PM

Keepers of Intheris:
Story and Aesthetics

MASTER PROJECT

Tatiana Severim Vieira
MASTER IN COMPUTER ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

January | 2017

Keepers of Intheris:
Story and Aesthetics

MASTER PROJECT

Tatiana Severim Vieira
MASTER IN COMPUTER ENGINEERING

SUPERVISOR
Sergi Bermúdez i Badia

CO-SUPERVISOR
Eduardo Leopoldo Fermé

UNIVERSIDADE da MADEIRA

Keepers of Intheris: Story and Aesthetics

Tatiana Severim Vieira

Constituição do júri de provas públicas:

Karolina Baras, Prof.ª Auxiliar da Universidade da Madeira, Presidente

Pedro Campos, Prof. Auxiliar da Universidade da Madeira, Vogal

Sergi Bermúdez i Badia, Prof. Auxiliar da Universidade da Madeira, Vogal

Março 2017

Funchal – Portugal

ABSTRACT

The recent releases of new and affordable software for developing video games, combined with an educated population and the large amount of knowledge available for free on the internet has created an explosion in the number of video games being developed and released independently. This project details part of the development of a video game based on a previously existing board game. This project focuses on the aesthetics and story, two elements of the game design tetrad that, all games possess in varying degrees. During this project the interface of the video game was designed, prototyped, tested and iterated upon. The game world, its props and characters were also modeled and a background story was established for the world, the monuments on it and, for each character and where they came from. The development of this project followed processes applied and thought by professionals of the game industry and was successful to implement a proof of concept of a video game. This work provides a deep insight for anyone that wishes to understand the game industry and learn the process of creating a game.

RESUMO

Os lançamentos de novos pacotes de *software* a preços acessíveis para o desenvolvimento de vídeo jogos, juntamente com uma população formada e a grande quantidade de conhecimento disponível de graça na internet geraram uma explosão no número de vídeo jogos a serem desenvolvidos e lançados independentemente. Este projeto detalha parte do desenvolvimento de um vídeo jogo baseado num jogo de tabuleiro previamente existente. Este projeto centra-se na estética e história, dois elementos da tétrade do desenho de jogos que todos os jogos possuem em graus variados. Durante este projeto a interface do vídeo jogo foi desenhada, foram criados protótipos desta, estes foram testados e iterados. O mundo do jogo, os objetos neste e as personagens foram modelados e uma história de fundo foi criada para o mundo, os monumentos deste, para cada personagem e a sua origem. O desenvolvimento deste projeto seguiu processos aplicados e ensinados por profissionais da indústria dos vídeo jogos e foi bem-sucedido na implementação de um vídeo jogo. Este trabalho apresenta uma visão aprofundada para que qualquer pessoa que deseje aprender sobre a indústria de jogos e o processo de criação destes.

KEYWORDS

Game development

Board game

Interface design

3D production pipeline

Unity3D.

PALAVRAS-CHAVE

Desenvolvimento de jogos

Jogo de tabuleiro

Desenho de interfaces

Processo de produção de 3D

Unity3D

ACKNOWLEDGEMENTS

Firstly, I would like to thank and acknowledge my parents and brothers who, are very important to me and always believed in me. Without them and their support on all levels I would not be here today.

I would also like to thank João Serina for always believing in me, for his patience and for helping me in everything I needed even if sometimes I did not know that I needed help.

To my supervisors, Professor Sergi Bermúdez i Badia and Professor Eduardo Leopoldo Fermé, I would like to thank for all the time they dedicated to this project, even during their holidays and, for always being ready and eager to help.

I would like to thank Yuri Almeida because without him this project would not exist and for his patience and time when planning parts of both our projects.

Furthermore, I would like to thank Juan Ponte for his assistance in improving the quality of the 3D renders and for making my days better with his, often, obscure humor.

Finally, I would like to thank all my friends, relatives and former professors that always believed in me and supported me throughout this journey.

CONTENTS

LIST OF FIGURES

ACRONYMS

CE5 – Cry Engine 5

DAP – Design and Aesthetics Project

FBX - A type of file that has everything an OBJ file has and more, such as animations and skeleton data.

FPS – First Person Shooter

GGP – General Game Project

GIP – Game Implementation Project

GUI – Game User Interface

HCI – Human Computer Interaction

MMORPG – Massive Multiplayer Online Role Playing Game

MOBA – Massive Online Battle Arena

MTP – Mechanics and Technology Project

OBJ - A type of file that has all the data of a 3d geometry, such as position of each vertices and texture vertices

PVP – Player versus Player

RPG – Role Playing Game

RTS – Real Time Strategy

UE4 – Unreal Engine 4

UI – User Interface

UX – User Experience

1 INTRODUCTION

1.1 MOTIVATION AND OBJECTIVES

This work started as a Bachelor Degree's final project in Computer Engineering at the University of Madeira which will be referred to as General Game Project (GGP). The GGP was shared between two students, Tatiana Vieira and Yuri Almeida. The goal of the GGP was to create a board game to validate a game concept and to establish the requirements for its digital implementation, the Game Implementation Project (GIP). Due to the sheer amount of work and the complexity of the GIP, the workload was divided between the two students.

The main purpose of the GIP is to implement a working digital prototype of the board game that was previously developed with all its rules and requirements (GGP). The biggest challenges to overcome were to ensure that the spirit of the board game was maintained in the digital implementation, learning and following the development process used in the game industry and collaborating in a multidisciplinary team.

This project, the Design and Aesthetics Project (DAP) into five sections each approaching a major topic: state of the art, the board game (GGP), game story development, game interface and the creation pipeline of the 3D assets. This last section was also used as a learning experience in 3D asset creation for games and all the knowledge and experience gained through the development of the DAP will contribute to finding a full-time position as a 3D Designer in a game development company.

