/systems component to them.

Lastly, a sizable error that was detected around mid May was that there was not enough time to complete the player vs player (PvP) integration which included the network management and the back-end. This mistake was caused due to the inexperience in long-term projects and on game development. During the first draft of planning, the workload of writing the final report was underestimated thus making the assumption that the PvP feature would be available along with the report. For the final Gantt Chart, the network management and the back-end were removed to free up time and dedicate more attention to the final report.

Some might consider the time dedicated to the overall research of the implementation to be insufficient. Nonetheless, considering that Glyn is more of a practical project rather than a scientific one, most of the activities associated with the project are much more accessible to the public than scientific or literature reviewed articles. A fair amount of the implementation techniques are already in Unity's documentation or in tutorial videos by other independent developers.

Figure 6.1: Final Timetable

Chapter 7

Risks

Carrying out the project itself is not a physically risky procedure in the least, however there are some risks that should be kept in mind when it comes to potential system failures or problems with the community.

One of the risks involved in MOBA games is the extensive strategic options available to the players. As the game can be played out in numerous different ways, all dependent on the player, these can result in players having conflicting ideals and approaches which can make the game confusing. This type of community disagreement can lead to toxicity and spiteful comments. The reputation of the game could be thrown off and affect its potential growth in the industry.

The risk of copyright infringement [32] is a potential risk in this situation. Although it might seem obvious that the solution is to not plagiarize software, the game development is such an explored research area that a vast amount of the implementation of general games has already been patented. That being said the copyright law does not protect ideas. Although there are plenty of MOBA games out there, and this project has been inspired by them, there is no law that states that the same ideas cannot be re-utilized by an independent developer. On the other hand, copying the software would be considered copyright. Hence, all the code in the project is either reused from publicly released code by other developers or written from scratch using the Unity documentation and tutorials as a guide.

One of the major risks of this project derives from the lack of software backup available. Developing an entire mobile application alone can lead to the loss of data due to human error. The chance of forgetting to create backups of the program is very common. The recommended maximum time without backing up your important files is about one week and the suggested frequency is every 24 hours [33]. Losing data on the project would most likely halt, delay or end the project. To minimize the risk, various other storage devices have been used to store a copy of the code (laptops, GitHub, USB drives) so that in the case the main personal computer collapses, the project can still be retrieved and restored. This type of risk can only be minimized and not completely neutralized as there is always a possibility of having your project's main supporting platform, in this case Unity, collapsing in which case there would not be any other remedy than to either wait for a solution from the company's end or to swap engine's such as to unreal-engine. In both cases, the project would have to be shutdown momentarily causing large scale consequences on the expected delivery date.

Working alone does not only affect the potential risk of data loss, but also the risk of clouding my assessment of unit components. Having several colleagues working on the same project can be beneficial because it forces the developer to put more effort into his work as he tries to match or exceed colleagues expectations and/or assessment requirements. Self-assessment can lead to misjudging of the quality of work and also more coding errors are bound to be found later on as code is not checked as thoroughly as when reviewed by a peer.

The game is still in development and there are no security measures implanted in the software, therefore its release at this point would result in players exploiting all the bugs encountered that might benefit them both inside the game and outside the game. Before releasing, cyber security measures would have to be implemented to prevent infringement of the EU Data Protection Regulations [34] to minimize the possibility of outside the game implications. As for inside of the game possible solutions for the player vs player conflicts and bugs exploitation could be a report system where players are allowed to report issues concerning the game such as bugs and technical difficulties as well as specific player concerns regarding unsportsmanlike behaviour. These reports could be viewed by a response team and appropriate actions could be taken after careful revision.

Another risk is my own inexperience as a game developer. Never having carried out such an extended project could have made me unorganized and to have irregular system developments. Unit components of the game could have been completed in unpredictable time periods possibly causing delays in the overall delivery of the product. This risk has been minimized by the project management techniques discussed in the 'Project Management' chapter.

