ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΑΝΑΠΤΥΞΗ ROLE-PLAYING ACTION ΠΑΙΧΝΙΔΙΟΥ ΓΙΑ ΣΥΣΚΕΥΕΣ ANDROID ΣΕ ΠΕΡΙΒΑΛΛΟΝ 2D ΜΕ ΧΡΗΣΗ UNITY.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εισηγητής: Πέτρος Γιαννακάκης ΔΕ838

Επιβλέπων: Καπανταϊδάκης Γιάννης

©
2022

HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF MANAGEMENT AND ECONOMICS SCIENCE

DEPARTMENT OF MANAGMENENT SCIENCE AND TECHNOLOGY

DEVELOPMENT OF ROLE-PLAYING ACTION GAME FOR ANDROID DEVICES ON A 2D PLANE USING UNITY

DIPLOMA THESIS

Student : Petros Giannakakis DE838

Supervisor : Kapantaidakis Yannis

©
2022

Υπεύθυνη Δήλωση *: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Διοικητικής Επιστήμης και Τεχνολογίας του ΕΛ.ΜΕ.ΠΑ.*

ΠΕΡΙΛΗΨΗ

Το θέμα της πτυχιακής είναι η δημιουργία ενός παιχνιδιού μονού παίχτη και υπόδησης ρόλου, για συσκευές Android σε δυσδιάστατο επίπεδο, χρησιμοποιώντας την μηχανή παιχνιδιών Unity μεταξύ άλλων. Ο κύριος σκοπός της πτυχιακής είναι η γενική κατανόηση της μηχανής παιχνιδιών Unity και πως να χρησιμοποιείται, καθώς επίσης η λογική και το σκεπτικό πίσω από τη κατασκευή όλων των σταδίων του παιχνιδιού. Στην εργασία επίσης συμπεριλαμβάνεται επεξήγηση και η λογική που χρησιμοποιήθηκε στον κώδικα. Επιπλέον το παιχνίδι χρησιμοποιεί ένα σπάνιο τρόπο μάχης, στον οποίο θα γίνει λεπτομερής εξήγηση, ο οποίος είναι εμπνευσμένος από ένα άλλο παιχνίδι και είναι δημιουργημένο από την αρχή. Τα εργαλεία που χρησιμοποιήθηκαν για την κατασκευή του παιχνιδιού είναι: Μηχανή παιχνιδιών Unity σε έκδοση 2020.2.7f1 για την δημιουργία και την απόδοση γραφικών, το τρέξιμο του κώδικα και την οριστικοποίηση της κατασκευής του παιχνιδιού. Για τη σύνταξη του κώδικα και την αποσφαλμάτωση του, χρησιμοποιήθηκε το Microsoft Visual Studio 2019. Για την επεξεργασία ήχου και βίντεο χρησιμοποιήθηκε το Video Pad Editor και για την επεξεργασία εικόνων, στοιχείων και λεπτομέρειες για γραφικά χρησιμοποιήθηκε το photoshop. Η γλώσσα προγραμματισμού που επιλέχτηκε είναι η C# (Σι-σαρπ) της Microsoft.

Λέξεις Κλειδιά : Κώδικας, Γραφικά, Παιχνίδι, Μηχανή, Android

ABSTRACT

The subject of this thesis is about the creation of a single player, role-playing action game for android devices in 2-dimensional (2D) plane, using the Unity Game Engine among other tools. The main goal of this thesis is to get a general understanding of what Unity Engine is and how to use it, as well as examinations and explanations behind the thought process of all the stages as the game creation goes through. Also included in the paper, there will be explanations behind the logic used in the creation of the scripts. The game also uses a rare battle system that is going to be explained in detail, which was inspired by another game and is built from scratch. While working on the game, several tools were used. Unity Engine version 2020.2.7f1 was used for rendering, texturing, running scripts and finalizing the build, Microsoft Visual Studio 2019 was used for writing the scripts and debugging the code in the scripts, Video Pad Editor was used for the editing of sounds, audio and video, photoshop was used for the editing of images, sprites and textures and lastly, the programming language of preference was Microsoft C# (See-sharp).

Key Words: Scripts, Graphics, Game, Engine, Android

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

2D – 2 Dimensional : It means that something is depicted in two dimensions. Whether it is on X and Y plane or X and Z plane or Y and Z plane.

3D – 3 Dimensional : It means that something is depicted in three dimensions. Everything is 3 dimensional. A human sees in 3 dimensions, most games are depicted in 3 dimensions now. The third dimension adds depth to a picture.

AI – Artificial Intelligence/Artificial Intellect : It means that something works like it has some kind of intelligence but its mostly man-made logical arguments run by a machine.

BBEG – Big Bad Evil Guy : This is a nickname that main villains usually have in games when they don't have an identity yet, meaning a name or an image. It symbolizes the villain to acknowledge his existence before he is "brought to life".

CPU – Central Processing Unit : This is the main part in the computer that handles the logical and numerical instructions and controls input and output.

DAU – Daily Active Users : It's a measurement to help count users that are actively using something on a daily basis.

EDSAC – Electronic Delay Storage Automatic Calculator : It's the first computer that featured electronically accessible memory.

ENIAC – Electronic Numerical Integrator And Computer : It's the first electronic computer that was programmable.

EULA – European User License Agreement : It's a complete European terms and conditions agreement regarding license for using something that has been copyrighted.

GPU – Graphics Processing Unit : It's a specialized processor besides the central processor (CPU) that handles graphics

HDD – Hard Disk Drive : It's a type of long-term storage device used in computers to save files and other programs.

HTML – Hypertext Markup Language : It's a language for encoding web pages and a formatting system for displaying files retrieved over the internet (.html files)

HTML5 – Hypertext Markup Language 5 : It's a better version of the HTML.

HZ – Hertz : It’s a measurement to measure frequency. It is used to describe how fast and how many times something happens, over the course of a specific time. For example a monitor that can handle displaying 120 frames per second has a frequency of 120hz.

IDE – Integrated Development Environment : It’s a software for building applications that combines common developer tools and provides comprehensive facilities to computer programmers for software development.

IOS – iPhone Operating System : It’s the operating system used in iPhones.

MB – Motherboard : It’s the main component on the computers that connects everything together and brings use to otherwise useless components (CPU, GPU, RAM, MEMORY, PSU and other less major components like north/southbridge, video cards and sound cards connections etc.)

MOBA – Multiplayer Online Battle Arena : It’s a type of game category that usually pits two or more teams in a competitive set with a mutual goal to win against the other whether it is reaching an end-point or attaining a certain objective or destroying something.

NES – Nintendo Entertainment System : It’s a game console that Nintendo released in 15 July 1983.

Nm – Nanometers : It’s a measurement based on the meter scale. 1nm is 1e-9m (0.000000001m).

NPC – Non-Player Character : Is a character that is handled by the computer, or rather whatever logic the developer adds to it. An NPC is handled by AI.

OS – Operating System : It’s a system software that manages the computer hardware, software resources and provides common services for computer programs.

OXO – Naughts and Crosses : It is considered the first electronically displayed and electronically controlled game made for the EDSAC in 1952 by Alexander S. Douglas. Its name is derived from : Naughts (not’s) meaning Null or Zeros and Crosses meaning the X.

PC – Personal Computer : It’s a multi-purpose microcomputer that is feasible for personal use.

PSU – Power Supply Unit : It’s the unit that supplies power to the several components that a computer uses.

RAM – Random Access Memory : It’s a short term memory that helps the CPU, GPU and Disk Drive access some things faster for better response times.

RPG – Role Playing Game : It’s a type of game category that usually immerses the player in the role of the main character.

SNES – Super Nintendo Entertainment System : It’s a game console that Nintendo released in November 21 in 1990

SSD – Solid State Drive : A better version of the HDD. It’s an improved type of long-term storage device used in computers to save files and other programs.

VR – Virtual Reality : It’s the use of computer modeling to create an environment that feels very similar to the real world and to create a simulation of an artificial 3-D space that enables a person to interact with, by using visual or other sensory equipment.

XML – Extensible Markup Language : It’s a simple text based format that is used for representing structured information like documents, data, books, transactions and a lot more.

μm – Micrometers : It’s a measurement based on the meter scale. 1nm is 1e-6m (0.000001m).

MY GRATITUDE

I would like to thank a few people. First, I want to thank Mr. Kapantaidakis for supporting the idea of making a game and helping out immensely with my first steps of developing it. I would also like to thank all my Professors from the 5 years of studies and classes I took, that gave me the knowledge I needed. I would also like to thank my parents for supporting me financially during my studies and my friends (online or not) for being there with me.

I dedicate my first game to my dog, Oskar, who is unfortunately not with us anymore and whom I loved with all my heart and literally went above and beyond for his every needs.

CHAPTER 1

INTRODUCTION

1.1 Summary of the Game

The game has not been named yet. The games categories are mainly classified as a Single player, 2D Role playing game (RPG) and secondary categories are Adventure, Top-Down and Story rich. It is mainly dialogue focused but cannot be classified as a Visual Novel category due to other systems that are implemented in the game, such as battles, enemies, stats and a unique battle system that is built from scratch. The game has a small story that revolves around the main character, Karp, his family and the BBEG that the main character has to face later on in the story and the characters of the game are the main character, his grandfather, his mother, the assistant of the grandfather and the BBEG.

1.2 Incentive

Lately, most major game companies have resorted to making online multiplayer games that pit teams and players against each other. This isn't necessarily bad, but the natural competitiveness of these games most of the time frustrates the players and the gaming experience is not overall enjoyable. So, with that in mind and inspired by "old school" games like Pokémon, I tried to combine "old school" gaming - by making the game in a top-down, 2-dimensional (2D) plane – with "new school" gaming, by optimizing the game for android devices, which is considered as a "to-go" platform and preferred for "lightweight" single player games in the latest years. But it's not the only reason why I chose a single player type of game. Another reason why, is that there are not a lot of new single player games lately and even less single player games that are actually worth playing. Among that, I have opted for a roleplaying type with a story rich adventure that I believe is going to bring back the fun and relaxing time going through the story at each ones pace along with some easy battles, to keep the "boring" factor from too much dialogue reading in check.

1.3 Goal of the Project

The purpose of this project is to study what is needed to create a simple game, which in the current scenario is an RPG, story rich, single player 2-dimensional game. In this paper I am going to analyse all the assets and aspects needed to make the game fully functional, as well as the logic behind the scripts created, what to keep in mind while creating said scripts, where to find and how to use assets, optimization for different resolutions and aspect ratios and different devices such as IOS, Android, Windows etc. and what steps the creator needs to follow, as well as when to take each step, for a smooth construction of the game with as little bugs, errors and optimization mistakes as possible. The goal is for this paper to serve as a tutorial on game creation for beginners on both the theoretical part and the practical part.

1.4 Game in Detail

1.4.1 Story

The story of the game is a typical Good Guy – Bad Guy relationship, where the bad guy hurts the main character and what he holds precious so the good guy hunts and fights against the bad guy.

1.4.2 Stages

The game is made with four (4) stages. The tutorial stage, where everything about the game and game controls are being explained, the introduction stage where most characters and story are introduced, the battle stage which contains a tutorial about the battle system as well as the last battle with the bad guy and the last stage which is a last dialogue between the main character and the bad guy as well as the finish screen.

1.4.3 Characters

The characters included in the game, with their names are:

Karp – Main character

John – Grandfather

Jonas – Assistant of grandfather

Joshlyne – Mother of main character

David-Ivan Okoniewski – Main Villain

1.4.4 Battle System

The battle system is a unique way of using attack moves and healing moves unlocking a lot of potential for other uses as well, such as dodging, combos or other strategies during fighting. The system uses the general skill-cooldown-skill cycle but in a different manner. There is a bar that represents the cooldown period of a skill, a text to represent what the skill is/does and a button to trigger the skill. The difference between this battle system and the more general battle system is that normally, the player uses one skill at a time and then the skill starts the cooldown, however in this, the player can trigger more than one skill at a time and all cooldowns work together. Main difference is that everything is parallelly running instead of the usual use-wait period.

1.5 Summary of the chapters that Follow

In the chapters that follow, I am going to talk about the history of games and consoles (particularly Android platforms), their evolution and evolution patterns in the years, what factors affect and how they dictate evolution, strengths and weaknesses of different consoles and gaming platforms, the programs I used to make the game and the logic behind the code.

CHAPTER 2

HISTORY OF GAMES AND PLATFORMS

2.1 Games in Different Eras

Games have been deeply implemented in everyone's life, since forever. It does not really matter if it's a single player game, a multiplayer game, online or offline, digital (e.g., computer game) or non-digital (e.g., tabletop games), or whether mostly adults or kids play it. A game exists to bring entertainment and fun to people. In previous eras, games can still be found in people's lives, such as Nintendo's Game Boy in the 2000s that featured a cable that allowed multiplayer, or even earlier - when computers did not exist- kids would play different games such as tabletop games, or games that included physical activities such as hide and seek and tag or other sports like football and basketball. In the current era, most games are digital and not a lot of games are played non-digitally. Even most tabletop games have a digital version made of them and the amount of tabletop games that still require a physical presence and handheld materials, such as pen and paper, are gradually getting less and less as technology continues to evolve.

2.2 History of Old Game Platforms

Whether the platform was designed for it or not, more and more platforms that support games have started to emerge. From dedicated consoles for games, to computers, to mobile phones every platform has its own strengths and weaknesses, and every platform has had its own trending phase. The first ever commercial game console created, as we know it today, was the Odyssey Magnavox and was released in 1972 and was developed by a German-American engineer named Ralph Baer *Wolf, M.J. ed., 2008*. The Magnavox featured a ping-pong style gameplay and inspired Nolan Bushnell to create the American Atari 2600 which was released in the year 1977 along with his own version of the game called *Pong* that ended up being more popular than the Magnavox. However, as time continued to move forwards and technology evolved at a rapid rate, more specialized and better at handling graphics consoles were released such as the Atari 2800 that was released in 1983, the Japanese Nintendo Entertainment System (NES) that was also released in 1983 and the Japan-only arcade console of Sega, the Sg-1000 that was also released in 1983, but later in 1985 was rebranded as Master System and was released to the whole world. These very early game platforms were most of the time game dedicated consoles that could play one game only, or just a very few and very specific number of games. As time went on, consoles only got better with handling the quality of graphics and the ability to play more than just a few very specific games. Most notable platforms in the 90's was Nintendo's SNES (Super Nintendo Entertainment System), the Sony PlayStation 1 that was released in 1994 which

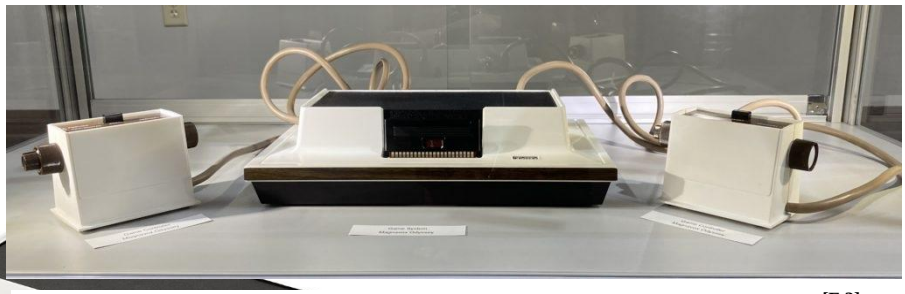


Figure 2: Magnavox Odyssey on the Computer Museum of America [F.2]

Figure 1: Magnavox Odyssey Console Set [F.1]

Figure 3: The Original PlayStation 1 Console [F.3]

was one of the few consoles that could handle 32-bits back then and used CDs for games instead of cartridges, the Nintendo 64 released in 1996 and Nintendo's Gameboy Color, the first in the Gameboy series, was released in 1998 and featured a rechargeable battery which made it one of the first consoles that could play multiple games as well as being portable, the way we know it today *Kowert, R. and Quandt, T., 2015*.