1.2 PROJECT DIVISION

The GIP division was based on the Elemental Tetrad of Games [1] which divides a game into four components: Aesthetics, Story, Mechanics and Technology. The DAP, focusses on the development and implementation of the first two components while, the other two can be found in the project Keepers of Intheris: Mechanics and Technology [2] which, will be referred to as Mechanics and Technology project (MTP). Figure 1 shows the complete game development responsibilities. Part of the game design document was done previously when creating the board game.

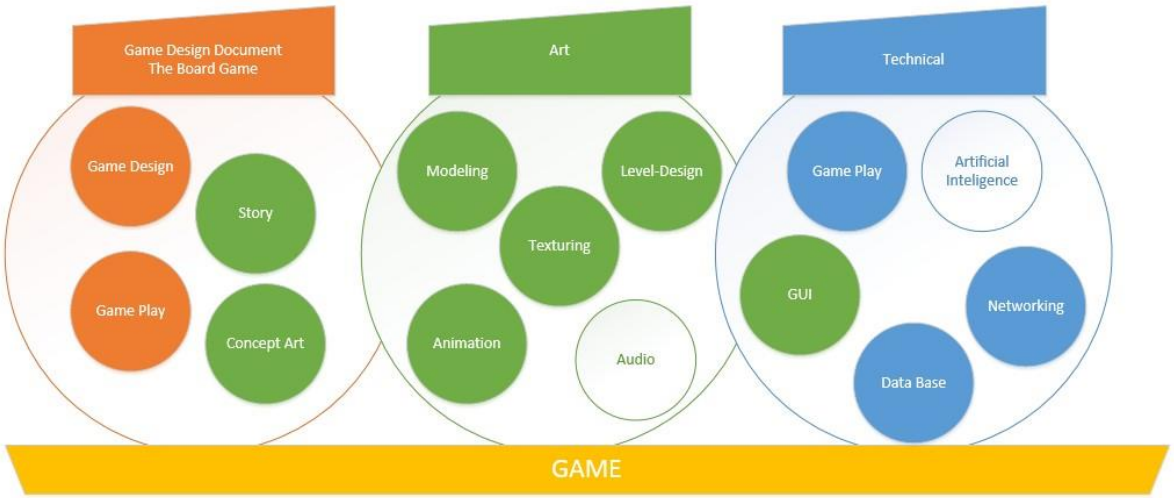


Figure 1 – Project's game development responsibilities; Orange represents everything done while developing the board game (GGP); Green represents the scope of the DAP; Blue represents the scope of the MTP.

This project is responsible for multiple aspects. Firstly, the development of the interface design and implementation. The interface is divided into two parts, the menus which include all aspects of the interface before starting a game and the in-game interface which includes all the elements required to play the game as in the board game. This project is also responsible for the development of the game story which, includes the story of the game's universe and the background of each character. This story also influenced the creation of the 3D assets created in this project. These were created according to a 3D production pipeline which started with a piece of concept art and ended with a final 3D game asset. Another responsibility of this project was to integrate the animations and particles of the game with their corresponding mechanics inside the game engine, Unity3D [3]. Finally, this project had to ensure that every asset created could be integrated into Unity3D but also that it was compatible with the MTP. Figure 2 shows how both projects were connected and the solid green shapes represent this project's responsibilities.

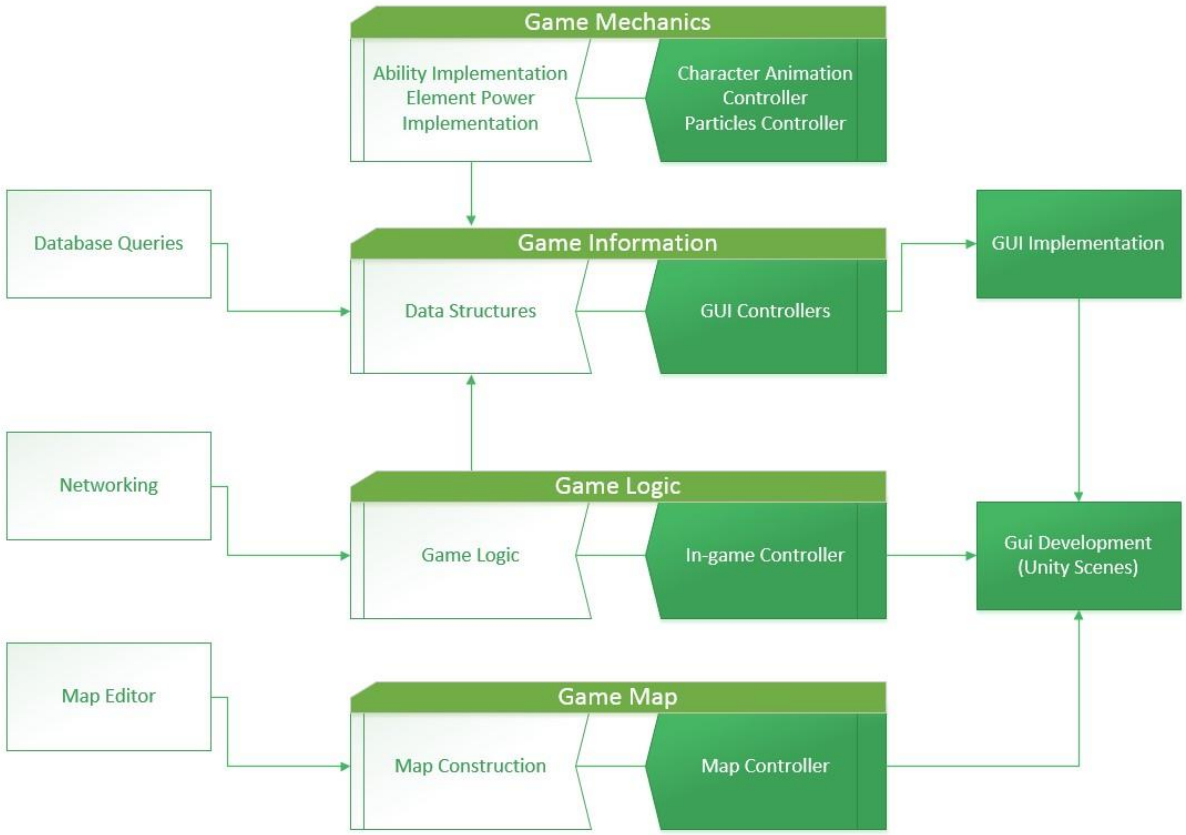


Figure 2 – GIP's architecture division (white represents the responsibilities of the MTP while solid green represents the DAP's responsibilities).