Chapter 8

Legal, Social, Ethical and Professional Issues

The game will be oriented towards audiences over 16 years of age. This has been based off of the Pan European Game Information (PEGI) rating system which is the most used system across Europe to assess a suitable age range for video games. The part of the current description of games rated 16+which the project should be concerned about is "May contain violence showing graphic and detailed depictions of brutality, death or injury to unrealistic humans or animals" according to the PEGI official website. Since the genre's main idea consists of players from opposing teams acquiring resources by slaying the opponent's characters and structures, and finally the enemy base structure, some form of violence will be involved in the animation of the game for realistic effects. Furthermore, the age of 16 seems to be the most suitable for MOBA's because it is a highly strategic and skill based game where a minimum level of maturity and understanding will be necessary for decent game-play.

Another organization that controls the classification and censorship of game scenes is known as 'Entertainment Software Rating Board' (ESRB) [35]. Their rating system is as follows:

- Everyone: games that are appropriate for all ages with the allowance of very 'mild violence' and/or 'mild language'.
- Everyone 10+: games that are appropriate for audiences of age 10 or older with the allowance of mild violence and/or mild language and 'mild provocative references or materials'
- Teen: games that are suited for audiences of age 13 and above due to the inclusion of 'simulated gambling', blood and/or the use of stronger language.
- Mature: games that are suited for audiences of age 17 and above. The game scenes might include intense violence and/or sexual content.
- Adults only: the 'Mature' and 'Adults only' rating are similar for the exception of allowing real currency to be used in the latter.
- Rating pending: this rating is given to physical games where the rating of a game has not yet been decided by the organization

Within these classes, Glyn would fall into the category of 'Teen'. Whilst there is killing involved in the game, there is no visual representation of such action and no sexual content has been included in the game so far.

In terms of ethical content issues, the testing phases will include a small number of participants from which their personal information will be gathered but will not be disclosed under any circumstance due to the Data Protection Act enforced by the UK government. The results presented in the final report will be made anonymous. Since the testers will be mostly from within my social circle, a formal consent will be unnecessary to carry out any experiments.

Chapter 9

Testing

Testing of the product was carried out by having players from my social circle participate in the game and then filling a questionnaire. A total of 10 participants undertook this test. The test consisted in the subjects moving a character in the world and carrying out specific tasks and experiencing a single player mode by using a personal mobile device. These tasks included:

1. Rotate the camera and observe the world
2. Move the character and observe the world
3. Move the camera and the character simultaneously and observe the world
4. Displace the character into enemy territory
5. Buy at least 2 items in the shop and try to use them
6. Shoot projectiles and damage enemies.

Tasks 1 and 2 were included to allow the respondent to get use to the controllers of the game. Note though that the experiment did not indicate the respondent how to complete the task. This was in order to verify the intuitiveness of the gameplay components.

Task 3 had to be carried out to showcase the ability of independent rotation and the effect it had on the player. Observing the world is emphasized repeatedly as the survey involves questions about the visuals and the performance of the game in terms of frame rate or as indicated in the survey 'fluent'. This is a small typing error which was meant to say 'fluid'.

Task 4 was to allow the respondents to see how the turret AI worked and interacted with their character. Task 5 was to showcase the shop and the list of items available to them as well as to check the intuitiveness of the item buttons. Finally but not least, task 6 was included to showcase the combat side of the gameplay as this was another section of the survey.
The survey they were given to complete is in the link provided in the references section [36]. Some results can be observed in figure 9.1 and figure 9.2.
Other survey questions were more open and allowed the respondents to develop their thoughts more. These types of questions were as follows:

- What is the main reason for your level of satisfaction with the visuals?: Most answers were concise and focused on the lack of visuals representing the character which made the game feel void despite the good visuals for other areas of the project.
- What is the main reason for your level of satisfaction with gameplay?: Once again, the respondents explained what the game was lacking and the reason why they rated the gameplay the way they had. The gameplay was good overall however, the most underwhelming section was the camera movement. According to the survey, 4 out of 10 people rated the camera