2.3 Modern Game Platforms

So what game platform is modern then? Technically, according to the words [L.1]definition on the Cambridge Dictionary, one can figure out that different game platforms are modern in different eras which renders the question as a hard to answer.

2.3.1 Factors

There are usually some factors most reviewers and companies generally use that dictate how game platforms are rated and most of the time a few of those factors are :

-Affordability, meaning how cheap or expensive the machine is.

-Processing power, meaning how fast the machine is at processing various elements like code and graphics.

-Flexibility, meaning how many different games and how many different controllers or sets of controls the machine can handle

-Mobility, meaning how easy it is to access and/or use the machine at any given time.

-Versatility, meaning how many different things you can do on the machine apart from gaming.

By studying these factors, patterns in the evolution of game platforms can be identified. By identifying those patterns, one can easily reach a conclusion on which platforms are considered modern.

2.4 Evolution of Game Platforms

Figure 4: Retro Arcade Machine, specifically designed to play the game "Pac Man" [F.4]

In the beginning, as noted before, game platforms could run specific games only. That means their Flexibility and Versatility was awful since they could have one function only. Their processing power was also lacklustre, as the technology was not very advanced at the time and since they were big and hefty too, it meant that they were expensive to make and hard to move around which means their mobility and affordability was extremely unsatisfactory as well. This made them very rare and for a normal customer to have, was quite unlikely, but a few could be found in what was then called, “Arcades”. They are tall box-like machines that house the screen, controls and the circuits and boards. They usually operated using coins, they could only play one game and they are the earliest form of stand-alone consoles. *Wolf, M.J. ed., 2008*

However, as technology progressed, the machines got progressively better in all aspects. First came Flexibility as development was initially focused on getting more games on one platform. Then, as engineers started to build more compact consoles to handle and combat high temperatures and heat-spread from increased power, machines started getting smaller and smaller, which made mobility drastically improve and as a by-product of that, affordability also improved since computers required less material to be made. This was good for the average consumer because this meant he could actually afford it but even with all that effort, processing power was still not enough for 3-dimensional graphics all the way until 1980 when Battlezone, the first 3-dimensional game was released to the market. Versatility came last when PC’s started getting attention because of their stronger processing power when compared to a console.

2.5 Old and New Platforms

For older platforms, I believe it's wise to note the first ever computer that utilized electricity here, as I believe it highlights every problem with the very early computers. The ENIAC machine, which was made by John W. Mauchly and J. Presper Eckert at the University of Pennsylvania. The machines' name is an acronym, and it stands for Electrical Numerical Integrator and Calculator *Freiberger, P. A. and Swaine, . Michael R. (*2022). Of course, as the name suggests, it could only produce numerical calculations through a very complex technique of using punched cards as input and output. The machine did not have any memory as well, which means every time they wanted to calculate something, they had to redo all the wiring and replace the punched cards on the input and output. The ENIAC took around 170 square meters of space and used around 18000 vacuum tubes, more than 7000 crystal diodes, 1500 relays, 70000 resistors, 10000 capacitors and approximately 5 million hand-soldered joints.

Figure 5: John Presper Eckert and John W. Mauchly [F.5]

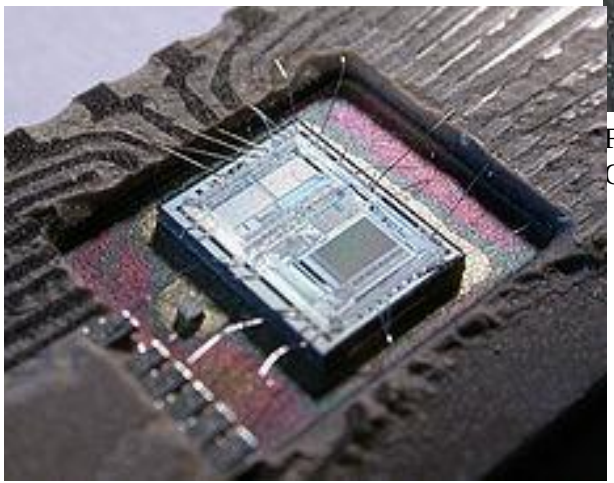
For its time, the machine was very fast and was able to solve a large amount of numerical problems but all these highlight all the problems that early machines faced, in comparison to modern ones. Enormous, time consuming to make, hard to maintain, difficult to use, extremely overpriced and could do only the one thing that was programmed to do, which was calculations and when compared to todays' calculators, it did not do a very good job at that either.

of the ENIAC on display on
nd applied Science [F.6]

Platforms quickly shrank down enormously when their respective components grew smaller as development continued. There was a huge spike of improvement on size when Intel introduced the first ever microprocessor in 1969, 23 years later after the ENIAC was invented, the "Intel 4004". That microprocessor is stated by intel to "have the same processing power as the first ever electronic computer, built in 1946" namely, the ENIAC. Its difference in size was from 170 square meters, which is around 2 average apartments in size (in comparison, a studio apartment where students usually live in, are on average around 40 square meters), to just a few

Figure 8 : The ENIAC itself and the room it was kept in [F.7]

Figure 7: Intel's first ever microprocessor the C4004 [F.8]



centimeters or the way Intel compares it, “the size of a little fingernail”. Intel then launched the chipset with an advertisement that claimed, “a new era of integrated electronics”.

Even though CPU chips as a whole, grew back in size a little bit as they evolved, the “die” as they call it and the “circuit line” went through extensive research and development in order to grow smaller and smaller with each new chipset. A good comparison with the 4004 chipset being a die width of 10 microns, or 10.000 nanometers in 1971 and 2008 when the first i7 generation launched that was at 45 nanometers die size, a whopping 22.222,22% decrease in size. After 2008 however, there was slower progress and Intel entered its famous “tick-tock” phase where one year the chipset die got smaller and another year the circuit line got improved. That meant every two years the die width got smaller from 45nm to 32nm to 22nm and lastly 14nm and when Intel got stuck, AMD capitalized on it and created its own 10nm chipset and even managed to improve it to 7nm. For reference, a study conducted by the university of San Diego of Jacobs school of engineering *Yang et al. 2019*, “a human hair is as thin as 80microns in diameter (80 microns = 80.000nm). Most companies now use the 7nm die, while some (like Intel) 5nm die. At that

Figure 9 : Die (Over

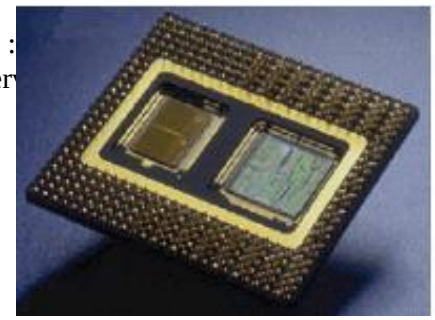


Figure 11: A complete chip with its Die (Underside View) [F.10]

Figure 10: Complete chipset (LGA775 specifically) on a circuit board (Motherboard) with the Chip fitted (Intel Pentium E2220) and its Northbridge (Small black square on the left) [F.11]

width, it does not make a noticeable difference, but a smaller die usually yields better results regarding energy requirements, power output and temperature control as electricity travels through a smaller and narrower path.

This development was similar for most components on electronics and most of these electronics are the same for most computers, consoles and other platforms. Just like today, they used a similar approach of having a processing unit (the CPU), short term memory to handle the processes that run at the time (the RAM), a processing unit that handles graphics/how things look (the GPU), a device for long term memory (the hard drive, HDD or most recently SSD), a power controller for electricity (the PSU) and a device to connect them all (the motherboard).

As these components got smaller and smaller, companies could also fit them in smaller cases, or smaller rooms while maintaining - or even surpassing - the processing output of the previous generation machines.

Another thing that highlights the problems of early technology on computers and games on them is, even though the first dedicated game platform was released in 1972 as mentioned before, the first ever game was actually released long before that in the 50s. The game was named OXO and it was the first electronically controlled and electronically displayed game, made by Alexander S. Douglas and simulated a game of Tic-Tac-Toe made for the computer EDSAC *Kowert, R. and Quandt,*

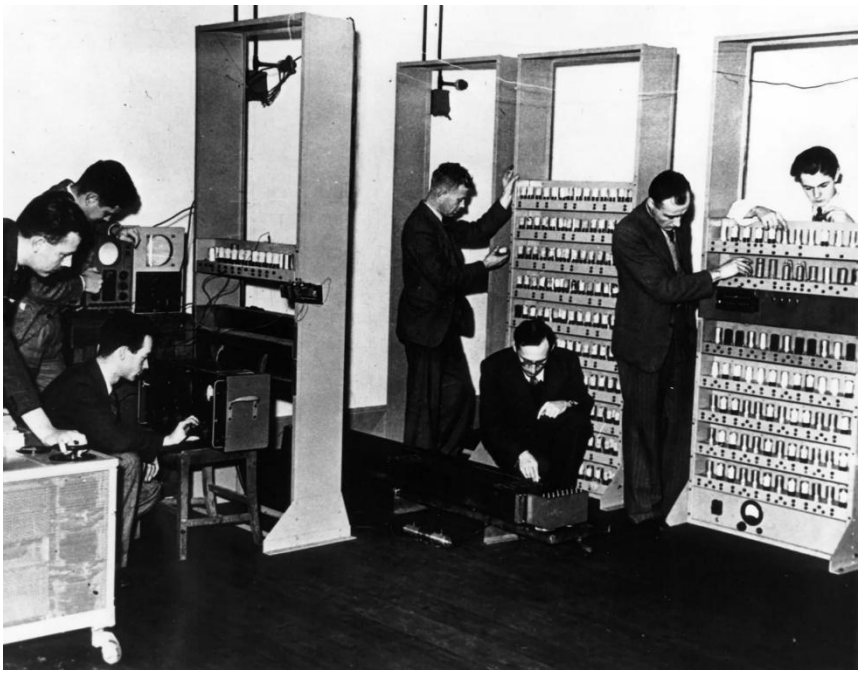


Figure 12: Maurice Vincent Wilkes and other Engineers working on the EDSAC (1947) **[F.12]**

T., 2015; however, it's generally considered that the first game was *spacewars* made in 1962 as OXO was only available to be played by the people of University of Cambridge's mathematical laboratory as the ESDAC could not be moved. It is also believed that many more games were created in the 50s but were not known as they were most likely only used in private machines such as the ESDAC *Freiberger, P. A. and Swaine, . Michael R. (*2015).

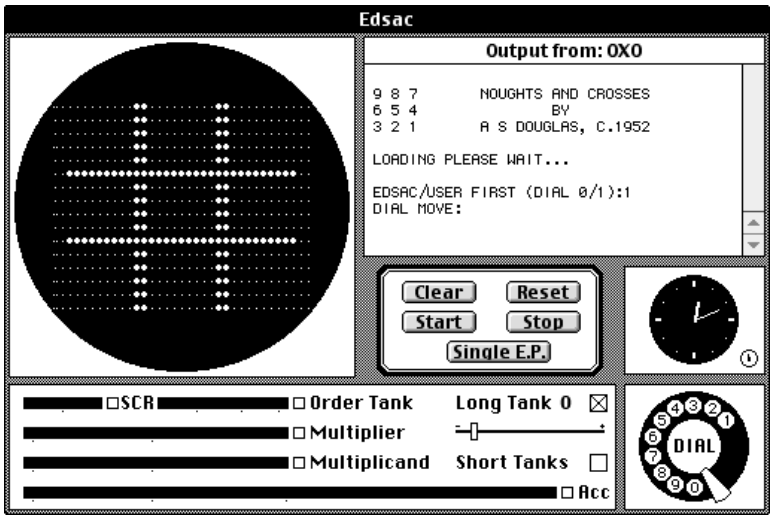


Figure 13: Interface of OXO, the TIC-TAC-TOE inspired game that was created and played on the EDSAC by Alexander S. Douglas [F.13]

2.6 Modern Game Platforms (2011-2020)

Newer game platforms when compared to older ones it's plain to see their differences. They are compact, faster, can do more things at the same time and also do a lot of different things rather than one thing only like older computers and platforms.

Finally, having a firm grasp of what modern is and how to find modern game platforms in different decades, the platforms that I would consider as modern are smartphones. This is based on the earlier factors that were mentioned along with some other considerations.

Smartphones tick most factors needed to be considered as the modern platform for gaming in the decade of 2011-2020. They are very small platforms at around 6-7 inches that makes it a pocket gaming platform which checks out the mobility factor, they are devices that cost around 300-800$ which makes it very affordable, compatible controller adaptors exists for different games that checks out the flexibility factor, there are applications that give almost the same functionalities as a desktop computer and it has powerful graphics processors and equipment that can run 3D graphics with a stable framerate around 60-120hz.

These factors essentially make it a smaller, with better mobility, laptop at the expense of some processing power and battery charge. There are also some other external considerations

to be taken into account, as the smartphone also acts as a mobile phone which the biggest percentage of first and second world population uses and it's a feature that no other gaming platform utilizes.

Smartphones also have had their trend-phase as a game platform especially when someone looks high quality games like clash of clans (released in 2012), Pokémon go (released in 2016) and PlayerUnknowns battleground Mobile (released in 2018), which only solidifies its position as a modern game platform.

2.7 Pokémon Go key history

Figure 14: Current Pokémon Go logo as taken from the site of Pokémon [F.14]

In its own right, Pokémon Go is a game that has reached many milestones that place it above other mobile games. When it was released in 2016, according to Screen Rant, it reached a whopping 147 million downloads within its first month, 75million downloads within its first 19 days, reached #1 in downloads on the Google App Store within its first week of release. However, Pokémon Go only reached its peak daily active user in 2018 with 45 million daily active users. The reason for the late DAU peak is because Niantic, the company that released Pokémon Go, did not expect such a high number of users. This meant Niantic did not prepare appropriately and the servers that hosted the online game could not handle such a huge player base which, in turn, meant many people experienced a lot of bugs and a lot of lag that ruined the game experience and many people stopped playing for that reason. Niantic quickly got on the problem and upgraded its servers which saw many players coming back to the game, unfortunately however, some people (including myself) that jumped off the “hype train” of the first 2-3 months never came back. This did not slow down Niantic at all though as analytics firm Sensor Tower estimated an average of $4million daily revenue in 2019 and this put Pokémon Go as the game that reached the $600million revenue mark the fastest, which did it in 90 days.

Within the first 3 to 6 months of the release of Pokémon Go, the game became so widespread and known that even the news everywhere talked about the game and how young people filled the streets walking up and down chasing Pokémon, meanwhile the players were so into the game that many reviewers and other gamers jokingly claimed that it was “the closest thing we had to world peace”.

Even with all these positives of the game, the competitive side of Pokémon Go with Gyms and Battles started taking over the game and things started going south at a rate of knots. Some players got so obsessed with the game that soon after its first months, instances of violence, carelessness, recklessness and negligence started occurring. Things started going so out of control that injuries related to Pokémon Go started being recorded and a lot of times it even resulted in death in one of the two parties or sometimes both parties. These instances include players beating up each other over Gym Ownership/Leadership, other times players would pay too much attention to the game while driving and injure/kill pedestrians. Specifically a death related to Pokémon Go was recorded when a man, that ended up drowning, went in the sea to catch a rare Water type Pokémon during a very rough tide.

This section dedicated to Pokémon Go shows exceptionally well, how much and how efficiently a mobile game can connect players. It also shows how much potential mobile games can have but it also shows the dangers that lie sneakily within such games in the mobile gaming community. Pokémon Go can be a incredible baseline for future games, both in positives and negatives.

CHAPTER 3

ANDROID

From personal knowledge and further research that I conducted, I came to the conclusion that in 2020 and onwards, the modern platform for games are smartphones, particularly android devices therefore it's the platform I chose to implement my game on.