The game development was planned through a Gantt chart (Annex Gantt) that included the responsibilities of the GIP, the dependencies of each part and a deadline for their completion. Both parts of the GIP were linked to a server version control, Microsoft Team Foundation [4], which was used to integrate both source codes and to be able to work on it simultaneously.

PART I:
STATE OF THE ART

2 STATE OF THE ART

2.1 GAME INDUSTRY EVOLUTION

Videogames started out as research. At the time, the goal was to push the boundaries of computing. The first computer game, Noughts and Crosses, was created in 1952 by A. S. Douglas at the University of Cambridge [5]. Despite this, the first recognized example of a game machine was presented in 1940 at the New York World's Fair and was called Nim [6]. Many other followed this path starting out as research but ultimately becoming entertainment. In 1962, another early computer game was created by MIT researchers but it could not be distributed since the hardware was too expensive to mass produce. By 1964 the programming language BASIC was developed which became instrumental in the creation and sharing of videogames [5].

By the 1970's video games had moved from research to a form of entertainment. The arrival of arcade games and home consoles due to Atari, further contributed to the fast adoption of videogames [5], [6]. The development of videogames continued to climb and by 1978 arcade videogames were so popular that they were part of everyone's life. During the early 80's, several iconic games were created, such as Pac-man, Donkey Kong (which later led to the development of Mario) and Tetris [5].

During those early days, the goal was not to inspire the audience, stir up emotions or expose them to thought provoking realities or perspectives but, simply to entertain and amuse. A major milestone in breaking away from simple entertainment into the rich works of art of today was the appearance of the text-based adventure games in 1976. Colossal Cave Adventure, developed by Will Crowther and Don Woods, combined elements from video games with tabletop roleplaying video games to create a completely new experience (Figure 3 - A) [7]. Narrative was used as a tool to overcome the limitations of that time. Computers could not render complex landscapes such as today but, with a detailed description, the player's imagination could do it. With the release of Donkey Kong in 1981, the world saw one of the first complete narratives in video games. A giant ape (Donkey Kong) kidnaps a woman while her boyfriend (Jumpman, later named Mario) attempts to rescue her over the course of several stages / levels (Figure 3 - B). The gameplay and narrative walked hand in hand since, when the protagonist failed, they antagonist smiled, the damsel in distress often shouted for help (through a speech bubble) until the player reached her. As this happened, a heart would appear between them just to be shattered by the villain as he escaped once again [7]. These simple narrative elements, were more than sufficient to convey the classical story of a damsel in distress and the hero that saved her.

Donkey Kong was also one of the first platform games. Due to technical limitations, these early platform game levels were constrained to the size of the screen. The next step in platform games was side scrolling which removed these early limitations. As technology evolved, games such as Q*bert, in 1982, started to emerge. These games had an isometric view giving the illusion of being three-dimensional but were in fact still 2D games (Figure 3 - C) [8].

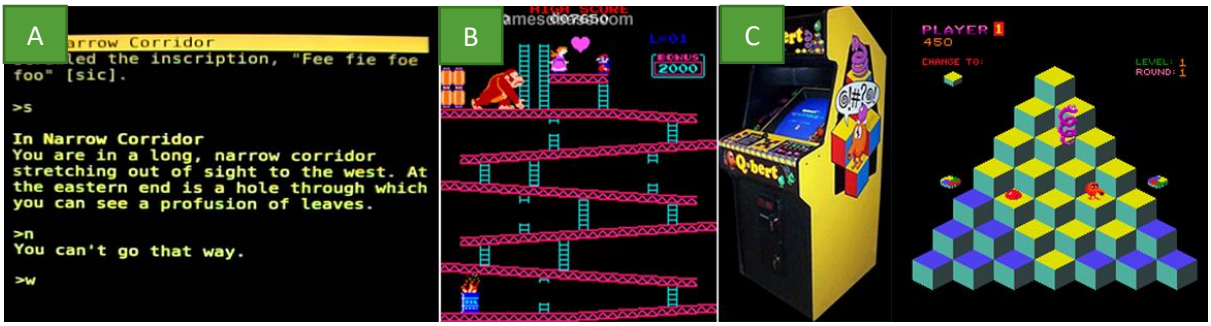


Figure 3 – (A) Colossal Cave Adventure screen; (B) Donkey Kong [7]**; (C) Q*bert in the arcade** [8]**.**

The videogame fever continued until the videogame's crash of 1983 [6]. This year was a turning point for computer and console games. While consoles started to decline in popularity, home computers started to grow. This was undoubtedly fueled by the BASIC programming language and magazines that shared videogame code for anyone that had a home computer [6].

The 80's were also an important decade for videogames due to the fast evolution in processing power. In 1983 8-bit consoles were released, bringing along more detailed colors, sound and processing power. By 1987 16-bit processing was a reality, taking another giant leap in these features [6].

In 1986, Nintendo released the game Metroid that would later lead to many more Metroid games. Metroid is an influential game not because of its narrative but, because of how it was able to deliver a clear narrative through aesthetics. The game takes place in the Planet Zebes and through the use of dark color pallet, eerie music and sound effects and, the lack of dialogue, the creators of the game instantly convey a clear message and the setting of the narrative: this is a harsh world and the player is all alone (Figure 4 - A) [7], [9]. Metroid also employed a technique used in other media, the protagonist's face was covered and never spoke. This allowed players to more easily project themselves onto the role of the protagonist [9].