3. Please rate the following statements regarding Glyn's visuals. Create Chart

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree	Response Total
Visuals for characters are well designed	0.0% (0)	60.0% (6)	30.0% (3)	10.0% (1)	0.0% (0)	10
Visuals for world environment are well developed	0.0% (0)	0.0% (0)	0.0% (0)	80.0% (8)	20.0% (2)	10
Visuals for actions are intuitive and good	0.0% (0)	0.0% (0)	70.0% (7)	30.0% (3)	0.0% (0)	10
Visuals for terrain are good	0.0% (0)	0.0% (0)	20.0% (2)	70.0% (7)	10.0% (1)	10

Figure 9.1: Survey results on visuals

5. Please rate the following statements regarding Glyn's gameplay. Create Chart

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree	Response Total
The camera movement is fluent and intuitive	0.0% (0)	0.0% (0)	40.0% (4)	60.0% (6)	0.0% (0)	10
The character's movement is fluent and intuitive	0.0% (0)	0.0% (0)	10.0% (1)	90.0% (9)	0.0% (0)	10
The character movement control is intuitive and easy	0.0% (0)	0.0% (0)	0.0% (0)	70.0% (7)	30.0% (3)	10
The character actions are intuitive and easy	0.0% (0)	0.0% (0)	0.0% (0)	100.0% (10)	0.0% (0)	10

Figure 9.2: Survey results on gameplay

movement as neutral and stated in the comments that the camera movement felt 'laggy' and 'clunky'.

- What changes/improvements would you like to see in Glyn in future patches?: This comment section was more varied and had more personal preferences and ideals of what the game should feel like. Some suggested simple changes such as increasing the speed of the character movements or making some objects bigger to make them more visible such as the character's projectiles. These are easy to fix and adjust as public variables for all these details was included during coding. The seemingly more experienced respondents responded more elaborate answers including the inclusion of multiplayer mode by client and server connection and other ideas such as a level system where the characters level up as the game progresses.
- Please feel free to add any further comments regarding Glyn in the field below: This comment section was mostly skipped by most respondents.

Chapter 10

Evaluation and conclusion

Many of the test results show positive feedback in most of the areas. It is a safe assumption that the survey respondents were biased because they were from the social circle of friends. Nevertheless, the results show clearly which sections of the game require most mending and attention.
The depth of the character visuals affects greatly the impressions of the game on the customers. Noticeably, regardless of the good ratings obtained in several sections of the visuals multiple choice matrix, the following comment section mostly concentrated on the negative aspects of the character visuals instead of the positive points. This suggests there exists a distribution of weightings for the visuals; where the character related visuals have a greater weighting than visuals related to environment.

The camera movement comments mentioned above show that the algorithms for camera movement are not optimal and require too much computational power to produce a smooth camera rotation.

To evaluate the completion of the project, the objectives will be reviewed and commented on depending on the level of complexity achieved on Glyn:

1. A visually appealing world environment for the player: according to test visual results 70% of respondents agreed or strongly agreed that the world decoration and terrain visuals were well designed. This suggests that the project has yielded good results in this area.

2. A set of controllable player characters: test gameplay results show that the survey participants found any character connected movement very instinctual and easy to use. Even users who were not experienced in MOBA mobile gaming founds the tasks easy to execute. In order to test this intuitiveness, realization times for how long the user takes to execute the task could have been recorded.

3. A controllable and intuitive third-person view camera setting: as suggested earlier, neutral votes can be considered below average as there is probably a bias in the population of the survey. Even though a third-person view camera was implemented, it was not the best performing camera mode.