3.1 General info about android

Android is an operating system for mobiles. The project was "born" in 2003, it was created in November 2007 and it was released in a phone for the first time in September 2008, with the HTC dream. It was created based on a Unix-Like OS family which is a modified Linux Kernel.

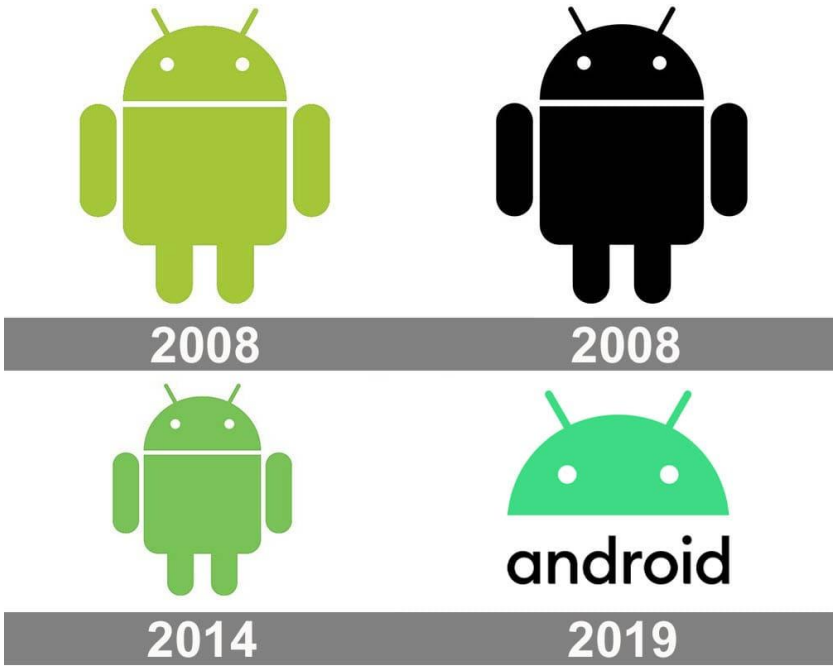


Figure 15: Evolution of the Android Logo over the years [F.15]

Android is an open-source code that is written in many different languages[L.8] including Java, C, C++, XML, Python and many others. As of 2018, android contains 6.714.784 lines of Java code, with it being roughly 43.1% of the total code and C following with 5.162.285 lines of code totaling at roughly 30.1% of the total code. With C++ being third at 2.164.433 lines of code and 14.5% of the total code, XML being fourth at 1.409.418 lines of code with 8.7% of total the total code and the rest of the languages all being below 1% each, its safe to say that Android is comprised of mostly Java, C and C++.

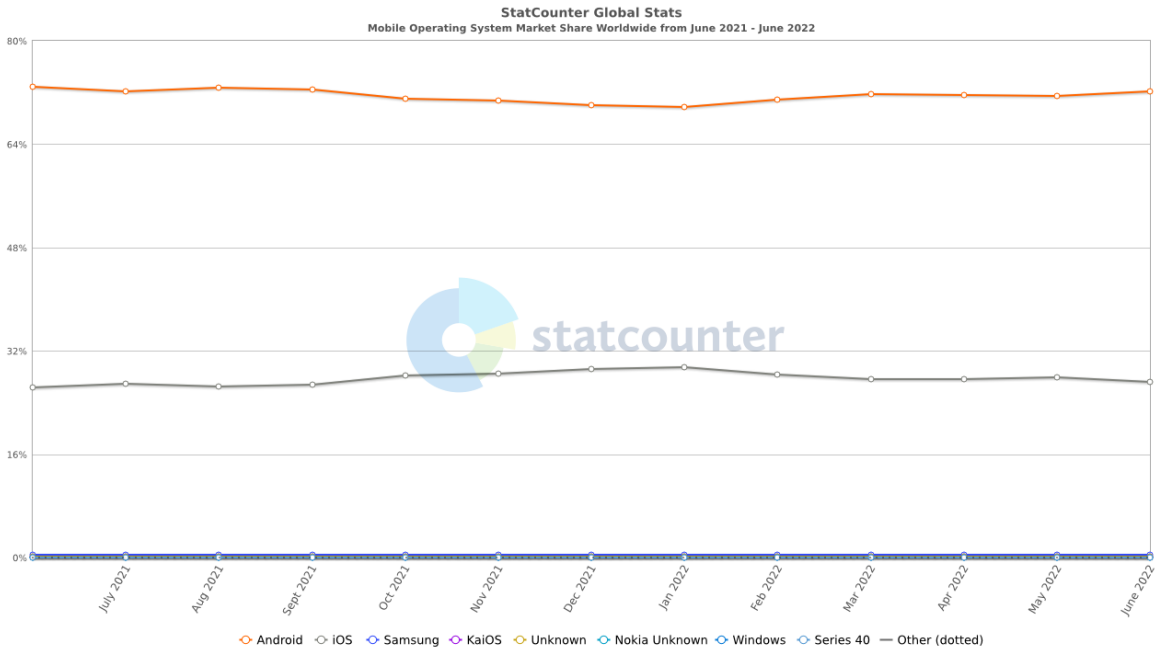


Figure 16 Diagram that shows the worldwide percentage of Mobile Operating System market share that each company holds. Data are from June 2021 to June 2022. **[F.16]**

Android, with its Android 12 release on 4th of October 2021, held a 71.09% of the total market share in the Smartphone OS market and by the end of June 2022 it held a 72.12% of the total market share in the Smartphone OS market. Android also holds the biggest share in the electronics market with 39.75% of the total OS market in October 2021 and a 44.15% of the total OS market by the end of June 2022 with Windows OS coming second with 32.44% and 29% in October 2021 and June 2022 respectively. This rising percentage in the android and OS market is mostly due to how common phones are and how common smartphones have become, all while android being open-source so most companies develop and use their own version of android-based OS, like Samsung using the name “Snow Cone” for their android 12 version, LG using the LG UX 10.0 for their android 11 version and One Plus using the name “OxygenOS 12” for their android 12 version etc.

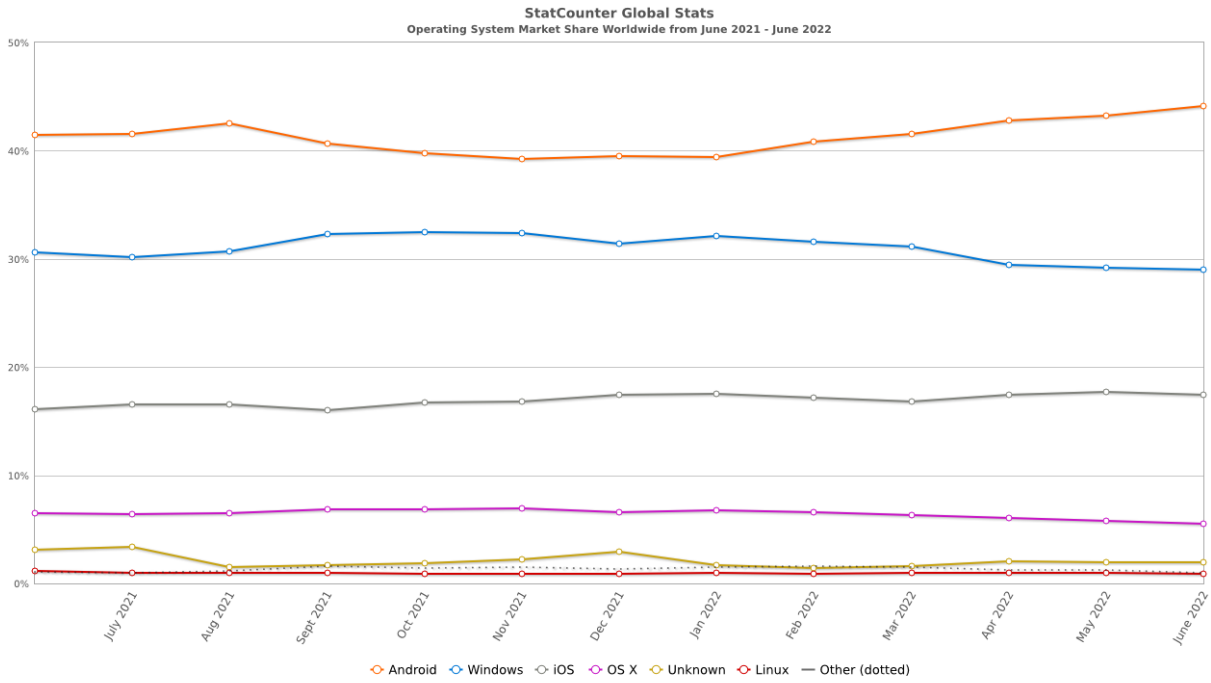


Figure 17: Diagram that shows the worldwide percentage of all Operating Systems market share that each OS holds. Data are from June 2021 to June 2022 [F.17]

The fact that android holds this big of a market share (almost half the market share!!), means that there will be a lot of apps that are intended for and targeted on android OS. With google play store being the main place as well as the safest to download apps, there are also plenty of other App Stores to download apps from, like APKMirror, Apptoide and Amazon Appstore among others being the some of the best alternatives to Google Play Store.

3.2 Google Play Statistics

Starting off with some general statistics, google play, according to research by analytics firm 42matters, had a total of 3.520.511 apps in its database by July 2022 however, analytics firm AppBrain state that google play has 2.671.743 total **Android** apps in its database[L.7], by July 2022. These numbers fluctuates a lot from quarter to quarter due to the fact that Google makes a continuous effort to delete apps that are not active, or do not comply with their terms of service in one way or another. A different research, done by Statista, shows that the free apps in the database of Google varies between 96% and 97% with the rest of 3 to 4 percent being paid apps[L.11].

Analytics firm Sensor Tower has conducted their own, rather interesting and extensive research[L.12] between the first quarter of 2021 and the first quarter of 2022, on Google's Play Store statistics including total downloads worldwide, top apps, top games, top publishers and top categories along with some other findings. I am going to be examining information regarding the downloads worldwide, top games and top categories along with their market growth.

They, *Sensor Tower, (2022)*, concluded the following:

3.2.1 Worldwide app Downloads on Google Play Store

Google play store has had a total of 28 billion total downloads in the first quarter of 2021, while the first quarter of 2022 had a 1.1% growth from 2021 that saw the downloads reach a total of 28.3 billion. For the first quarter of 2022, the app with the most total downloads was Instagram, surpassing the 120 million downloads mark, with TikTok coming in third with approximately 118 million just below Facebook. However, in certain countries like the US, TikTok has around 30 to 40 percent more downloads than Instagram and around 25 to 30% more downloads than Facebook.

3.2.2 Game Apps Available in Google Play Store

Out of the 3.520.511 total apps that google play has[L.9], only 477.688 are gaming apps with the rest being non-gaming apps. This is a 13.6% of gaming apps and 86.4% non-gaming apps for the total apps.

Out of the 2.671.743 Android apps that google play has[L.10], 449.497 are gaming apps and 2.222.246 are non-gaming apps. This means that 16.82% is gaming apps and 83.18% percent are non-gaming apps.

Using these numbers, a conclusion can be reached that 848.768 apps are not for android and 28.191 are gaming apps made for other OS apart from Android.

3.2.2.1 Game Apps Downloads from Google Play Store

Even though gaming apps hold a small percentage of apps in google play, they boast a 11.79 billion downloads in the first quarter of 2021 and a 2.1% growth in the first quarter of 2022 touching the 12.03 billion downloads. One can see the significance of this number when compared to the second most downloaded category, being “Tools Apps” with a 2.12 billion downloads for the first quarter of 2021 and 2.53 billion downloads in the first quarter of 2022. Their difference is 475.5% more downloads in the game category! That is almost 5 times more downloads than the second most downloaded category! It is very obvious how much influence the gaming category has on smartphones, android in particular, and how much influence it will have in the future as devices and OSs evolves.

3.2.3 Top Games in Google Play Store

Top downloaded games in Google Play Store are actually very different in each country/continent and the gigantic difference in market sizes affects the results on worldwide downloads immensely.

We can see that in the US, top downloaded games are more casual and time killers, with Count Masters ranking most downloaded game for the first quarter of 2022 with approximately 2 million downloads, while in Europe top downloaded games are more adventurous and puzzles that provide a different type of entertainment with Subway Surfers reaching an approximate mark of 8 million downloads. The Asian game market has more competitive games and shooter games as their most downloaded games with Garena Free Fire ranking first with almost 40 million downloads and Ludo King with approximately 37 million downloads as well as older battle royale shooter PUBG mobile ranked among them.

Looking at the overall download statistics, results are heavily altered due to Asia having a much bigger population ([L.3]China with 1.450.478.060 and [L.4]India with 1.406.631.776 for a total of 2.857.109.836 from just two countries and a total of [L.2]4.722.743.200 overall) than the [L.5]US (332.403.650) and [L.6]Europe (748.550.855) which are both whole Continents.

3.2.3.1 Analysis of Influence of Games

Since games are not like overall Top Apps which yield more or less the same results worldwide (same Top Apps regardless of place), it would be better to make an analysis of how much influence these games have on the market.

With almost 40 million downloads, the Top downloaded App of the Asian market has 500% more downloads than Europe's most downloaded app and 2000% more downloads than US most downloaded app.

Therefore, from a raw numbers point of view, publishing a game tailored to the Asian market would be the best choice on paper. However, there are some risks that need to be taken into account. Asia has a total population of approximately 4.56 billion. That means the 40million downloads reflect the 0.877% of the total population while the 8 million downloads on Europe market reflect 1.07% of the population and the 2 million downloads in the US reflect only 0.6% of the population.

Considering these percentages, it would mean that actually a game tailored and published on the European market would carry less risk and have more chances for it to become known as a bigger percentage of the population would react. This works vice-versa as publishing a game on the US market would have more risk and less chances for it to become known as a smaller percentage of the population would react.

One last very important note about the US population, even if US had the biggest percentage out of the three, if someone considers releasing an android app there is the risk of it failing because the US market is comprised by mostly iOS with Apple App Store (approximately 68% of the market) and not Android, considering the US overall app download statistic.

CHAPTER 4

UNITY, GAME ENGINES AND OTHER TOOLS

After choosing the platform to create the game on, comes with its own set of decisions to be made. One needs to select between operating systems, coding languages, game engines or frameworks, different tools to aid design and other things that are essential as well, but not as important, such as background music, art etc.

Everything that has been used to make the game, has been chosen very carefully to fit the needs, goals of the game and target audience. From the platform and game category chosen, to the game engine and coding language that has been used to create, design, code and render.

4.1 Tools

Tools that have been used for the development are Unity for the game engine, C# as a coding language, Microsoft Visual Studio for writing the scripts used, Unity Asset store for premade art, photoshop for editing pictures, VPP video pad editor for editing music tracks for background music.

4.2 What is a Game Engine

An engine is specialized software that helps with the development of the environment of other programs into a working finished product. An engine usually contains pre-made components that are used constantly in order to save time. It's something like a software framework but instead of making each components from scratch each time, you can save them once and keep re-using them over and over while being able to make changes to their looks if needed. Engines are used mostly by games, hence the term "game engine" but it's fairly common for engines to be used from businesses and corporations to make their own specialized apps, software and programs.

A game engine is an engine that specializes in creating environments and controls for games only. They usually have components that are used in games, for example camera

control to simulate where the player looks in first person or timer controls to simulate animations etc. Game engines also include relevant libraries, other support programs and a render engine that helps render and create the necessary files so that the finished product can be optimized and working for their respective platforms and operating systems (for example different files for android and iOS).

Game engines[L.13] are important in the gaming industry as the reusable game assets and code improves productivity and makes game creation easier rather than creating everything from scratch.

4.2.1 Framework

A framework is the most basic tool of a developer. It provides a foundation for creating anything with already written code that does specific things. Using a framework saves time, provides cleaner code and reduces errors and mistakes.