The Legend of Zelda, considered the forerunner of role-playing games (RPG) was also release in 1986. One of the main innovations brought by Zelda was a non-linear narrative. This feature was adopted by many more games through the years and became an essential part of future RPGs. The influence of the game's fantasy setting and musical style can still be seen in current video games (Figure 4 - B) [10].

Figure 4 – (A) Legend of Zelda 1986 [11]**; (B) Metroid** [12]**.**

Technology continued to evolve and in 1993 the internet became part of the public domain. This was crucial to the sharing of information but also to the rise of multiplayer games. The main boosters to

the popularity of these games were Doom and Pathway to Darkness also released in 1993. Doom and Wolfenstein 3D brought along the first 3D concepts for games (Figure 5 – A and B). At the time, these games took place in a three-dimensional space but many of the game elements were still 2D sprites [13], [14].

The following year saw the release of Warcraft: Orcs and Humans which, was one of the first strategy games with detailed missions. This franchise would later lead to the development of the most popular massively multiplayer online game of all time, World of Warcraft [5].

By 1996, as a successor of Doom, Quake was release. It managed to overcome the technical limitations and all the elements in the game world were now 3D (Figure 5 - C).

Figure 5 – (A) Wolfenstein 3D [13] **(B) Doom** [14]**; (C) Quake I** [15]**.**

Video games continued their course until 2007. This year was marked by the release of smartphones and app stores. A new era begun in a new system with new features and restrictions. This new age of gamming fueled an inevitable evolution in game interfaces as smartphones had smaller screens but still needed to present a lot of information during game sessions.

Video game aesthetics have been a hot topic in the last few years as consumers and reviewers mistook aesthetics for graphics (graphical fidelity). As technology evolved, the graphical fidelity that games achieved also improved. This led to the creation of characters that were visually closer to reality and moved them away from the uncanny valley. As this happened, games looked better and consumers started to associate the technological evolution with better looking games. This made them ask for better graphics and the game industry provided better graphics. However, many games that did not look better with this technological evolution and,this led to many video games looking bland and generic [16]. In the early days of the video game industry, audio, visuals, gameplay and narrative were all on the same level or very close to each other. In Figure 6 we can see how Legend of Zelda evolved but kept the same aesthetic over the years. While all of them have come a long way, visuals and audio are the ones that benefited the most from the technological evolution. As these aspects of games evolved they started to grow apart in cohesion since some developers and designers wanted to simply incorporate the latest innovation into their games sometimes without care or thought [16], [17].

Figure 6 – Evolution of Zelda through the years [11]

In the last two years, virtual reality appeared to be the future of videogames but currently it is starting to lose popularity due to its high costs and a lack of games using this technology. Regardless of what the future holds for videogames it will surely be entertaining [6].

While the narrative is often an after though in many player versus player (PVP) games, it is essential for player's emersion and enjoyment. In this game, the narrative was one of the first elements to be developed. This was also important to define the setting of the game and its aesthetics.

2.1.1 THE EVOLUTION OF STRATEGY GAMES AND THE RISE OF MOBAS

Since the late 90's, players have been able to compete with each other and this led to the rise of player versus player games. From this competitive atmosphere, two game genres emerged, the strategy games like StarCraft, Warcraft, Command & Conquer and first person shooters such as Doom, Quake and Team Fortress. These games allowed players to compete in an arena where they could show off their skills and knowledge [18]. Due to the scope of the DAP, the focus will be on strategy games. This genre can be divided in two gameplay styles, turn-based (TBS) and real-time strategy (RTS).

Turn based games come from the initial concepts of games. From the early games such as checkers to the first multiplayer game, Empire, released in 1973 [6]. In these games, players play in turns where each can decide where to move and what to do on their turn and then wait for the other player to play. There are many games that can fit in this category, while in Civilization, players can conquer the world from a perspective of a god or an observer, in XCom the player is closer to the action and can see their troops up close and feel like they are on the field leading them. Even though many games in this genre have real time mechanics, this subgenre focus more on strategy and planning than on quick execution (Figure 7).

Figure 7 – (A) Empire game screen [19]**; (B) Civilization V gameplay** [20]**; (C) XCom gameplay** [21]**.**

Real-time strategy games on the other hand, create anxiety and adrenaline rushes which is something that a lot of players enjoy. Games like Dune II, StarCraft and Warcraft led to the creation of this subgenre [18]. In this type of games, players can deploy units and create structures under their control to secure areas of a map and can destroy their enemy's base and troops. Since the amount of tasks a player must perform to be able to succeed in an RTS can be very demanding, complex user interfaces had to be created to help the players cope with this big challenge (Figure 8).

Figure 8 – (A) Dune II gameplay [22]**; (B) StarCraft 1 gameplay; (C) Warcraft 1 gameplay.**

Some games also combine these two subgenres. One of the most known examples are the Total War series which allows players to play both subgenres at the same time. In a first phase, players are faced with a map where they move around and start spying and exploring the world. In a second phase, they must go into battle, to conquer new territories or defend their own. Before the battles start, the player places their units on the map and once the battle starts it takes place in real time [23].

With StarCraft and Warcraft, Blizzard released a Map editor, where players could create their own levels and game ideas. It did not take long for the most curious players to start creating their own maps and it was then that another game subgenre came up, Multiplayer Online Battle Arena (MOBAs). It started as a StarCraft mini-game called Aeon of Strife and then Defense of the Ancients, another mod of a Blizzard game, Warcraft III, that continued to evolve this new subgenre (Figure 9 - A) [18], [24].