4. Artificially intelligent objects: Not many of the comments included any mentions about the turret tracking nor the projectile tracking. The reason for this is believed to be that turret tracking is a subtle action and not many players would notice it after trying the game once. Regarding the tracking projectiles, the testing phase did include displacing the character into the enemy territory to test the turret projectiles. However, this topic was also disregarded by the respondents in most comment sections. As a self assessment on this matter, the artificial intelligent projectiles are well implemented. On the other hand, a better implementation for the turret AI's would have been to target enemies depending on status rather than distance. The reason for this suggestion is because targeting characters with high status can result in the emergence of involuntary roles in the game such as tanks and damage dealers where the

tanks buy defensive items and protect the damage dealers from turrets and other characters which would make the game more dynamic.

5. Appropriate user interface: the user interface was not part of the survey questions directly. This was not necessary as observations on the respondents' testing phase was enough to prove that the user interface was well designed as it did not take them too long to figure out how to carry out the different actions presented and no questions about the interface were asked during the testing.

6. A multiplayer gameplay: this component was a failure due to reasons already explained in project management. Evaluation from the survey show that participants were already expecting a multiplayer based gameplay as some suggested it as a possible improvement in the future.

In conclusion: 1) a third-person view camera needs too much GPU power to recalculate regularly a detailed representation of an extensive MOBA map which suggests a possible causal link as to the lack of third-person MOBA games in the top rankings of mobile gaming; 2) the character movement control is relatively easy and cheap to execute which leads to the closure that the bottleneck of mobile MOBA games is the graphic quality achievable using mobile devices; 3) game development extends to many more fields than initially anticipated and MOBA games specially require a considerable amount of workload more than indie or repetition games.

Chapter 11

Future Improvements

MOBA games are meant to be multiplayer which Glyn does not support as of yet. This is most likely the first improvement the game would have to include. To accomplish this, a client and a server would be required. These two would need to require a connection building our own networking system, for example by using the 'NetworkTransport' class [37]. The servers and the client must have the same configuration, such as the number of players allowed to connect to the server and other settings, must be the same for both.
The server would require host socket ports so that the client can connect. The IP would probably be set to null as anybody should be able to connect to the game and we do not want to limit the game to certain ip ranges. The client will have to connect to the same network transport as the server by using the server's IP address. Once connection is established, messages will be able to be sent and received between the client and the server. This might seem pointless for the current state of the game as only two players cannot be play at the same time. However, in the case a server is established and multiple users with Unity access this server, the different players will be able to interact with each other. As a simple example, imagine player 1 attacking player 2. Player 1's client would send a message to the server when the projectile collides with the Player 2's character. The server would then send messages to both player 1 and player 2 acknowledging the character hit which would lower the player 2's health.

If Glyn did end up becoming a multiplayer game, then a database of player accounts would be required. A front-end, for logging in and registering, and a back-end with the server connecting to the database would be a good idea. This would open up possible ideas for strengthening security as private user information disclosure such as emails could result in Data Protection Law infringement.

Another proposition would be to include other maps in the game with some other objectives. Although Glyn is inspired by the MOBA genre, other game modes could implemented. As mentioned in the introduction, one of the most popular genres at the current time is the 'Battle Royale' genre with games such as Fortnite, PlayerUnknown's Battleground and APEX Legends. A similar idea could be implemented on a different Unity Scene giving the players more options and not only narrowing the audience to MOBA players. The objective of this new gamemode would be to be the last player alive. The 'teams' would be disbanded and the game would become an 'all vs all' match where everyone is a foe. The map could be designed in such a way that there are imbalances. The current symmetrical layout is designed the way it is to prevent either team from having an unfair random generated advantage over the game. For a 'Battle Royale' gamemode in Glyn, certain areas of the map could have significant advantages over others leading to the classic theme of "only the strongest will survive". The longer you survive, the stronger the players would become as they are able to relocate themselves into better combat areas. Another characteristic of this genre is that it includes some sort of timer that forces the players to combat each other. This can be in multiple forms such as a ring of fire that reduces the size of the map periodically or maybe the reduction of player statistics the longer they remain out of combat. The idea of creating a new scene within the same

game would require the client server to use the 'DoNotDestroyOnLoad()' [38]. If not used, the client would shutdown and reopen every time a new scene is loaded, which can make the game look buggy and unprofessional as the transitions would not be smooth.