Figure 18: The logo of the framework that Microsoft uses to create its own apps. Version 4.5v [F.18]

It is possible to create a game using a framework, in fact older games before game engines started appearing, were created using frameworks only. However, there were many drawbacks to frameworks being used because whenever a game was created, everything had to be re-created from scratch every time meaning they were immensely inefficient and difficult to use.

One can easily confuse a framework with a game engine as they seem closely related to each other and both perform similar tasks and activities. Just like game engines, a framework is a structure you can build software on, serves as a foundation and are typically associated with specific languages but they are not the same thing.

4.2.2 Framework V. Game Engines

With both being similar to each other, their differences are hard to see. However, when one starts working with both, there are huge differences between them. Their biggest

difference is that a framework usually creates using code however, a game engine just creates and at the end, it renders the appropriate code and files needed for the product. A game engine also provides visual aids and is a lot less complex than a framework.

4.2.2.1 Unity

The game engine I chose for my project is Unity engine. Unity was not the first, or among the first game engines but it's one of the best engines out on the market right now. It was first announced at Apple's Worldwide Development Center in 2005. The engine can export and render games for most platforms and operating systems, including Windows, Android, iOS, PlayStation, Xbox, Linux and several other game platforms and operating systems.

Figure 19: Unity logo as provided by Unity for trademark purposes **[F.19]**

It's important to note that Unity Advanced is free to use for students who are eligible for personal and educational uses while Unity is free to use for games that produce revenue or get a funding of under $100.000 in the last 12 months. Unity also provides complete tutorials both with theory and application of certain tasks which makes Unity the ideal beginners choice.

Unity has a simple interface with easy controls to move and edit objects, but it also provides tutorials not just for the basics, but for a lot more complex creations and other tutorials suitable enough for more advanced users like educators and creators or even more advanced tutorials for professionals. These Unity tutorials have a difficult task assignment at each respective tutorial that if completed, you get a Certification for completing the tutorial.

Unity was developed using only C++ as its programming language and has its own scripting language, UnityScript which its syntax is derived from and is similar to JavaScript, however it is possible to use C# for scripting and its even recommended by Unity tutorials for beginners to use C#.

4.2.2.1.1 Interface of Unity

This is the default interface of Unity Game Engine. Removing the already added scenes and what the display shows, is what Unity shows as a default when you first open it after a clean installation of the engine. The interface is very simple and pretty self-explanatory.

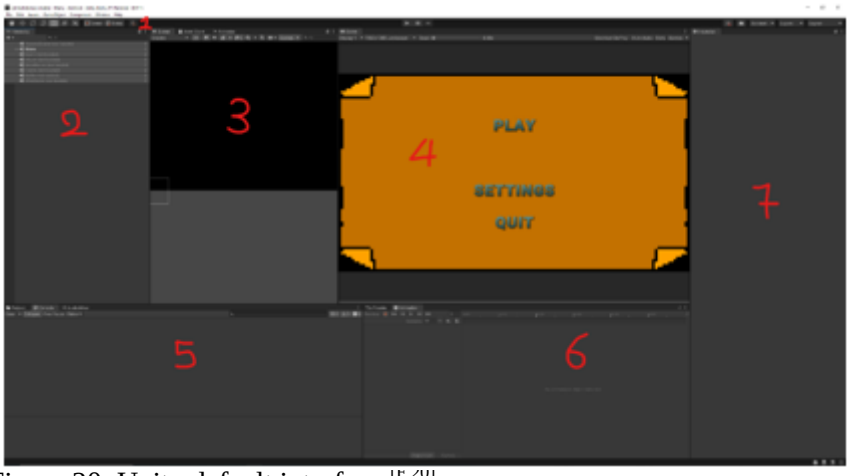


Figure 20: Unity default interface [F.20]

Unity has 7 windows opened and set by default, 1.the toolbar, 2.the hierarchy window, 3.the scene window, 4.the display window, 5.the console window, 6.the animation window and the 7.inspector window. All 7 windows can be removed, although it is not recommended to do so, but they can be re-enabled from the "component" tab on the top. Unity also provides the developer the option of opening these windows as a separate window, using the "window" tab at the top, outside the main unity window to use, for example, on a second monitor.

1-Up top, Unity provides a toolbar that can not be moved. The toolbar can be customized with different tools, although it's a bit trickier to add than others, Unity provides more options to what you can add and how. The default toolbar provides tools for quick editing, scene viewing and playing, layering, layout and pretty much what every basic and

common thing a developer will need. To add extra tools in Unity, you have to insert the code itself, which is the tricky part. However, there is more than ample community support and already pre-coded tools to insert and immediately use, error free and tested. Adding tools like this is also a big help to more advanced developers also who want to make their own tools in their own way, Unity provides the freedom for them to do as they please. Therefore it is a sacrifice of some usability for more practical uses.

2-On the left side, on the hierarchy tab (2), Unity shows all the scenes the developer has created. There are options on which scene to show and which to hide and it's also the main window where game objects of all sorts can be created by using the white "+" on the top-left.

3-The scene window is where everything shows. When the developer adds a game object on the scene, it immediately pops up on the scene view as a first preview. The editing tools can be used on the scene to move game objects around for quick creation and editing and placing everything exactly where they should be for the display to be correct. Upon the creation of the first scene, Unity creates the camera game object on its own but does not create it on the second scene. This can be slightly confusing to the developer as he is going to get just a black screen and the camera has to be inserted manually. The camera is needed to show and relay visual information on the scene and display screens. It is what the player/user sees when playing the game or using an application. Other Game Objects can be either 2d or 3d objects, or audio and video related or just folders with parents and children for better organization on the game objects, for example splitting the game objects used between two houses as to not mess up any settings.

4-The Display window is more or less the same scene window with some differences that look minor but are actually major. The difference is that even though you see the same things, the display scene is what triggers when you press the "play" button on the toolbar and anything that has been created on the scene, finally runs. It doesn't matter what has been created as long as there are no errors. If an object has been scripted for movement, then it can be controlled in the environment through movement, so it is basically like live-testing the application and how it runs, something you can not do on scene.

5-The console window, or developer console, is pretty simple. It shows the code that runs when playing the scene, if there is any. It also shows any results that might occur from the scripts that run including all outputs, warnings and errors from faulty code.

6-The animation window shows and handles animations. The most common use for the animation is the character movement as every game with a moving character needs it. Animation can be done in two different ways. One is frame-by-frame animation, meaning that for every frame an animation has to be drawn which is the easy way to setup an animation. The other way is to create what is called a "skeleton" and program the movement. A "Skeleton" animation is used in 3d movement mostly as its not that efficient in 2d. I have chosen the frame-by-frame animation for my character.

7-The inspector window displays detailed information about the currently selected Game Object, including attached components and their properties. Information about the Game Object can be an explanation of what Object is used, its dimensions relative to the scene grid, coordinates meaning where its placed relative to the scene grid etc. Other attached components can be either scripts, for example movement script or trigger scripts, different visual components like picture to portray a character or an object like a rock or fence etc.

4.2.2.2 CryEngine

CryEngine was released on May 2nd of 2002 by game developer Crytek. It was released a few years earlier than Unity and has been used in several known game titles, like Far Cry 1 and Sniper Ghost Warrior 2 among others. CryEngine, like Unity, is considered a third-party engine that can't export in as many platforms

Figure 21: Two CryEngine logos as provided by CryEngine for trademark purposes. It comes in both black and white colours to fit any background [F.21]

and operating systems as Unity but exporting options include all major platforms and operating systems such as Windows, iOS, Android, PlayStation, Xbox, Linux and Wii U.

CryEngine is completely free to anyone, for any use and no limitations, however, it's the most obscure engine out of the 3 industry-level (third-party) engines mentioned, to learn how to work on. CryEngine has a difficult interface to work and semi-difficult controls, doesn't have an asset store like Unreal and Unity and its community is quite small which means there is little to no support. It was developed using Lua and C++ while available scripting languages are C++ and C#.

CryEngine also offers some tutorials for beginners and does not really delve that much into complex and more advanced applications the way Unity does, but it does award 3 different certifications for completing the respective tutorials.

However, even after those drawbacks, CryEngine most of the time produces the most astounding graphics and performance optimizations. This can be backed up by the fact that the engine won the 2014 SIGGRAPH award for Best Real – Time graphics with the game Ryse : Son of Rome that was developed with CryEngine.

4.2.2.2.1 Interface of CryEngine

Figure 22: The most basic Interface of CryEngine depicts the most basic interface of CryEngine. When the program is installed and opened for the first time, this is what the user sees. The interface at a first glance is not exactly the most user friendly,

4.2.2.2.1 Interface of CryEngine

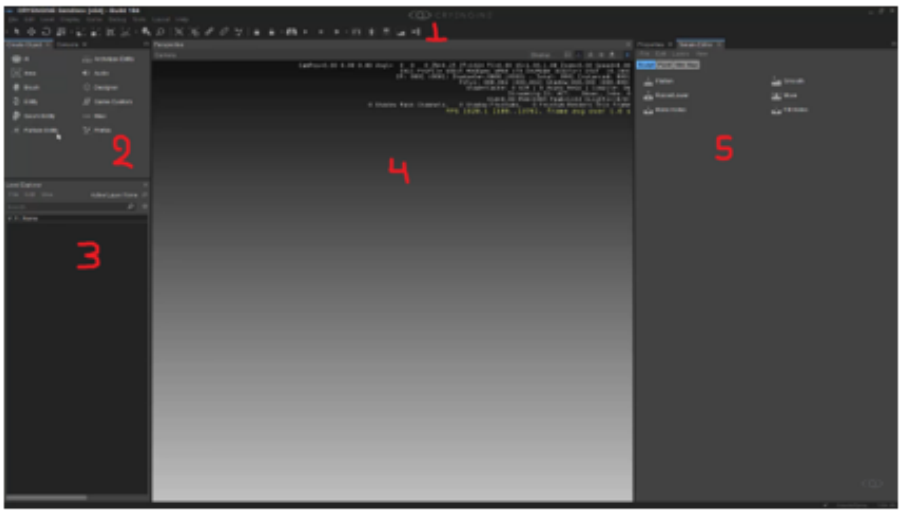


Figure 22: The most basic Interface of CryEngine 5.6 [F.22]

especially when someone new sees all the numbers on the “Camera (4)” tab.

The numbers depict each different part of the interface. They can be removed at will, at any time the user wants, by clicking the X and can bring them back by browsing the tool tab above the toolbar (1). So what is each number and what purpose do they serve?

1-The toolbar is a menu that provides quick and easy access to many tools and features that are the most commonly used tools in the sandbox editor. It cannot be moved from its position, but it can be completely customized from icon size, to what it contains or doesn’t. By right clicking on an empty part of the toolbar the user get some options and can choose what the toolbar shows by enabling (visible tick) or disabling (non-visible tick) each menu. The different menus are Audio, Constraints, Coordinates, Edit Modes, Game, Layout, Physics, Selection, Standard, Viewmodes and Customize. Every menu has some settings or tools that help with certain aspects, one example out of thousands is gravity in physics menu. You can quickly set if gravity should be enabled and how strong it should be.

These are the most common and default menus that CryEngine has. Of course, it can be customized in any way the user desires and can offer much more and a lot more complex menus, uses and settings to suit needs better.

2-The “Create Object” menu contains all the objects the developer can use to create all sorts of things like characters, backgrounds, boundaries, pop-ups, decisions/actions etc that contribute to gameplay. An example I chose is an object. It can be as simple as a picture that depicts a rock or a bench or an obstacle or it can be as complex as a picture of someone that can later be tuned with physics, motion and animation that is used as a character.

3-The “Layers” menu is almost the equivalent of hierarchy menu of Unity. It contains the game objects created but it has some differences. Layers affect how everything shows, for example two backgrounds that cover all of the screen, with background-1 above background-2 will show background-1. So, game objects with the same hierarchy in CryEngine and Unity might show differently. Unity has this “Layering” separated as a unique feature for each game object.

4-The “Viewport” or “Perspective” is what the display is in Unity. It displays how the game will look in real time but since CryEngine does not have an extra “scene creator” like Unity, it has some extra features that allow the developer to do so, as a 2-in-1 thing.

5-The “Properties” menu contains two tabs, the “properties” and the “editor”. Both are what the inspector window is in Unity. The properties tab contains properties that all elements and scripts that an object might have and in the editor they can be edited. In Unity these are both done in the inspector as the elements are separated in small sections that contain the editable variables but in CryEngine you access the editable properties on the editor by selecting the element you want to edit in the “properties” tab.

These are the most common and default menus that CryEngine has. Of course, it can be customized in any way the user desires and can offer much more and a lot more complex menus, uses and settings to suit needs better.

4.2.2.3 Unreal Engine

Unreal engine is among the first and one of the earliest 3D game engines. It was developed by Epic Games and was released in 1998 when they released a first-person shooter game named Unreal that was used to showcase the game engine. Unreal engine was initially developed for first-person shooter games solely but over time they developed it to the point where developers can use it to create any game genre.

Unreal Engine is free to use with limited support and Epic Games states that games that are developed using the free version of the software are royalty free as long as the lifetime revenue of the game stays below the $1.000.000 mark and that once a game passes that mark, under the standard EULA, a 5% royalty goes to Epic Games.

The engine was created using C++ and uses the same language for scripting. It doesn’t support exporting to as many platforms and operating systems as Unity does however, Unreal Engine has two unique benefits that Unity and CryEngine do not have.

The first one is that Unreal Engine is a cross-platform engine meaning that games created can be played simultaneously by multiple platforms at the same time. Example, a PC-Windows player can play online with (or against) someone playing from Console-PlayStation. The second benefit is that even though it doesn't support as many platforms to export game to, it can render "browser" games for HTML5 using OpenGL, something which neither Unity nor CryEngine can do.

4.2.2.3.1 Unreal Engine Interface

Unreal Engine does not have a set interface like CryEngine and Unity. In Unreal Engine version 5, the developer gets different options on templates for each individual thing. A template for a first-person shooter game might be different from a Third-Person or a Top-Down, among other templates game AND non-gaming related.

Even though Unreal Engine has slightly more difficult and confusing interface, it offers just as much documentation and tutorials. It also has quite a big community, like Unity does for forum support, but does not have as much video tutorials from freelance creators and educators on platforms like YouTube.

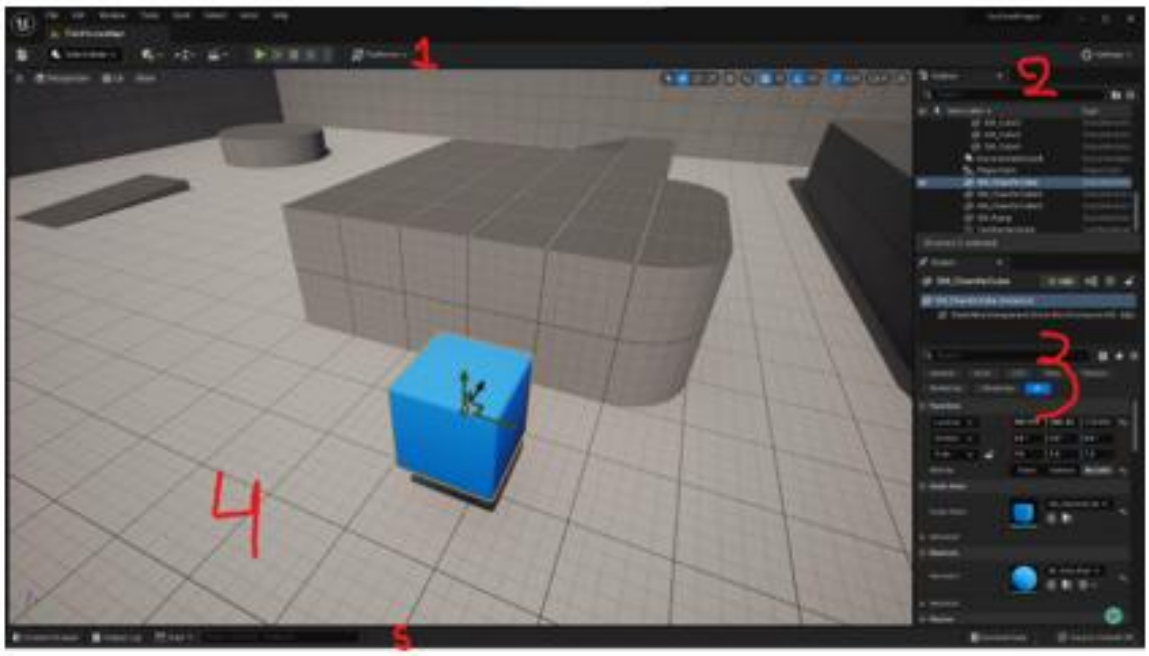


Figure 23: Interface of a 3-D Third Person template [F.23]

The Interface of a 3D third person template contains 5 sections, 1.the toolbar, 2.the outliner, 3.the details, 4.the prespective and 5.the console.