In this subgenre, the player controls a single hero as part of a team. The objective is to destroy the enemy's base using the hero's unique abilities and the team's minions. This game completely relies on the player's quick reactions and the number of actions they can perform in a certain amount of time. Players have to manage playing as a team and evolving their hero as game time progresses (Figure 9 - B). Games such as League of Legends [25] and DOTA2 [26] were the first MOBA's released. Heroes of the Storm [27], a more recent MOBA, changed some of the base rules/mechanics. Players no longer need to upgrade the armor on their heroes as they level up and experience is shared between the team instead of being personal. Another MOBA that drew player's attention was SMITE [28]. Despite not changing the game mechanics like Heroes of the Storm, SMITE chose to change the perspective of the player from an isometric view to a third person view [18]. This completely changed the way players experienced a MOBA.

Figure 9 – (A) Warcraft III map editor; (B) MOBA map explanation.

MOBAs were one of the big inspirations in the board game. Since the board game shared many mechanics and system with MOBAs, its interface will also be influenced by the interfaces of MOBAS.

The board game was created with the goal of incorporating the most enjoyable mechanics from existing games into a single cohesive experience. The digital game, had to go beyond this and provide a new experience to players that had already experienced everything that strategy games had to offer.

2.2 GAME INTERFACES

The interface is what allows the interaction between the player and the game itself [29]–[31]. Contrary to popular belief, the interface is not just what can be seen on a screen but, it also includes all the controllers that help the player bridge the gap between the physical and the virtual world [29].

The fields of 'User interface' (UI) and 'User Experience' (UX) are often confused. The UI is a set of tools that the player uses to interact with the game. On the other hand, UX refers to how easy and pleasant it is to establish the interaction. In games, UX is mostly known as 'Gameplay'. Often, the line between game interface and gameplay is blurry [29], [32]. Gameplay is defined as the emotionally positive experience that users get when interacting with the UI and its visual elements, information architecture and navigation [29]. Sometimes, the interface may be user-friendly, while the gameplay is challenging and, in other cases, the interface may be difficult to use in order to further separate players based on their skill, without compromising the enjoyment of its audience [32].

In video game design, one of the most important things is to keep a player entertained and engaged in the game [33]. Despite being fundamental, the UI has been one of the few things that has been the most neglected in the past fifteen years [30]. Other fields, such as physics, artificial intelligence and even 3D modeling have been constantly evolving in games [29], [30].

In many games, UI design moves away from traditional UI designs because it has to accommodate an additional element, fiction, which is an avatar of the actual player [34]. It becomes a portal to the inside of the game bridging fantasy and reality. Through the years, terminologies have been used to identify each type of interface depending on how linked to the game geometry and narrative they are [35], [36]. They are divided into four different types: diegetic, non-diegetic, spatial and meta.

Diegetic interface elements exist inside the game and can be interacted with by the player or its avatar. If this type of interface is well executed, it will provide a more immersive experience to the player, enhancing their experience of the game. An example is the eagle vision highlight in Assassin's creed by Ubisoft [37] where the player can see targets and track patrols behind walls (Figure 10 – A). On the

other hand, non-diegetic interfaces are the ones that are rendered outside the game world and are only seen or heard by the player. These are the most common game interfaces (Figure 10 – B). Spatial interfaces are composed by elements that are inside the game's 3D but are not necessarily an entity of the game world itself, for example when the amount of damage done to a target is shown over its head (Figure 10 – C). Meta representations in the interface are elements that can exist in the game world but are not necessarily seen spatially by the player, for example blood spatter on the camera to indicate the player has been damaged (Figure 10 – D) [34], [36].

Figure 10 – (A) Diegetic interface example – Assassin's Creed eagle's vision [38]**; (B) Non-diegetic interface - DOTA 2; (C) Spatial interface – World of Warcraft Scrolling Combat Text; (D) Meta interface – Blood Spatter in Call of Duty** [39]**.**

Game UIs come in many forms and should never be limited by static game elements, there is also audio, animations and even game effects to make the player aware of what is going on inside the game [36].

Video games often combine different types of interfaces to provide feedback to the player in several ways. This feedback is sometimes redundant but, contributes to a higher awareness of the game state. In the game being developed, diegetic, non-diegetic and spatial interface elements were combined to provide a richer experience while still displaying all the necessary information.

2.2.1 EVOLUTION OF GAME INTERFACES

Over the years, there has been a trend on game UIs of the same genre. The similarities include style, screen layout and the use of UI elements [30]. Many game genres have been emerging, but the most played are Massive Multiplayer Online Role Playing Games (MMORPG), Real Time Strategy games (RTS) and First Person Shooters (FPS) [40].

MMORPGs have been using a standardized layout based on Everquest (1999) [41], [42] and later updated with the launch of another game, World of Warcraft (2004) [43], [44]. This type of games usually shows a health bar, an action bar and a mini map to help navigate around the open world (Figure 11 – A and B). There are also a lot of player modifications (mods) that can be created for this kind of interfaces, making the interface more versatile and deeply customizable [45]. With these new features, some researchers in the field of HCI think that there is a lot to learn from game UIs that can help to create better UIs outside of the game industry [46].

FPSs have been presenting the same information since the first game of this genre was launched, Wolfenstein 3D (1992) (health points, ammunition, armor points, score and equipped weapons) (Figure 11 – C and D) [13], [35], [47]. The launch of the second FPS game, Doom (1993) brought new controls [14]. As time went by, some interface elements started to emerge in this type of games only in some contexts, showing up only when needed. They became part of the game world, sometimes even mimicking a heads-up display (HUD) in an aircraft and creating a more immersive gameplay experience [35], [47].

RTSs have been using the same set of controls and UI elements since Warcraft II was launched [48]. The UI elements of these games include unit health bars, amount of resources, small action bars and a mini map to improve strategy and navigation in the game (Figure 11 - E). This pattern has been repeating on every Blizzard RTS and it has also been adopted by other RTS games [49]. As mentioned previously, these games lead to the development of the MOBA genre. In these games, the UI was maintained and as they evolved, some mechanics were removed and so were their representations on the interface (Figure 11 - F).