An aspect of the game that would probably require attention would be the design of the characters, both visually and gameplay-wise. The current visual of the player is just different coloured capsule shaped objects. As stated earlier in this report, this capsule objects can be textured and or animated with a customized avatar. This would require imported animation model files and the avatar to be mapped to an 'Animator' component attached to the capsule in order to make the avatar move when the capsule moves. The capsule would have its mesh rendering disabled which would make it invisible in the scene, whilst the avatar is visible. This could possibly lead to confusion when it comes to hitboxes of characters. If an avatar has larger dimensions that the capsule then the collider component (which is on the capsule, not the avatar) would not detect projectiles that seem like a hit. However, this should be easily manageable by either increasing the size of the capsule and therefore its collider, or by simply reducing the size of the avatar to mold into the similar dimensions as the capsule. Even though the gameplay of Glyn would not be affected greatly by the appearance of the players, the experience of the players would be completely different as the creativity and originality of the game would really be accentuated in the visual presentation.

Currently the game generates no sounds at all. As explained in the objectives section, the aim of the project is to create as much of a captivating game as possible. Sounds on games can be very expressive and many games have accomplished their player base to recognize the games through their soundtracks. Famous examples are games such as 'Super Mario Bros' and 'Tetris', figure 11.1, or even the more modern console games such as 'Wii Sports'. All of these games have soundtracks that captivate the audience and match the pace of the game depending on the scene. Super Mario Bros music changes to a more dramatic, dark and quick tempo instrumental when the enemy Bowser appears or the Tetris' soundtracks becomes speeds up the higher the level and the faster the pieces fall. These all contribute to creating an ambience and a tone to the game. Basing off of Glyn's aesthetic visuals, the game would have to gear towards a more medieval and fantasy style music background music. Another perspective of sounds would be the action sounds. As game audio designer and professor Kristine Jørgensen acknowledged in her article of 2009, 'A Comprehensive Study of Sound in Computer Games', audio in games is not solely comprised of musicality and mood-enhancers and rather 'it also works as support for gameplay, by providing the player different kinds of information that needs to be comprehended in the light of the specific context'. This can be confirmed in countless games such as FPS games where firing a gun returns the sound of guns shooting. Glyn could incorporate original sounds for certain attacks to make them sound more realistic and feedback the player on their interaction.

On another note, the current extent of the character development is minimal as there only exists one character. An improvement would be to design and create new characters so that players have a wider pool to pick from. These characters would all be distinct in some way or another starting off with main differences such as range attacks or close-combat attacks. Alongside creating new characters, the current characters' design could change into a more realistic interpretation of a humanoid character following the remarks the survey participants made.

(a) Super Mario Bros

(b) Tetris

Figure 11.1: Games

Chapter 12

Acknowledgements

I would like to thank my supervisor Ranko Lazic for his continued assistance and time throughout the period of my project. His continued feedback on my work and my own queries has been insightful and has truly shaped the project into a more defined piece of work.

I would also like to express my gratitude towards my family and friends whose support and contribution led to the successful completion of this project.

Bibliography

[1] "League of legends tops free-to-play revenue charts in 2017." https://comicbook.com/gaming/2018/01/30/league-of-legends-top-free-to-play-revenue-charts-in-2017/.

[2] "Dota 2 revenue 2017." https://www.statista.com/statistics/807617/dota-2-revenue/.

[3] "Most popular mobile games today." https://www.ranker.com/list/most-popular-mobile-games-today/ranker-games.

[4] E. Pentinnen, M. Rossi, and V. K. Tuunainen, "Mobile games: Analyzing the needs and values of the consumers," *Journal of Information Technology Theory and Application*, vol. 11, p. 5–22, Mar 2010.