1-The toolbar, like CryEngine and Unity contain basic tools to play the scene, control the camera and change the display platforms. Since the templates change in each different type, the controls are placed differently. This means the controls are not on the toolbar but rather they are on the display itself as they change related to the template.

2-The outliner is the menu that creates and holds all the game objects that exist in a scene. It's the eqivalent of the hierarchy menu of Unity. They do not have many differences apart from minor details like the placement of the menu, names etc.

3-The details menu holds all details, scripts, components and parameters for game objects selected from the outliner menu. It's the equivalent of the Inspector menu from Unity and like the outliner menu, they don't have many differences other than the names and placement.

4-The prespective view is similar to the display and scene view of Unity and the prespective view of CryEngine. The developer can see the scene and create simultaneously like in CryEngine with the minor difference of the game object controls as they are placed on the prespective view and change when the templates are changed.

5-The console menu is pretty simple but slightly different than Unity and CryEngine. It has 3 tabs, the content drawer, the output log and the console command. The content drawer is the file where the scripts are kept/saved, the output log shows every script that runs and whatever outputs they may have along with the warnings and errors that come from faulty codes and lastly the console command. This console command is the difference between Unity and CryEngine because it can be used "on-the-fly" to create "script" commands to check certain parts of code or how a few lines of commands would react.

4.2.2.4 Other Game Engines

Unity as a game engine, like Unreal and Cry, is considered a third-party engine and all three of them are very well-known engines, with many games under their name. The way

it has been developed allows for it to be used by any creator to make all different kinds of games, however, not every engine is a third-party engine. All these three provide more or less the same capabilities to the creator with no major differences between their interfaces.

The other type of game engines that exist are non-third-party engines. That means that they are developed by the game studio itself and usually specialize in a specific game genre, or in some rare cases, one specific game only. Non-third-party engines that are quite known are Creation Engine, Dunia Engine, Source, Source 2 and many others.

4.2.2.4.1 Dunia Engine

Dunia Engine was developed by Ubisoft in 2004 and was released in 2008 when they showcased their first game based on the engine, Far Cry 2. Ubisoft developed the Engine by copying the source code of CryEngine and engineered their own distinct version, which they named Dunia. The engine featured in every game that Ubisoft released, like the rest of the Far Cry titles, the assassin's creed titles, watch dogs and many more.

Figure 24: The logo of DUNIA Engine as it is portrayed in games, with minor different details around the edges. [F.24]

Dunia engine was very advanced during its release, featuring a lot of things that most game engines did not have at the time including movable objects such as barrels or crates for example, ladders that acted like normal ground that was vertical – meaning that extra controls were not necessary, better physics, enhanced lights that provided less illumination the further away the player went, advanced shadows that reacted to lighting and a lot more.

4.2.2.4.2 Source and Source 2

Figure 26: Logo of source Engine, as portrayed in source engine powered Steam.[F.26]

Both Source and Source 2 are engines made by Valve. Source was showcased in June 2004 with the release of Half-Life : Source, as a successor to GoldSrc. It was also used for Half-Life 2 and Counter-Strike: Source a year later. Source 2 is the successor to Source Engine and was released in 2015. It was released as a game engine suitable for MOBA games (Multiplayer Online Battle Arena) and was showcased when the game Dota 2 was released the same year.

Figure 25: Logo of Source 2 [F.25]

Both engines were developed using C++ as programming language and were both released as “improvements” to fill the different needs that each game had.

4.2.2.4.3 Creation Engine

Creation is an engine made by Bethesda Game Studios. It was released in November 2011 along with the release of the game it powered, Skyrim from the title “The Elder Scrolls”. Creation engine was developed using C++ and it was optimized in order to be used to create big Open World RPG games like Fallout and The Elder Scrolls.

Creation engine has some unique features that other engines did not have at the time. Bethesda implemented middleware that helped with AI behavior such as animation AI to blend movement better and improved versions of AI software “Radiant AI” that helps dictate how non-player characters move and interact with the environment around them such as

working around the house or eating different foods depending on the hour or even going to a tavern/inn for drinks.

4.2.2.5 Differences among Game Engines

The biggest difference among them is the fact that Unity, CryEngine and Unreal Engine are in fact available to the public for free (provided that games created for free follow the terms and agreement mentioned above) while Creation, Dunia and Source are made by their respective game studio for private use, meaning that they are not available to the public in any way.

Another difference that they have is what games they are optimized to create/render and on which platforms. For example, the only engine among the ones mentioned above that can create a game and export it for HTML5 (browser platform/browser game) is Unreal Engine or if a developer requires to render for "pocket consoles" like PlayStation Vita, Unity is the only able engine to do that.

There is also the option of which coding language one prefers to use for scripting. This influences a developer's option on which engine to use depending on whether he wants to use C#, C++, Java, JavaScript or other custom languages for scripting.

The last big difference is whether the developer is a beginner or a more advanced developer because each individual game engine has different interfaces of various difficulties regarding their navigation and how much tutorials, certifications or support they provide for users, with Unity providing the easiest interface controls along with the most thorough tutorials and best community support with tens (if not hundreds) of subjects being posted and answered each day on their forums.

4.2.3 Coding Languages

Coding languages, or Programming languages, are like normal languages that

Figure 27: Coding Language Python Logo [F.27]

have their own vocabulary and grammatical rules. They are used in computers and have specific sets of rules, keywords and syntax that, if used correctly, can instruct a computer or a device to perform certain specific tasks. When someone uses the term "Coding/Programming language" they usually refer to C, C++, C#, Java, COBOL, Fortran, Basic, Python etc.

Figure 28: An unofficial logo of C#. There is no official logo for C# as stated by Microsoft support [F.28]

4.2.3.1 C#

Unity has two ways of reading scripts. One way is using its own language for scripting, the UnityScript and the other is Using C#. As a beginner I elected to use C# like the tutorials recommended.

C# (pronounced as see-sharp) first appeared in 2000 along with .NET framework and Visual Studio. It's a simple, modern, general-purpose and object-oriented language that uses classes which is ideal for coding a game.

4.2.3.2 Similar Languages to C#

A similar coding language to C# is Java. It is similar in many aspects as its simple and general-purpose, but the biggest similarity is that Java also uses classes. This is very important, because someone who has studied java before and wants to start on C# (like myself) will find it much easier to transition rather than using a completely different language like C++ or other custom languages like UnityScript.

Figure 29 : Current official logo of coding language Java [F.29]

4.2.3.3 Coding Environments and Editors

Figure 31 : Official Logo for Visual Studio [F.30]

An editor is a program where a developer can write the code in, and that code is read by the computer using the coding environment or runtime environment. Editors are generally anything you can write in such as word documents and notepads, however its trickier to use these for writing code as they do not have the same marks and structure of a full-fledged code editor that uses colours and other marks that help out with visual order.

Figure 30: Notepad++, a simple editor for writing code [F.31]

Full-fledged editors are made with writing code in mind. This means they provide structure, markings, colours and other visual aids that help the developer write the code. One lightweight, good and simple program, that I also used a lot, is notepad++.

Figure 32 : Logo of the Microsoft IDE, Visual Studio. The shape represents the infinity symbol as a testament for all the countless possibilities the IDE offers. [F.32]

Coding environments, or “Runtime Environments” provide comprehension of said written code on editors, to computers, as a computer cannot understand the coding language without the runtime environment. It basically acts like a “translator” to the computer. Runtime Environments usually specialize on specific languages by using “libraries” that contain the information of the syntax and the set of rules of the language, but their true purpose is executing the written code. These Runtime Environments have a few drawbacks though, regarding faulty code, as most of the time they don’t show which and where an error is found, so it’s up to the developer to figure out what is wrong.

Figure 33: Netbeans logo, a complete Java IDE [F.33]

There are programs that integrate both the runtime environment as well as the editor. These programs are called “Integrated Development Environments” or “IDE’s” and they are specialized software that have the ability to have both the runtime environment as well as an integrated editor meaning that you can edit and change sections of the code while also checking and debugging simultaneously. Since the editor is connected to the runtime environment directly, this saves a lot of time - especially in debugging - as the runtime environment can communicate with the editor directly and show exactly where the error is. This means that the error can be changed immediately and retested without wasting time searching for the error manually. An IDE also saves files with the appropriate file extension, so the user does not have to go back saving or changing the file extension every time.

IDE’s, like runtime environments, also specialize in specific languages which means that there are a lot of programs that integrate both the IDE and the Editor in one package. One of them, which I also used, is Microsoft’s Visual Studio. Microsoft’s Visual Studio supports C, C++ and C#. There are other IDE’s out on the market that support more coding languages, such as Oracle for Java, PyCharm for Python code, WebStorm for JavaScript, or NetBeans which supports Java, PHP, C and C++, among many other IDE’s.

Figure 34: Eclipse IDE logo, an IDE for Java and C [F.34]

4.2.4 Art and Design

Art and Design in games is the second most important thing, after creating the scripts needed for everything to run properly. Art can be found everywhere in a game, from the ground, to background, to characters, surroundings, buildings and generally every object. It also dictates how a game looks and feels aesthetically and it's important to use the correct design for each game/genre.

A good example of how important the selection of design is – using "futuristic" design for houses on a medieval RPG would be a terrible idea and not make all that much sense, but using design that looks old and rough, with houses looking like they are made out of wood, stone, marble of a combination of the 3, would make much more sense and produce much higher quality aesthetics and a neater fit for the theme of the game.

There are two ways to get art for the game. One way is to create it yourself and the other way is to get someone else to create it for you, with both ways providing their own unique drawbacks and benefits.

Creating your own art and designs has 2 major benefits. One benefit is that usually you can allot as much time as you want or need on each design and the other benefit is that creating your own designs means that it can be closer to how you want it to look by re-designing it as many times as it takes. This method though, has two major drawbacks. In order to create your own designs, one needs to be a good artist which is a very difficult skill to even learn - let alone master. So, for someone who is not a good enough artist, they might be better off relying on someone else to create designs for them. The second drawback is that no matter how much, you still HAVE to a lot time to create a new design which sometimes might be a waste.

The other way, having someone else making your designs, could be achieved through two ways. One is either having someone working with you privately, making designs solely for your needs, or get someone's already premade designs and use them.

Ideally, it's better to create your own designs, but when this is not possible for any reason, its better off to have someone working with you privately, rather than resorting to

premade assets. The reason for this is that using premade assets will mean that its highly likely that someone else has used the same assets, which is a drawback. This could result in mainstream design while making your own design, or someone making your own design privately, can bring unique assets to the game since it's quite unlikely that someone had the same idea.

This does not mean that getting premade assets is entirely a bad thing. Getting premade assets has its benefits too. Getting already made designs means that the developer wastes almost no time creating how something looks and can focus on other things. This acts the same way when you have someone else making designs for you privately. Even though you have someone making designs for you, you have to look at it and decide whether it's good or not and communication still costs some time, even though it still costs less time than making it your own design.

4.2.4.1 Asset Store

Whether creating your own design or getting someone to create a design for you, it can be saved locally. However, finding pre-made assets is a bit trickier. These are usually not saved locally, therefore finding them requires some research. Unity and Unreal have already thought a few steps ahead and have created their own asset store. An asset store is where creators, artists and designers upload their work. This unlocks a lot of potential for both of the developers and designers, as developers can have a place, where everything is orderly and organized, to find assets that suit each game best, without wasting time searching and designers can upload their work there and potentially sell it or advertise it.

4.2.4.2 Other Programs used for Design

Figure 35: Digital Drawing program, Krita logo[F.35]

Art and Designs can be made through different programs on a computer, phone, tablet or drawing pad. For simple picture editing and frame-by-frame animation

just using photoshop is enough, however there are many programs out on the market for digital drawings that can do a better job for someone that wants good quality design.

Some of these advanced programs include Krita, MediBang, Ibis Paint X, Clip Studio Paint and many more. Out of the four programs mentioned, Krita, MediBang and Ibis X are free to use, while Clip Studio Paint requires a one-time payment to buy and is also the only one out of these that supports 3d designs and movement.

Figure 36: Logo of Ibis Paint X, taken from google play store [F.36]

4.2.5 Game Categories

This is not exactly a tool, but it is a very important aspect that should never be overlooked when making a game. Game categories usually provide information on what the game will be like, generally. For example, a game that is categorized as “Hack and Slash” makes sure that the player knows the game is probably not going to have guns as weapons but rather its going to have swords, axes, shields and generally medieval melee weapons. Having guns in this scenario would leave a weird impression and not make much sense, therefore it is important for a developer to think about what category he would like his game to be, before starting to create it.

Game categories are divided into **Main Categories** and **Subcategories**.

4.2.5.1 Main Categories

Main categories contain the nature of the game. They include categories like RPG (role-playing games), Racing games, Strategy games, MOBA games (multiplayer online battle arena), Sports games and some others.

Usually, games do not have more than one main category as for example a Racing game cannot be a MOBA, or and RPG or Sport game etc.

4.2.5.2 Sub-Categories

Subcategories contain secondary features of a game. These include categories like action games, adventure games, fighting games, roguelike games, visual novels, simulation, open world games, sandbox games, fantasy games and hundreds of hundreds other categories.

Games can have many subcategories and most of the time a game can have any subcategory as long as it makes sense with its main category. For example, a Racing game usually does not have adventure or action as subcategories, but it may be categorized as VR and simulation.

CHAPTER 5

MAKING OF THE GAME
THOUGHT PROCESS, LOGIC USED AND WHICH STEPS WERE TAKEN WHEN, DURING THE MAKING

In this chapter I will thoroughly explain all the steps I took while making the game and how my thought process looked when making decisions from the most simple things to the most complex ones.

5.1 General Information

The steps I followed here are not set in stone. Making something from scratch and for the first time proved quite tricky and it was very common that I had to keep revisiting earlier steps to fix mistakes, add things that proved necessary or changing between two or more things for the better.

5.2 Step One – The Decision to Make a Game

The first steps I took to make a game was choosing what type of game it would be, what the story would be like and what kind of categories I would link it to. The most important in all that is the story, as it dictates the other two.

5.3 Step Two – Where to Make the Game

The second step I took was choosing the platform which I would make the game for. Choosing the platform is important in many ways. First, it's the performance optimizations that would need to be considered and implemented. A smartphone for example does not have the same processing power as a desktop computer, therefore adding very high-quality graphics might reduce framerates and stability since a smartphone would not be able to handle it.

Another reason are the controls that change depending on the platform and OS. A smartphone is usually touch screen which is much more different than consoles that use a

specialized controller or a desktop computer that uses a keyboard and mouse. It's worth noting that there are existing compatible peripherals made for smartphones, like controllers and other aiding devices that help out gameplay.