Figure 11 – (A) Everquest Interface; (B) World of Warcraft interface; (C) Wolfenstein3D interface; (D) Counter Strike: GO interface; (E) Warcraft III interface; (F) Heroes of the Storm interface.

Video games have been through a massive evolution in the last few decades but interfaces have barely evolved. This has its benefits but also its problems. On the bright side, once players learn the interface for one game genre they will be sufficiently proficient with other games in the genre through familiarity. On the other hand, innovation is often left behind or very slow for fear of straying too far from the familiar. This would force players to have to learn a new interface for a genre they already knew giving them the impression that this new game was needlessly complicated and inferior to the others they already knew.

2.2.2 HOW INFORMATION IS DISPLAYED IN OTHER GAMES

Even though the game being developed is a turn-based game it has several components that can also be found in other game genres. For the purposes of the DAP, the research on how information is displayed for the player focused mainly on turn-based strategy games but, it also included MOBAs, (MMO)RPGs and RTSs.

The main difference between the game implemented and the board game, is that each player controls a team of three heroes instead of having each player control a single hero. The board game has a lot of information that needs to be displayed at all times but, there is also a lot of contextual information that the player needs throughout the game. In this sub-section, research was done based on the type of information that is going to be displayed in the digital implementation according to the board game's requirements (more information can be seen in The Board Game section). The game has a hex map. In these games, this type of maps is shown in two different ways: a grid shown at all times or an overlay of what are the boundaries the player can move to. For example, in Might & Magic [50], the grid is always present but in XCOM [21], the movement boundaries are only shown when the unit is playing (Figure 12).

Figure 12 – (A) Might & Magic game grid; (B) XCom movement phase grid.

Team structures (such as buildings) are represented in games as 3D objects inside the map, usually with a health bar, a resource bar and other information as part of its spatial interface. For example, in Heroes of the Storm [27], in addition to its health, each tower has ammunition, so it has a floating bar showing the remaining ammunition (Figure 13 – A).

Since the early days of multiplayer that teams are represented by blue and red colors. Usually friendly units are colored as green or blue and enemies as red or purple (Figure 13 - B). Team units are usually represented in the interface as a series of portraits and in addition to their avatar, they also show their health and resources (Figure 13 - C).

When a unit is selected, sometimes, it is possible to see more information about it. Its list of current active abilities, state (has played/is dead/other) and a list of current negative and positive effects (known as debuffs and buffs, respectively). These effects are sometimes accompanied by animations to help the player quickly realize what is going on with their units (Figure 13 - D) [51]. Each unit is also represented simultaneously in the map as a 3D object with information as spatial interface (Figure 13 - B and C).

Figure 13 - (A) Tower with ammunitions in Heroes of the Storm; (B) Enemies represented as 3D objects and with information as spatial interface; (C) Units selected in Warcraft 3 showing their health at the bottom; (D) Water drops in Magicka 2 representing the element affecting the player.

The score is a very important part in these games. Some of the most popular MOBAs have chosen to show the score like an FPS game, where the players can see their kills, assists and deaths. Sometimes the score is also positioned at the top of the screen (Figure 14 - A).

RTS and MOBA games do not have a history list where the players can see the last attacks and recent events because more than 20 actions happen each second. These are normally shown in the turn based games, where each player plays once and it is very important to 'document' what has recently happened. Hearthstone [52] is a good example of this kind of games. It has a game history of the last few plays where each square on the side represents a player action (Figure 14 - C).

Other interesting parts of the interface are the floating combat text which gives instant feedback on health lost and gained for each unit and floating messages that provide feedback for certain events that the player needs to be aware of (Figure 14 – A and B). In fast paced games, using an ability is an instant action, the player has a target, presses a key and done, ability used. In turn based games the process is a bit lengthier. Normally, players choose the ability, then choose their targets and finally decide whether to use it or not. The result is the same, but for interface design, it is completely different. A good example of a turn based game where this is done is XCom in which, the player goes through the process described above and the ability panel provides additional information such as range and damage (Figure 14 - D).

Figure 14 - (A) Score and Floating Messages on Heroes of the Storm; (B) Floating combat text on League of Legends; (C) Match history on Hearthstone; (D) Target system in XCom 2.

Outside of the game, this interface should accommodate many more functionalities. Players should be able to change settings in their game such as video, audio and key bindings. The interface should also provide a way for players to see the information about their heroes and purchase new ones. This is usually done through a store. The game Heroes of the Storm is a good example on how this information should be displayed (Figure 15).

Figure 15 –(A) Hero information panel; (B) and (C) Game Shop; All images from Heroes of the storm

While developing the game's interface, inspiration was drawn from different games but, always considering the best way to keep the player familiar with it by using solutions from well-known games.

2.3 TEXTURING AND TEXTURE MAPS

When thinking about textures, one of the first things that often comes to mind is color, because that is normally the base of a texture, color. There are several ways to create a texture, from photographs to painting them by hand. Textures are often thought of as a single image, but in fact they are a combination of several layers (texture maps), which together form a shader.

2.3.1 COLOR MAPS

The most common color map types are the Diffuse and Detail maps. Diffuse maps are the colors seen on the model. [53] They are normally used as a base for the other texture maps and are created from different photographs of real-world materials. When there are no additional maps, the diffuse map will also simulate shadow, light and highlight, all at once [53]–[55]. For example in mobile game development, where there are more restrictions in the amount of memory and CPU usage it is not possible to use shaders that have more than two texture maps (diffuse and bump) [56].

Detail maps are used to show additional detail when seeing the model up close. By using a tiled texture together with a transparency map, it is defined where there should or should not be any detail as the camera gets closer to the model [53].