[5] Z. Tai and F. Hu, "Mobile games in china: Ongoing industry transformations, emerging game genres, and evolving player dynamics," *Mobile Communication in Asia: Local Insights, Global Implications Mobile Gaming in Asia*, p. 173–190, 2016.

[6] S. S. Lim and G. Goggin, "Mobile communication in asia: Issues and imperatives," *Journal of Computer-Mediated Communication*, vol. 19, p. 663–666, Apr 2014.

[7] J. Lynch, "As nfl ratings drop again, a new internet study says young men like watching esports more than traditional sports," Sep 2017.

[8] "The highest peak viewership twitch streamers, april 2019." https://www.twitchmetrics.net/channels/peak.

[9] J. H. A. Ströh, "The esports market," *The eSports Market and eSports Sponsoring*, p. 16–66, 2017.

[10] "Mobile vs desktop usage study." https://www.stonetemple.com/mobile-vs-desktop-usage-study/.

[11] "Unity3d physics engine." https://docs.unity3d.com/Manual/PhysicsSection.html.

[12] "Forward rendering." https://unity3d.com/learn/tutorials/topics/graphics/choosing-rendering-path.

[13] "Unity terrain." https://docs.unity3d.com/Manual/terrain-UsingTerrains.html.

[14] "Unity terrain collider." https://docs.unity3d.com/Manual/class-TerrainCollider.html.

[15] "Seamless textures (realistic)." https://assetstore.unity.com/packages/2d/textures-materials/seamless-textures-realistic-105177.

[16] "Unity rigidbody." https://docs.unity3d.com/ScriptReference/Rigidbody.html.

[17] "Unity character controller." https://docs.unity3d.com/ScriptReference/CharacterController.html.

[18] "Unity on drag." https://docs.unity3d.com/ScriptReference/EventSystems.EventTrigger.OnDrag.html.

[19] "Unity late update." https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html.

[20] "Euler angles." http://www.chrobotics.com/library/understanding-euler-angles.

[21] "Gimbal lock." https://en.wikipedia.org/wiki/Gimbal_lock.

[22] "When to use quaternion vs euler angles." https://answers.unity.com/questions/765683/when-to-use-quaternion-vs-euler-angles.html.

[23] "Unity character controller move." https://docs.unity3d.com/ScriptReference/CharacterController.Move.html.

[24] "Unity vector3." https://docs.unity3d.com/ScriptReference/Vector3.html.

[25] "Unity physics gravity." https://docs.unity3d.com/ScriptReference/Physics-gravity.html.

[26] "Unity button." https://docs.unity3d.com/ScriptReference/UI.Button.html.

[27] "Unity 3d - how to add health to the player enemies." https://www.youtube.com/watch?v=RYtLG8R2_wU.

[28] "Unity coroutines." https://docs.unity3d.com/Manual/Coroutines.html.

[29] "Armor icons." https://assetstore.unity.com/packages/2d/gui/icons/item-icons-armor-138703.

[30] "Melee weapons icons." https://assetstore.unity.com/packages/2d/gui/icons/item-icons-melee-weapon-109263.

[31] "Rpg unitframes asset package." https://assetstore.unity.com/packages/2d/gui/icons/rpg-unitframes-1-powerful-metal-95252.

[32] "Copyright, designs and patents act 1988." https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/772818/copyright-designs-and-patents-act-1988.pdf.

[33] "How often should you backup your files?." https://www.datarecoverylabs.com/company/resources/how-often-should-you-backup-your-files.

[34] "The eu general data protection regulation." https://eugdpr.org/.

[35] "Esbr ratings guide." https://www.esrb.org/ratings/ratings_guide.aspx.

[36] "Survey on glyn." https://www.smartsurvey.co.uk/s/PSWDL/.

[37] "Unity network transport." https://docs.unity3d.com/Manual/UNetUsingTransport.html.

[38] "Unity dont destroy on load." https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html.