Another reason is the market. A market influences how many people will play the game and how much the game will be known, also referred to as how much "clout" it will gain. When trying to release a game to a certain platform that is not as known as other platforms, players might not get to hear about the game. Considering the previous research of worldwide OS market, one can conclude that making a game optimized for Linux and not Windows would not be a great decision, same thing with smartphones and making a game optimized for iOS only and not Android. Decisions like that would create a very small game community and would most likely not be a very effective way to bring in profits, considering the business point of view.

5.4 Step Three – Choosing how to Make the Game

This step was about finding ways to actually make the game, meaning which tools would be used to breathe life to it. Tools include everything from IDE or Editors for scripts, programs to edit and process video, audio and frames/pictures, making the research to find which game engine to actually use, where to find the designs and necessary art for characters, background etc.

5.5 Step Four – Deciding how the Game Flows

This step is a follow up of the previous step. When the programs have been gathered and installed it was time to start making the game. Before I started adding the characters and background to set the scenes, I had to envision how the game would flow, which stages I wanted to add and which scenes would come first. This coincides with making the story for the game which prompted me to think about it a bit more. In the end I decided to make 4 stages of the game. The first stage I decided would be a "tutorial stage" that would focus on showing the movement and action controls to the player. This tutorial stage would transition to the second stage where the focus would be to move around and present the story of the game. The focus in the second stage would be all the dialogue and explaining the story to the

player. The third stage would be a “battle-stage” where the character would fight through enemies to reach the main villain, which he would eventually fight as well. The fourth and last stage, or most commonly the “epilogue” of the story, would be the aftermath and the credits of the game.

This ordeal is mostly known as “directing” the story, usually done by directors to ensure the story flows smoothly without the so-called plot-holes and repeated story or scenes, both of which are never a good thing to have in stories.

This step was mostly done in theory, with an approximate direction of how I want the story to go. This was one of the steps I kept revisiting before it was actually finalized.

5.6 Step Five – Getting the assets needed and preparing the designs necessary.

In this step I focused on finding and getting all the assets that I think would be necessary for the creation of backgrounds and characters. This is the step I kept revisiting the most, as I kept having to find more and more assets for each stage to create enemies, paths, icons and other backgrounds. This included scouring through the Unity Asset Store for designs and importing it on Unity for use.

5.7 Step Six – Stage one, Tutorial

As I said before, the first stage would be a tutorial that focuses on showing the controls and movement of the game. At this point I had an idea of how my background and stage would look like. It was time to create the background and place all items, characters and path blockers needed. This in turn created the need for player movement, the animation for the characters moving and making game objects that worked as limits to obstruct the player from going “off-course”

The limits were the easy part as the object was marked with a collider to simulate a collision that did not allow the player to go through. This was a good way to ensure that the player stays within the limits of the stage so I reused this method to create limits up until the end.

The same collider was also used as a trigger for popping up dialogues automatically for the tutorial to inform the player for certain actions and explanations on controls.

The last and most important part was making the movement for the character and synchronizing the frames for a smooth walking animation.

5.7.1 Step Seven – Creating the Movement Controls and Animation

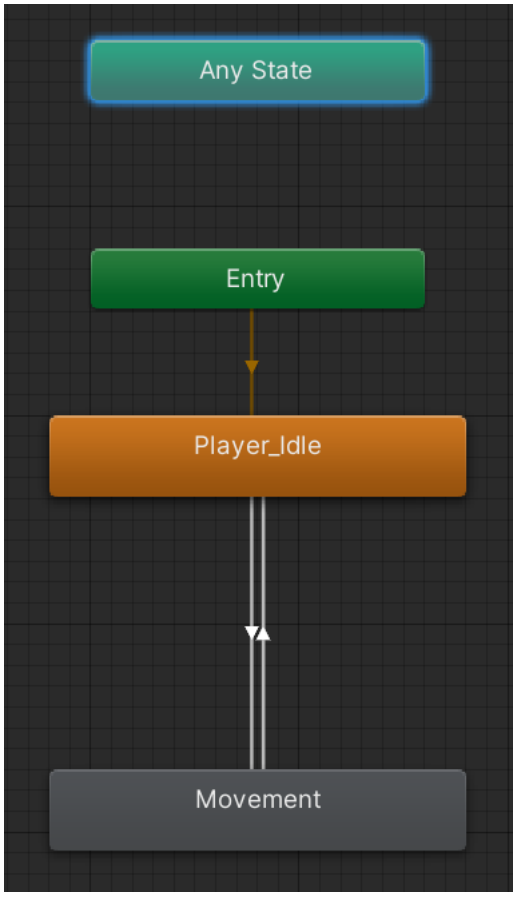


Figure 37: The animation phase connections between idling and moving that dictate which animation will be triggered.

The first thing I did was to create the movement and sync the animation after. I believed it would be easier to sync the animation to walking after setting the speed. The making of movement consisted of two different parts, creating the script that would allow the movement as well as setting the speed of the movement and creating the controls that would work parallel to the script depending on which is being pressed each time.

The way I made the script was to set a movement command that would move the character with a set speed along an axis and it would be triggered when a button was pressed and would be triggered off when that button was no longer pressed. This command was made four times, one time for each axis, the X positive, the X negative, the Y positive and Y negative. At this point I had the command that would move the player along all 4 axis but no buttons were linked.

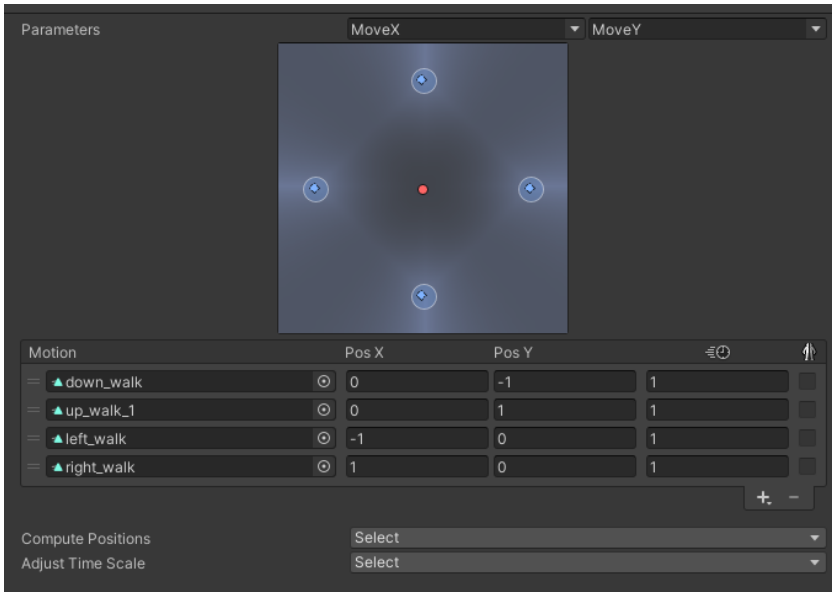


Figure 38: The calibrated parameters needed to ensure that the correct animation will be played, that are set during the pressing (or not pressing) the movement control buttons. These are set between -1 and 1, with 0 being not pressed..

This meant the need for controls was created. In smartphones there are two most common ways to create controls. One is the "swipe and hold" which is also the most common that feels like a joystick and the other is the one I used, the button that acts as tap and hold. I elected to use the tap and hold controls as I felt like it would be a better fit for the "old-school retro game" I wanted to make. With this in mind I created the interface for the controls, placed the icons that act as buttons when tapped and I also connected each button to a command that moves the player along an axis as well as setting several variables when pressed or not, that were needed in the animation process.

The only thing missing now was the animation itself. Unity provides easy animation interface that made animation creating a piece of cake. All I had to do was link the movement and controls to a "stopwatch" of sorts and I just picked the time where the animation would change frame to ensure smooth walking animation. I also had to create the idle animation where the player faced the correct way when stopped. For example if the player moved west and stopped, his animation turned back to facing north, or up. I

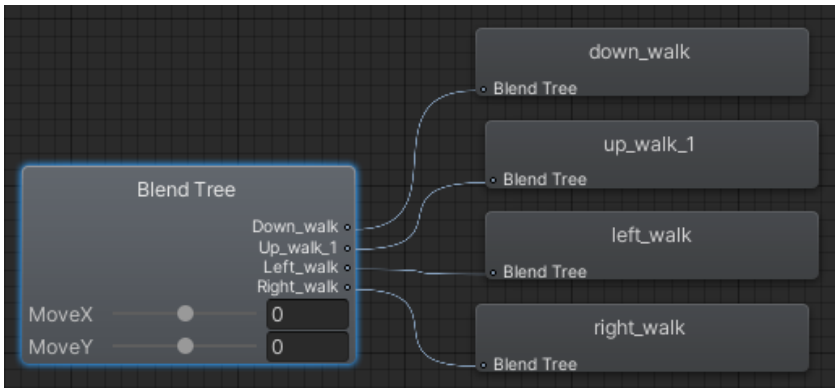


Figure 39: The connections that dictate which animations are being played depending on the different parameters and variables set from Figure 38 and movement controls.

had to create another 4 animations that played when the player was just standing and it used an X variable as a “last facing direction” to ensure proper direction when the player stopped.

The whole process of creating the parameters, setting the variables and ensuring the correct variables with the press of the buttons, proved to be quite complex and challenging but once the whole process was completed, it could be used for the rest of the game as well and could be used on NPC’s as well with minor modifications to the code to act as an AI.

I used this movement script and animation up until the end. The script also used certain variables that calculated the speed of the player, so if I needed I could have speed changes easily without changing any parts of the script and the movement was calculated with a Unity time command.

5.7.2 How the animation works

Basically, when the MoveX and MoveY variables were set to 0, no movement buttons are being pressed therefore the player is not moving so the animation focuses on the Player_Idle section which containes the idling animations. The idle section also contains two extra variables, the LastMoveX and LastMoveY which are being utilized to determine the last known facing direction of the players movement and are used to ensure that the character continues to look at the correct direction when movement controls are not being pressed.

When a movement control button is pressed again, all four variables are set to their correct numbers. For the sake of the example, if the button to walk up is pressed, The X variables would both remain 0 but the Y variables would both be set as 1 since it was calibrated that way. Now since one of the two variables, MoveX and MoveY, are no longer 0 (as the MoveY is now 1), the animation now focuses on the Movement Section.

The movement section now checks the variables and decides which animation will be played. With MoveX equaling to 0 and MoveY equaling to 1, it changes its focus to Up_Walk_1 animation, which starts the “stopwatch” with the correct frames and the animation is now being played.

When the movement button is released again, the X variables will stay the same since they were 0, but the Y variables are a little different. The MoveY will turn back to zero and the focus will change back to the Player_Idle section but the LastMoveY will remain at 1, because it needs to check the last known facing direction so that the correct idle animation will be played.

Pressing the buttons triggers both the movement script formula and the animation variables which means both work together and parallel to each other for a smooth movement. If animation does not exist, the character looks like an object that is being dragged around and if the movement formula does not exist, then the animation starts playing but the character stays still.

5.8 Step Eight – Creating the transition from stages

After creating everything needed for the movement and scripts I had to create the first transition from stage 1, tutorial to the stage 2, story. This was done with a collider that took away the movement of the player and slowly blacked out the screen and placed the player in another room.

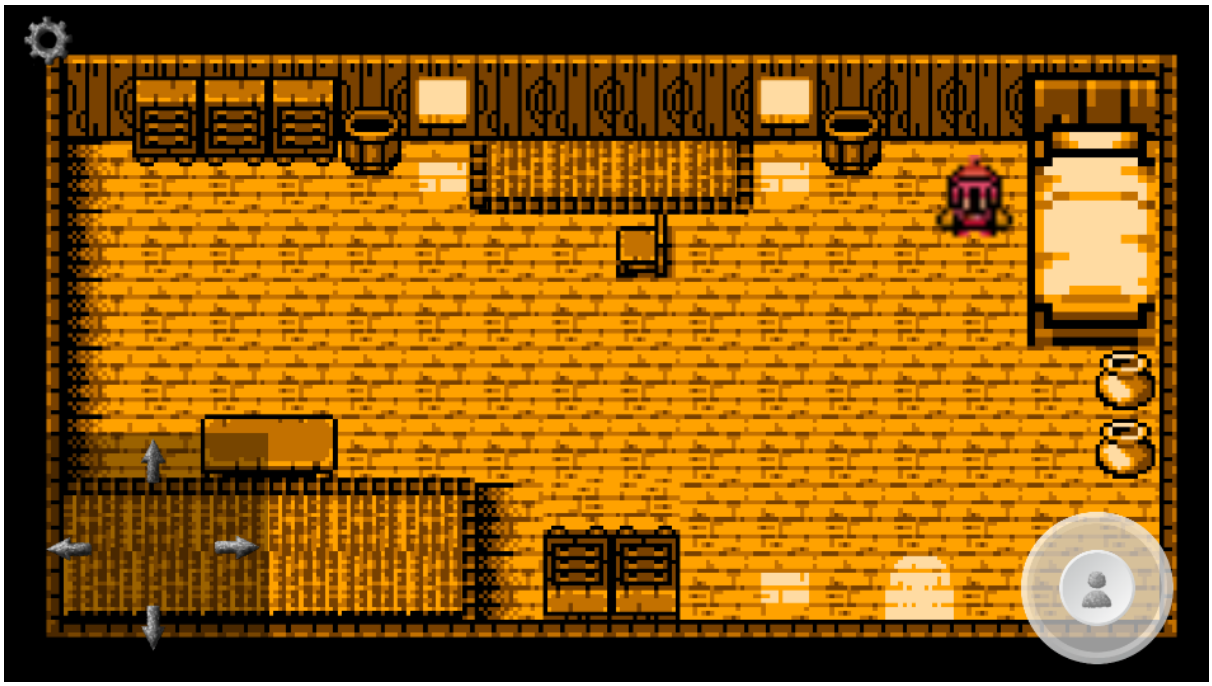


Figure 40: The room where the main character wakes up after the tutorial scene. Top right is the main character beside his bed, bottom right is the action button from controls interface and bottom left is the movement controls from movement interface.

5.9 Step Nine – Stage Two, Story and Dialogues

This stage was focused on dialogue and progressing/explaining the story to the player. At this point no dialogue has been added even to the tutorial stage yet. I had to create the interface and the dialogue scripts, as well as the flow of the dialogue and the colliders that triggered each dialogue.

The first thing I did was to create the interface for the dialogue. This meant going back to find assets for the interface and the font that I would use for the letters. The script was synchronized with the interface for the dialogue and would trigger the interface whenever a collider was triggered. The interface would show whatever dialogue was up depending on where in the conversation the player was.

The interface took away the movement controls from the player and would give it back when the dialogue ended. This was done to ensure that the player would not be able to move during dialogue to avoid bugs.

After finishing creating the colliders for triggering the dialogues, the interface for the dialogue and the dialogue itself, I went back to stage one, tutorial to add dialogue and colliders the same way.

The dialogue in stage one contained explanations for the controls and other dialogues that acted as thoughts of the player when he tries to go outside of bounds. The dialogue in stage two contains information about the story of the game.

5.10 Step Ten – Stage Three, Battle Stage

This stage contains a battle stage. A battle stage means that the player has to fight certain enemies using certain ways to reach a goal. As I said previously, the way the battling works here is a unique way inspired by another idler game I played in the past. I created the first battle as a tutorial, using the same dialogue technique from earlier stage to explain how everything works.

When the player triggers a collider for battle, he is prompted to another interface which consists of the health bars of both the enemy and player, the available skills of the player and a bar for each skill that simulates the cooldown. When the player uses a skill, it does whatever it is supposed to, for example heal raises the players health and the bar empties. Once the bar empties, it starts filling back up simulating the cooldown and when its full it changes colour indicating that the skill is ready to be used again.