2.3.2 TRANSPARENCY MAPS

Also, known as opacity or alpha maps, transparency maps are usually stored in the alpha channel of the diffuse, specular and light maps. They are usually used for blending and for letting the texture map know where to show a certain part of its texture [53]. Since it uses the alpha channel, the only colors accepted are a full gradient from white to black, where white defines the texture as visible and black as completely invisible. This type of map is useful when creating hidden parts of a model, for example for low-poly grass. This is accomplished by creating a plane where the grass texture is applied, and its alpha map will make sure that only the grass image is visible, hiding everything else in the plane (Figure 16).

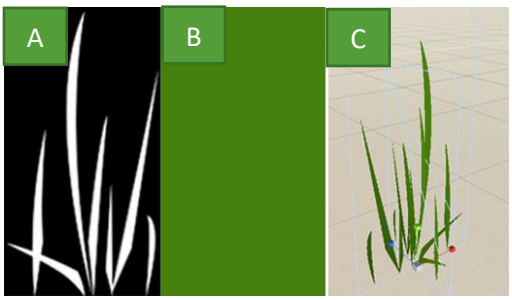


Figure 16 – (A) Alpha Map; (B) Color Map; (C) Grass model in the game engine.

2.3.3 BUMP MAPS

Textures are not limited to color maps and can be used to change almost any channel in a model's material. In addition to the Bump map itself, the most commonly known are Normal, Displacement and Height maps.

The first thing to understand about Bump maps is that the detail they create is an illusion. When looking at the model up close, it is clear that the detail is actually not there, it only appears to be. These maps are greyscale gradient images where the 3D model reads two things, white for details that appear to pull up the 3D model's surface and black for details that appear to push into the surface [57]. Bump Maps are valuable when adding small details such as wrinkles, cloth texture and imprints. Even though they do not change the model's silhouette, they still give that small detail that makes a massive difference, as long as the camera is kept at a safe distance (Figure 17 - B) [54].

Normal maps can be seen as an improved version of Bump maps (Figure 17 - A). And just like Bump maps, the details they create are also an illusion, they do not affect the silhouette of the model. While bump maps create this illusion using greyscale maps and only go up and down, Normal maps use RGB (red, green and blue) information corresponding to the XYZ axis in the 3D world. The RGB colors give the 3D application the orientation of where each polygon is facing, creating the fake detail (Figure 17 - C) [53]. It is important to notice that there are two different types of Normal Maps, the Tangent Space Normal Maps and the Object Space Normal Maps. Tangent Space Normal Maps work best for models that will be deformed during their animations (like characters) (Figure 18 - B), while Object Space Normal maps are usually used for game assets that do not need to be deformed, like rocks and other static assets (Figure 18 - A) [57], [58]. Normal maps are not so easily created in a 2D Software and are normally part of the asset creation pipeline where a normal map is baked using a high definition version of the 3D model [57].

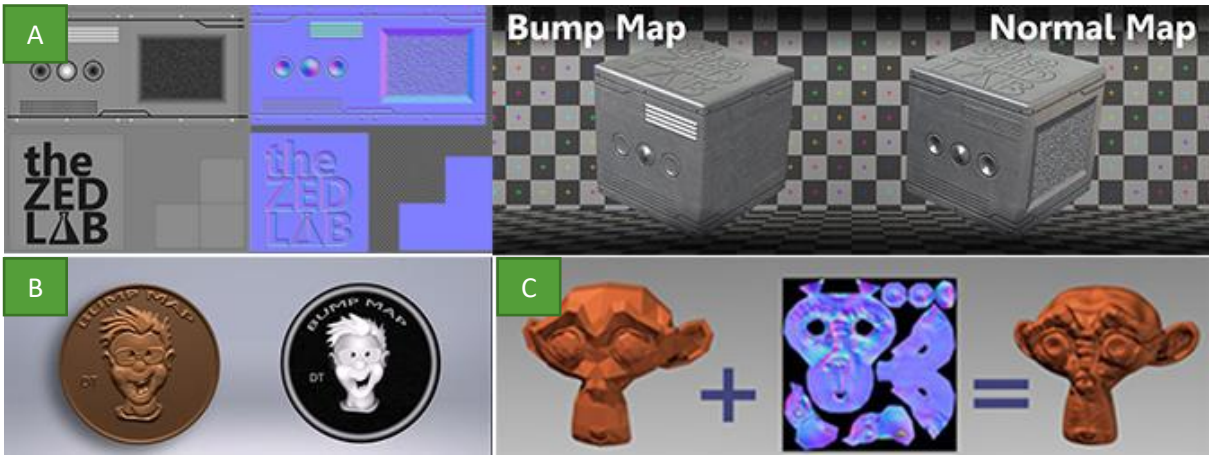


Figure 17 – (A) Bump and normal Map aspects [59] **and their texture detail differences a 3D cube** [59] **(B) Bump Map in use** [54]**; (C) Normal Map in use.**

Displacement maps do not create the illusion of detail. They are equivalent to Bump Maps and the only difference between them is that they actually modify the model's geometry when rendering. But because of this, they use a considerable amount of resources at render time (Figure 18 - C) [53], [57]. Height Maps are usually used in terrain meshes. They are equal to Displacement Maps, using a

greyscale texture to define the topography of the mesh but modify the vertex height. Game Engines often use them to define the game's terrain landscape [60].