To create such battle stage many scripts were needed. One script contains the stats of the player, meaning the skills, health etc. Another script contains the stats of the enemy also meaning skills, health etc. These are connected on the player and the enemy when the battle interface starts. The player and enemy also have a variable to dictate their level and the stats scripts use this level to dictate how high the damage, health etc would be set. This is done using a linear formula that has a different output depending on where the level is set for easy and automatic stat check of both player and enemy. It is the simplest form of getting stats in a battle.

Another script was needed for the skills of the player and is also connected to the player when the battle interface starts. It contains all the skills the player is allowed to use along

with what they do and the variable for cooldown of each skill. A similar script is also connected to the enemy so they can also attack back.

Lastly another script is needed that contains all the skills. These skills have the formulas needed to calculate the outputs needed, for example the damage skill takes the level and calculates the damage it will inflict on the enemy when pressed. It also contains the formula needed to fill up the bar for the cooldown of the skill.

5.10.1 Battle Stage Interface

Figure 41: The battle stage interface

This is the interface of the battle stage I created to use. The different numbers indicate all the things that are happening during this stage. Number 1 is the text that indicates a battle has started and by pressing the next button all the scripts start running to give the player some time before actually starting. Number 2 indicates the two platforms that the player and enemy are placed upon. 2.1 is the platform the player stands on and 2.2 is the platform the enemy stands on. 3 is the interface with the skills available to the player as well as their description and cooldown bars. Number 4 are the stats for the player and enemy, with the name linked, the Health portrayed as a bar and the level of the characters. 4.1 contains the stats for the player and 4.2 contains the stats for the enemy. 5 is the text that explains what is

happening during the battle and Number 6 is the background I created using the assets I found from the asset store.

5.11 Step Eleven – Stage 4, Boss fight

This step contains the last fight. After passing all the enemies in the battle stage, the player enters a room that triggers the next stage. This stage contains just the room and the main villain. Going close triggers the dialogue between them and after the dialogue they engage in a fight like the previous ones. This fight has a minor difference as the player can now use a special attack that has a very long cooldown but it's the only skill to defeat the main villain.

5.12 Step Twelve – Stage Five, Epilogue and Credits

This stage triggers after defeating the main villain. It shows the end of the story, also known as the epilogue. Its just some dialogue to close the story. Most scripts made for stage one two and three have been used repeatedly throughout. Dialogue and movement is the same in every stage and battle stage is the same for both enemies and main villain.

5.13 Step Thirteen – Menu

After finishing creating all the stages, I created a menu with starting the game along with some settings for audio and graphics. Unity has premade settings so the player can use these without wasting too much time. All I had to do was create the interface and the transitions between options and different menus and the sliders or boxes for different graphics and audio triggers.

Figure 42: The menu of the game with buttons for to play, load a certain scene, settings menu and quitting options

5.13.1 Step Fourteen – Adding Audio

After I created the menu with the audio settings, I decided to add background music for each stage. Music helps with stimulating feelings of fear or joy when used properly. This required two scripts. One script handles playing the music and finding it and another that handles the settings and using the correct file depending on the stage of the game.

All music that has been used is copyright free and has been edited whenever needed using the program Video Pad Editor.

5.14 Step Fourteen – Saving System

The next step after I added the audio settings and scripts was to create a saving system. There are two different ways to save data. One is by creating a file that contains all the data and is used by the game to read certain points to load the correct things and the other way is to use something called “flags”.

Creating a file to contain all data is always better since it neatly saves all progress but its much more complex to create.

Creating flags is much easier to manage when there are only a few things to manage however it can easily get jumbled up quickly and then you start having major problems when more and more data are saved.

Since I don’t use a lot of data I decided to use flags to save time. This means that every stage the player passes, it raises a flag. If the player then quits before finishing the game, the flag is raised and when he presses play it transitions to the correct stage instead of back at the start again. When all stages are passed, all flags are lowered so all stages can be replayed but now another flag is raised as the end. This flag enables another menu that the player can use to replay any stage he wants no matter the order.

5.15 Step Fifteen – Transition Between Stages and Play button in menu

This has to do with scripts used in the menu to transition between all of the different menus. It is basically just a script that enables one variable and disables another to the correct menu is shown when a button is pressed.

Same thing with the play button. It has a script attached to it that reads which flags are raised so that the correct stage is loaded. It is the same with the quit button, a script is attached that closes the game when quit button is pressed.

CHAPTER 6

FUTURE OF GAMES – ADDITIONS FOR THIS GAME

6.1 Future of Games

The gaming industry is one that has been gaining more and more attention the last years. With online gaming becoming more and more available for gamers, with e-sport tournaments becoming very popular in many competitive games and plenty of games with amazing story line and even better gameplay, it is plain to see that the gaming industry has been thriving. With smartphones providing all the necessary things a mobile phone needs as well as a mobile source to play games, that can easily surpass old, dedicated pocket consoles of the past and being a device that almost everyone has its clear as day that the next gen of gaming will be mobile gaming for quite a long time.

6.2 Potential of this game

The game currently is lacking in gameplay. Everything is mostly made for demonstration. However, there are many aspects that can be improved upon to give the game, the gameplay it needs. The story can be improved upon to have smoother flow and better narrating, scenes can be added before stage two to escalate the events and after stage two for gameplay before reaching the climax of the story. More characters can be added in the story to play a role and immerse the player better. More stages can be added where the player can fight enemies. More mechanics can be added in battle. The battle system can be further improved upon with more skills, spells, damage prevention moves, combos etc. More characters that act as a main character with different skill sets can be added for additional variety. And last but not least, the saving system will need to be changed from flags to file data.

6.3 Possible Additions

I will list some possible additions and a baseline of what will be needed for these additions.

1-Adding more story. This is a directing matter mostly. A screen play script is mostly needed to create certain lines and dialogues that the characters will have. The rest will be just creating backgrounds with assets and adding the coding scripts as well as the colliders needed for the conversations and dialogues.

2-Adding more characters that bring in different tasks, or information that can be used. This can be done by just adding game objects with similar parameters and scripts with the main characters with minor differences to the collider so it can trigger a dialogue and with minor adjustments to the movement script so they can have movement on their own.

3-For starters, adding different simple skills to the character beside damage and heal that he can use. In a more complex and more advanced manner, dodging or parrying mechanics can be added with timing. For future additions, more skills can be added that can be unlocked on certain levels or even different skill sets and skill trees can be added. For a baseline, a dodge can be added the same way other normal skills are added which can be pressed and stay active for a few seconds and it will negate the next spell that hits within a time limit.

4-The game save and data can be improved by removing the flags completely. Unity provides a way to save stuff within a file format like XML or JSON. That way it is very simple to store data and progress and it is very easy to modify as well. However, their strength is also their weakness as the ability to modify them makes them not at all secure.

Another way to save stuff is creating a custom binary file. In theory and using a quick explanation, a binary file saves data the same way but its much more secure due to data being in binary. The way to create a custom binary file is to create a class that contains all the variables in either string, bool, float or int, make that class serializable and use a binary formatter to turn these data into binary. The saved file can be imported back to the game and with the use of the binary formatter, it can turn back from binary, into usable variables.

CHAPTER 7

BIBLIOGRAPHY

A. REFERENCES

Alliance, O.H., 2010. Android

Andrade, A., 2015. Game engines: A survey. EAI Endorsed Trans. Serious Games, 2(6), p.e8.

Bandung, S.T.T., Tunggal, S.B. and Muttaqien, S.D.K., Intel 4004.

Burks, A.W., 1947. *Electronic computing circuits of the ENIAC*. Proceedings of the IRE, 35(8), pp.756-767.

CryEngine, (2022). *Documented tutorials of CryEngine.* Tutorial as made by CryEngine professionals. Available at : https://docs.cryengine.com/display/CEMANUAL/Beginner%27s+Guide (Accessed: 18 July 2022)

Douglas, A.S., *OXO (video game).*

FinancesOnline, (2022). *37 Crucial Pokemon Go statistics: 2022 Data on Downloads, Revenue & Usage.* Available at: https://financesonline.com/pokemon-go-statistics/#1 (Accessed: 14 July 2022)

Fish, C. 2021. *The History of Video Games.* White Owl Pen&Swords Book Ltd, Great Britain.

Freiberger, P. A. and Swaine, . Michael R. (2015) *EDSAC* Encyclopaedia Britannica Available at : https://www.britannica.com/technology/EDSAC (Accessed: 11 July 2022)

Freiberger, P. A. and Swaine, . Michael R. (2022) *ENIAC* Encyclopaedia Britannica Available at : https://www.britannica.ccom/technology/ENIAC (Accessed: 11 July 2022)

Fritts, J., 2013. *History of computer & video games*. PowerPoint Slides retrieved from http://cs.slu.edu/~fritts/csci130/schedule/csci130_games_history.Pdf. (Accessed: 11 July 2022)

Gargenta, M. 2011. *Learning Android.* First Edition, O'Reilly Media Inc. USA.
Haas, J.K., 2014. A history of the unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE*, *483*, p.484.

Gregory, J., 2018. *Game engine architecture*. AK Peters/CRC Press.

Haigh, T., Priestley, P.M., Priestley, M. and Rope, C., 2016. *ENIAC in action: Making and remaking the modern computer*. MIT press.

Intel, (Date Unknown). *Intel's First Microprocessor. The story of intel 4004*. Available at : https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html (Accessed: 11 July 2022)

Iverson, K.E., 1962, May. A programming language. In *Proceedings of the May 1-3, 1962, spring joint computer conference* (pp. 345-351).

Ivory, J.D., 2015. *A brief history of video games. In The Video Game Debate* (pp. 1-21). Routledge.

John E. Ayers, 2003. *Digital Integrated Circuits : Analysis and Design* Illustrated Edition. Taylor & Francis. United Kingdom.

Kent, S.L., 2010. *The Ultimate History of Video Games, Volume 1: From Pong to Pokemon and Beyond... the Story Behind the Craze That Touched Our Lives and Changed the World* (Vol. 1). Crown.

Kowert, R. and Quandt, T., 2015. *Video Game Debate*. Taylor & Francis.

Krajci, I. and Cummings, D., 2013. History and Evolution of the Android OS. In *Android on x86* (pp. 1-8). Apress, Berkeley, CA.

Lendino, J., 2018. *Adventure: The Atari 2600 at the Dawn of Console Gaming*. Ziff Davis LLC, New York.

Levvvel, (2022). *Pokémon GO statistics and facts 2022.* Available at : https://levvvel.com/pokemon-go-statistics-and-facts/#:~:text=It%20went%20from%2028%20million,total%20of%20147%20million%20users. (Accessed: 14 July 2022)

Lewis, M. and Jacobson, J., 2002. Game engines. *Communications of the ACM*, *45*(1), p.27.

McCartney, S., 1999. *ENIAC: The triumphs and tragedies of the world's first computer*.

Metropolis, N. ed., 2014. *History of computing in the twentieth century*. Elsevier.

Noyce, R. and Hoff, M., 1981. *A history of microprocessor development at Intel*. IEEE Micro, 1(01), pp.8-21.

Pokemongodeathtracker, (2021). *Pokémon Go Death Tracker.* Available at : http://pokemongodeathtracker.com/ (Accessed: 14 July 2022)

Sensor Tower, (2022). *Q1 2022: Store Intelligence Data Digest*. Available at : https://go.sensortower.com/rs/351-RWH-315/images/Sensor-Tower-Q1-2022-Data-Digest.pdf (Accessed 12 July 2022)

ShackNews, (2018). *Pokemon Go sees its largest player count since 2016.* Available at: https://www.shacknews.com/article/105830/pokemon-go-sees-its-largest-player-count-since-2016 (Accessed: 14 July 2022)

Unity, (2022). *Products of Unity* Available at : https://unity.com/products (Accessed 14 July 2022)

Unity, (2022). *Learn Unity, Tutorials* Available at : https://unity.com/learn (Accessed 14 July 2022)

Unity, (2022). *Unity Frequently Asked Questions* Available at : https://unity.com/support-services (Accessed 14 July 2022)

Unity, (2022). *Unity branding trademarks and other permissions and guidelines for using Unity* Available at: https://unity3d.com/legal/branding_trademarks (Accessed: 14 July 2022)

Wikipedia (2022) "OXO Video Game" Available at : https://en.wikipedia.org/wiki/OXO_(video_game)#History (Accessed: 11 July 2022)

Wikipedia, (2022). *Android (Operating System)* Available at : https://en.wikipedia.org/wiki/Android_(operating_system) (Accessed in: 12 July 2022)

Wikipedia, (2022). *Creation Engine* Available at : https://en.wikipedia.org/wiki/Creation_Engine (Accessed in: 12 July 2022)

Wikipedia, (2022). *Dunia Engine* Available at : https://en.wikipedia.org/wiki/Ubisoft#Dunia_Engine (Accessed in: 12 July 2022)

Wikipedia, (2022). *Source* Available at : https://en.wikipedia.org/wiki/Source_(game_engine) (Accessed in: 12 July 2022)

Wikipedia, (2022). *Source2* Available at : https://en.wikipedia.org/wiki/Source_2 (Accessed in: 12 July 2022)

Wolf, M.J. ed., 2008. *The video game explosion: a history from PONG to PlayStation and beyond*. ABC-CLIO.

Yang et al. 2019, *On the strength of hair across species.* University of San Diego, Jacobs school of engineering. https://jacobsschool.ucsd.edu/news/release/2937#:~:text=These%20hairs%20vary%20in%20thickness,over%20350%20microns%20in%20diameter.

B. LINKS

[L.1] https://dictionary.cambridge.org/dictionary/english/modern - Definition of the word 'Modern'

[L.2] https://www.worldometers.info/world-population/asia-population/ - Live Population of Asia

[L.3] https://www.worldometers.info/world-population/china-population/ - Live Population of China

[L.4] https://www.worldometers.info/world-population/india-population/ - Live Population of India

[L.5] https://www.worldometers.info/world-population/us-population/ - Live Population of US

[L.6] https://www.worldometers.info/world-population/europe-population/ - Live Population of Europe

[L.7] https://www.appbrain.com/stats/number-of-android-apps - Live Count of Android Apps on Google Play Store

[L.8] https://www.openhub.net/p/android/analyses/latest/languages_summary - Statistics on Languages and Code lines used in android OS.

[L.9] https://42matters.com/google-play-statistics-and-trends - Statistics on total apps on Google Play Store and Percentage graph of total Gaming and Non-Gaming apps.

[L.10] https://www.appbrain.com/stats/number-of-android-apps - Statistics on Android apps in Google Play Store

[L.11] https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/ - Graph on percentage of Paid and Free apps on Google Play Store

[L.12] https://go.sensortower.com/rs/351-RWH-315/images/Sensor-Tower-Q1-2022-Data-Digest.pdf - Statistics on top apps, top gaming apps and categories, separated by worldwide data and different specific countries data.

[L.13] https://en.wikipedia.org/wiki/List_of_game_engines - List of game engines, both current and discontinued ones.

C. PICTURES

[F.1]Wikipedia, (2022). *Magnavox Odyssey* Available at : https://en.wikipedia.org/wiki/Magnavox_Odyssey#/media/File:Magnavox-Odyssey-Console-Set.jpg (Accessed: 11 July 2022)

[F.2] Benjamin L. (2020), *Magnavox*, Computer Museum of America Available at : https://www.computermuseumofamerica.org/wp-content/uploads/2020/06/IMG_3290-1-1024x310.jpg (Accessed: 11 July 2022)

[F.3] Wikipedia (2022), *PlayStation 1* Available at : https://en.wikipedia.org/wiki/PlayStation#/media/File:PlayStation-SCPH-1000-with-Controller.jpg (Accessed: 13 July 2022)

[F.4] Unknown Author, (Unknown Date). *Retro Arcade Machine* Available at: https://numskull.com/wp-content/uploads/Pac-Man-QA-1000x1000-01.jpg (Accessed: 13 July 2022)

[F.5] OhioHistoryCentral, (Unknown Date). *John W. Mauchly and John Presper Eckert* Available at: https://ohiohistorycentral.org/images/3/32/Mauchly%2C_John_and_Eckert%2C_J._Presper.jpg (Accessed: 13 July 2022)

[F.6] Wikipedia, (2022). *2 Pieces of the ENIAC on display on Moore School of Engineering and Applied Science* Available at: https://en.wikipedia.org/wiki/ENIAC#/media/File:ENIAC_Penn1.jpg (Accessed: 14 July 2022)

[F.7] Freiberger, P. A. and Swaine, . Michael R. (2022) ENIAC Encyclopedia Britannica Available at : https://cdn.britannica.com/95/170195-050-EFCB2F83/ENIAC-1946.jpg (Accessed: 11 July 2022)

[F.8] Intel, (Date Unknown). *Intel Microprocessor 4004* Available at: http://mail.indosingo.com/_buku_manual/_baca_blob.php?book=lain&kodegb=220px-C4004_Intel.jpg (Accessed: 14 July 2022)

[F.9] “Electronic Integrated Circuit of an Intel 8742” (2003) Wikipedia Available at : https://en.wikipedia.org/wiki/Electronic_circuit#/media/File:Intel_8742_153056995.jpg (Accessed: 14 July 2022)

[F.10] "Chip" (Unknown Date). University of Rhode Island Available at : https://homepage.cs.uri.edu/faculty/wolfe/book/images/R03/chip.gif (Accessed: 14 July 2022)

[F.11] "CHIPSET" (2022) Wikipedia Available at : https://en.wikipedia.org/wiki/Chipset#/media/File:Pentium_E2220_with_Intel_i945GC_Chipset.jpg (Accessed: 11 July 2022)

[F.12] Freiberger, P. A. and Swaine, . Michael R. (2015) EDSAC Encyclopaedia Britannica Available at : https://cdn.britannica.com/18/23618-050-EC6AC575/EDSAC-Maurice-Wilkes-computer-1947.jpg (Accessed: 11 July 2022)

[F.13] Wikipedia (2022) *OXO video game Interface – Emulated Screenshot* Available at : https://en.wikipedia.org/wiki/OXO_(video_game)#/media/File:OXO_emulated_screenshot.png (Accessed: 11 July 2022)

[F.14] Pokémon, (2022). *Pokémon Go app Logo as shown in AppStore* Available at: https://assets.pokemon.com/assets//cms2/img/video-games/video-games/pokemon_go/app_store_badge_us_135x40.jpg (Accessed: 14 July 2022)

[F.15] 1000logos, (2022). *Evolution of android logo* Available at: https://1000logos.net/wp-content/uploads/2016/10/Android-history-logo-640x547.jpg (Accessed: 14 July 2022)

[F.16] Statcounter, (2022). *Mobile Operating System Market Worldwide* Available at : https://gs.statcounter.com/os-market-share/mobile/worldwide (Accessed: 14 July 2022)

[F.17] Statcounter, (2022). *Operating System Market Worldwide* Available at : https://gs.statcounter.com/os-market-share (Accessed: 14 July 2022)

[F.18] Wikipedia, (2022). *Microsofts .NET framework logo of Version 4.5v* Available at: https://upload.wikimedia.org/wikipedia/en/0/0d/Microsoft_.NET_Framework_v4.5_logo.png (Accessed: 14 July 2022)

[F.19] Unity, (2022). *Unity logo as provided by Unity for trademark purposes* Available at: https://unity3d.com/profiles/unity3d/themes/unity/images/pages/branding_trademarks/unity-masterbrand-black.png (Accessed: 14 July 2022)

[F.20] Petros, (2022). *Unity Interface of version 2020.2.7f1*

[F.21] CryEngine, (2022). *CryEngine 5.6 logo for trademark purposes* Available at: https://www.cryengine.com/brand (Accessed: 18 July 2022)

[F.22] Petros, (2022). *CryEngine Interface of version CE5*

[F.23] Petros, (2022). *UnrealEngine 5 interface of template for 3D*

[F.24] Unknown Author, (Unknown Date). *Dunia Engine Logo* Available at: https://trivia.serendip.in/sites/trivia.serendip.in/files/styles/large/public/image_primary/Dunia_Engine_logo.jpg?itok=3gm3lF5B (Accessed: 18 July 2022)

[F.25] Steam, (2022). *Source game engine logo* Available at: https://avatars.cloudflare.steamstatic.com/f4bf325094f9adb1de1ecb72c249113b9295997d_full.jpg (Accessed: 18 July 2022)

[F.26] Steam, (2022). *Source 2 game engine logo* Available at: https://mygaming.co.za/news/wp-content/uploads/2012/08/source-2-engine-header1.jpg (Accessed: 18 July 2022)

[F.27] Python, (2022). *Current Official Python Logo* Available at: https://www.python.org/static/community_logos/python-logo.png (Accessed 21 July 2022)

[F.28] SeekLogo, (2022). *Unofficial Microsoft C# logo* Available at: https://seeklogo.com/images/C/c-sharp-c-logo-02F17714BA-seeklogo.com.png (Accessed 21 July 2022)

[F.29] Javatpoint, (Unknown Date). *Current Official Java Logo* Available at: https://static.javatpoint.com/core/images/java-logo3.png (Accessed 21 July 2022)

[F.30] Microsoft, (2022). *Official Microsoft Visual Studio Logo* Available at: https://visualstudio.microsoft.com/wp-content/uploads/2021/10/Product-Icon.svg (Accessed 21 July 2022)

[F.31] Don Ho, (2022). *Notepad++ Logo* Available at: https://notepad-plus-plus.org/images/logo.svg (Accessed 21 July 2022)

[F.32] Wizcase, (2021). *Logo of Microsofts IDE, Visual Studio* Available at: https://www.wizcase.com/wp-content/uploads/2021/05/visual-studio-logo.jpeg (Accessed 20 July 2022)

[F.33] Netbeans, (2022). *Logo of Apache Netbeans* Available at: https://netbeans.apache.org/images/apache-netbeans.svg (Accessed 21 July 2022)

[F.34] Eclipse, (2022). *Logo of Eclipse IDE,* Available at: https://www.eclipse.org/ide/images/eclipse-logo.png (Accessed 21 July 2022)

[F.35] Krita, (2022). *Logo of Digital Drawing program, Krita with minor background adjustment for better visibility,* Available at: https://krita.org/wp-content/themes/krita-org-theme/images/krita-logo.png?v2022 (Accessed 21 July 2022)

[F.36] GooglePlayStore, (2022). *Logo of Ibis Paint X digital drawing app, from google play store* Available at: https://play-lh.googleusercontent.com/s8moWkCF9wE-ynJgNyq8k3uhhVlbQLdphqTYJWkrsLRxkFZxx9FvykHmwXYmTl_h0l8 (Accessed 21 July 2022)

ΠΑΡΑΡΤΗΜΑ Α

Code section for filling the cooldown bar. If it is full it stops running until empty again.

```
1 reference
IEnumerator fillSlider()
{
    for ( ix = MinValue; ix <= MaxValue;)
    {
        isRunning = true;
        yield return new WaitForSeconds(0.06f);
        ix = ix + 0.06f;
        sliderValue.value = ix;
        if (ix > MaxValue)
        {
            isfull = true;
            isRunning = false;
        }
    }

}
```

Code section for the attack button

```
public void OnAttackButton()
{
    if (sliderFill.isfull)
    {
        sliderFill.isfull = false;
        sliderValue.value = 0;
        bool isDead = enemyUnit.TakeDamage(playerUnit.damage);

        enemyHUD.SetHp(enemyUnit.HPcurrent);

        if (enemyUnit.HPcurrent <= 0)
        {
            state = BattleState.WON;
            EndBattle();
        }
    }
}// on attack button ends here
```

BattleHUD Initialization and Stats

```
Unity Script (2 asset references) | 2 references
public class BattleHUD : MonoBehaviour
{
    public Text nameText;

    Unity Script (5 asset references) | 18 references
    public class Unit : MonoBehaviour
    {
        public string unitname;
        public int level;

        public int damage;
        public int healpower;
        public int SpecialDamage;

        public int HPmax;
        public int HPcurrent;

        3 references
        public bool TakeDamage(int dmg)
        {
            HPcurrent -= dmg;
            if(HPcurrent <= 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        1 reference
        public bool HealDamage(int heal)
        {
            HPcurrent += heal;
            if(HPcurrent <= 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

Code section that calculates the damage taken and return whether the unit died or is still alive.

Code section for dialogue. The dialogue manager has variables that are filled with the dialogue within the Unity interface. This dialogue takes the correct texts and makes it appear. Could be

```
Unity Message | 0 references
public void Update()
{
    if (checkcheckdialogebox.checkdialoguebox == false)
    {
        if (FlagA == 4)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text2);
            FlagA = 5;
        }
        else if (FlagA == 5)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text3);
            FlagA = 6;
        }
        else if (FlagA == 6)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text4);
            FlagA = 7;
        }
        else if (FlagA == 7)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text5);
            FlagA = 8;
        }
        else if (FlagA == 8)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text6);
            FlagA = 9;
        }
        else if (FlagA == 9)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text7);
            FlagA = 10;
        }
        else if (FlagA == 10)
        {
            FindObjectOfType<DialogueManager>().StartDialogue(text8);
            FlagA = 11;
        }
        else if (FlagA == 11)
        {
            FlagA = 0;
            StartCoroutine(triggerintrigger());
        }
    }
```

made better with a for loop.

Code section that opens the dialogue box

```
void Start()
{
    sentences = new Queue<string>();
    checkdialoguebox = false;
    animator.SetBool("IsOpen", false);
}
```

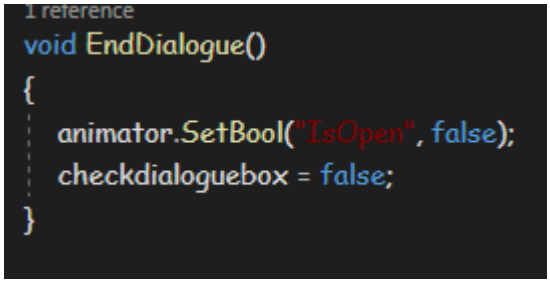
Code section that queues the dialogue sentences

```
public void StartDialogue (Dialogue dialogue)
{
    sentencescount = 1;
    animator.SetBool("IsOpen", true);
    checkdialoguebox = true;

    nameText.text = dialogue.name;

    sentences.Clear();

    foreach (string sentence in dialogue.sentences)
    {
        sentences.Enqueue(sentence);
    }

    DisplayNextSentence();
    sentencescount = 0;
}
```

```
public void DisplayNextSentence()
{
    FindObjectOfType<AudioManager>().Play("NextSentence");
    if (sentences.Count == 0)
    {
        EndDialogue();
        return;
    }
    string sentence = sentences.Dequeue();
    StopAllCoroutines();
    StartCoroutine(TypeSentence(sentence));
}


    1 reference
    IEnumerator TypeSentence (string sentence)
    {
    DialogueText.text = "";
        foreach (char letter in sentence.ToCharArray())
        {
            DialogueText.text += letter;
        yield return new WaitForSecondsRealtime(letterPause);
        }
    }
```

Code section that fetches the next sentence from the script that contains the dialogue and makes it appear on the screen.

Code section that ends and closes the dialogue



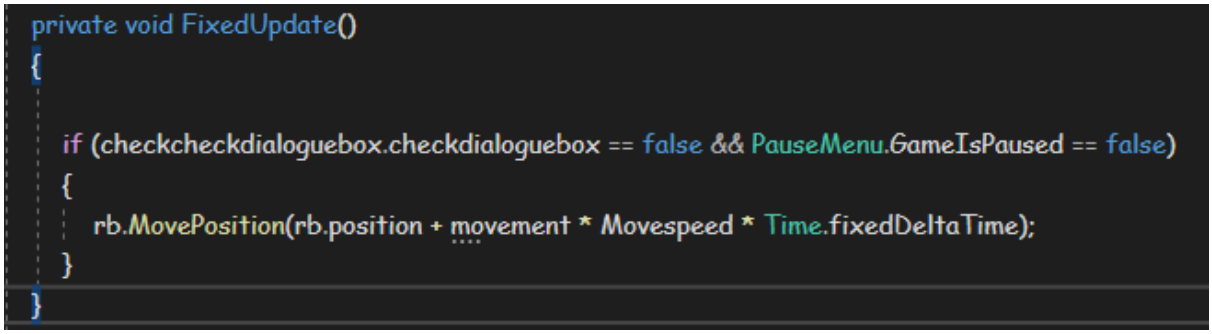
Simple collider trigger for dialogue

```
public class ColliderTrigger : MonoBehaviour
{
    public Dialogue dialogue;

    Unity Message | 0 references
    void OnTriggerEnter2D(Collider2D collider)
    {
        FindObjectOfType<DialogueManager>().StartDialogue(dialogue);
    }
}
```

Code section that changes the parameters and variables for animation purposes

```
Unity Message | 0 references
void Update()
{
    if (checkcheckdialoguebox.checkdialoguebox == false && PauseMenu.GameIsPaused == false )
    {
        animator.SetBool("IsOnDialogue", false);
        //movement.x = Input.GetAxisRaw("Horizontal");
        //movement.y = Input.GetAxisRaw("Vertical");

        //animator.SetFloat("MoveX", movement.x);
        //animator.SetFloat("MoveY", movement.y);
        animator.SetFloat("Speed", movement.sqrMagnitude);

        if (movement.x == 1 || movement.x == -1 || movement.y == 1 || movement.y == -1)
        {
            animator.SetFloat("LastMoveX", movement.x);

            animator.SetFloat("LastMoveY", movement.y);
        }
    }
    else
    {
        animator.SetBool("IsOnDialogue", true);
        animator.SetFloat("Speed", 0);
    }

}
```

Code section that calculates the speed of moving



```
private void FixedUpdate()
{

    if (checkcheckdialoguebox.checkdialoguebox == false && PauseMenu.GameIsPaused == false)
    {
        rb.MovePosition(rb.position + movement * Movespeed * Time.fixedDeltaTime);
    }
}
```

Code section that dictates the direction the character will move

```
public void MoveUp()
{
    movement.y = 1;
    animator.SetFloat("MoveY", 1);
}

0 references
public void MoveDown()
{
    movement.y = -1;
    animator.SetFloat("MoveY", -1);
}

0 references
public void MoveLeft()
{
    movement.x = -1;
    animator.SetFloat("MoveX", -1);
}

0 references
public void MoveRight()
{
    movement.x = 1;
    animator.SetFloat("MoveX", 1);
}

0 references
public void OnPointerUp()
{
    movement.x = 0;
    animator.SetFloat("MoveY", 0);
    animator.SetFloat("MoveX", 0);
    movement.y = 0;
}
```

Code section that moves the camera parallel to the character

```
void LateUpdate()
{
    if (scene == 1)
    {
        if (target.transform.position.y < 0)
        {
            transform.position = new Vector3(transform.position.x, transform.position.y, transform.position.z);
        }
        else
        {
            transform.position = new Vector3(transform.position.x, target.transform.position.y, transform.position.z);
        }
    }else if (scene == 2)
    {
        transform.position = new Vector3(target.transform.position.x, target.transform.position.y,transform.position.z);
    }
    else if (scene == 3)
    {
        transform.position = new Vector3(target.transform.position.x, transform.position.y, transform.position.z);
    }
}
}
```

Vector3 Tra
The world spac

Code section that places an interface in Unity where the developer can add dialogue and other sentences directly from the UI. It is a string that contains the sentences that will be used by the earlier code to display the dialogue

```
[System.Serializable]

public class Dialogue
{
    public string name;

    [TextArea(3, 10)]
    public string[] sentences;
}
```