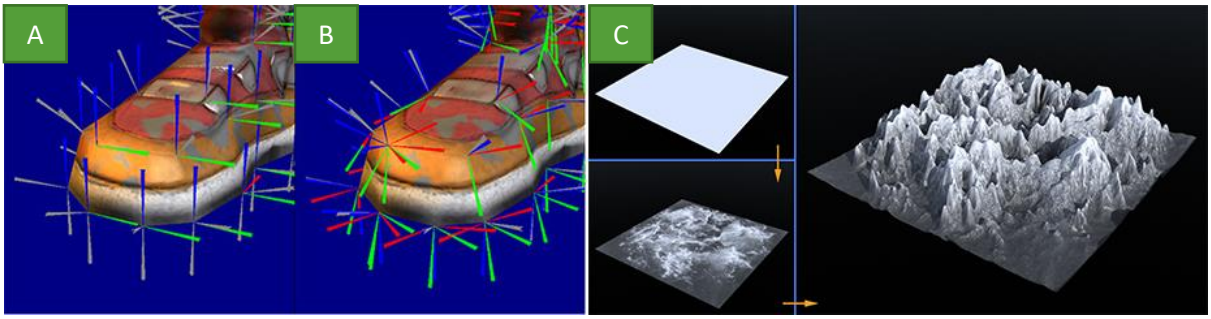


Figure 18 – (A) Object Space Normals; (B) Tangent Space Normals [61]**; (C) Displacement Map** [62]**.**

2.3.4 SPECULAR MAPS

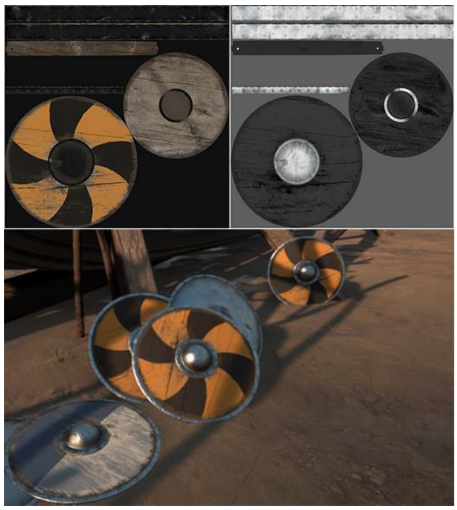


Figure 19 – Shield model in the game engine with light reflecting on the metal material [145]**.**

Specular Maps control how light reflects on a surface. These instruct the program on how shiny a certain area of a mesh should be. Using a Color Map and changing the image levels and curves it is possible to create a good specular map, making it brighter to increase the light reflection or darker to reduce the amount of light being reflected. This map also uses the alpha channel to define which parts of the mesh it will affect (Figure 19) [53], [54].

2.3.5 LIGHT MAPS

Light Maps are an efficient way to save hardware resources inside a game by storing all the pre-rendered lightning in it. The most common are Ambient Occlusion and Emissive Maps [63].

Ambient Occlusion Maps are used to create soft shadows. These make the model look as if it was illuminated with indirect light, like on a cloudy day (Figure 20 – A and B). In the texturing pipeline, artists add this map as an overlay of the diffuse or specular map, instead of being stored only as a unique map [53], [64].

Emissive Maps are also known as Glow Maps. They also use the alpha channel to know where to apply the texture in the mesh. They are used to create a 'glow in the dark' effect but do not affect the other surfaces around them [53], [63]. They are very useful when adding glow to details such as magic runes and crystals (Figure 20 – C and D).

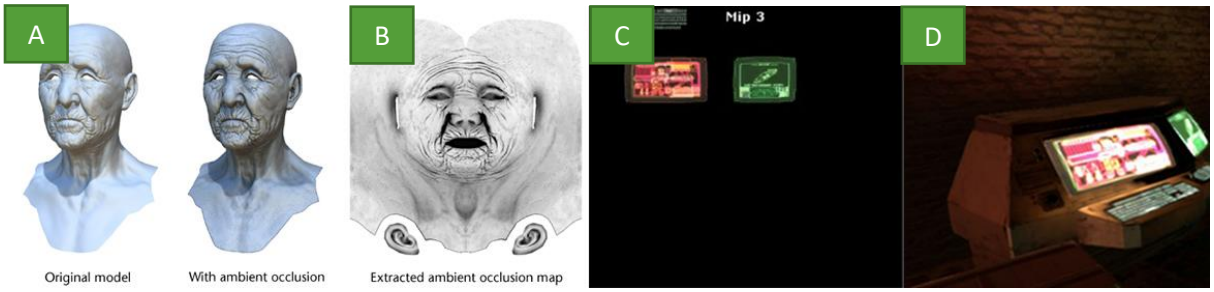


Figure 20 – (A) Model without and with occlusion maps; (B) Occlusion Map example [65]**; (C) Emissive map example; (D) Model with emissive map applied in the game engine** [66]**.**

2.3.6 CREATING TEXTURES

As shown previously, textures can be any image that give color or details to a 3D object in order to overcome the limitations of game engines and hardware which do not always have memory to process 3D models with a high number of polygons (check section 2.6 for more information on the 3D creation pipeline).

There are many ways to create textures. These range from creating them in Adobe Photoshop [67] to taking photographs, scanning them and even by using 3D applications like ZBrush [68] to model and render images. Although there are many ways to create textures for game assets, regardless of the way we choose to create them, image treatment will always be required. [69].

Adobe Photoshop is also very popular among game artists to create game art. Some use it to create concepts while others use it for detailed paintings of objects and characters. Due to it many tools and capabilities, Photoshop is considered a standard in digital art [69]. It is also very useful when creating seamless textures (a texture that can be tiled without any noticeable joining parts). Starting from a raw image from a photograph or online texture library it is possible to turn it into a pattern. Once the image editing software has been mastered, it is easier to achieve consistency among the textures created. As such, a program similar to Photoshop is vital to create great textures.

When creating textures from photos, imagination is key. Regardless of the original photograph and its context, it can be used to generate a texture in a completely different setting. For example, small cracks on one surface can become large cracks somewhere else, a pebble can be turned into a mountain. Even a small vent can become a large sewer grate [69]. To ensure this straight forward transformation of images it is vital that they have a large resolution and that they are captured without flash.

When it is not possible to photograph, scan or accurately draw the desired texture, it is possible to create the original object in 3D and make a texture out of it. This is an advanced method as it requires additional knowledge in 3D modeling and software. The first step in this method is creating a 3D model with incredible detail. Then, the model is painted, the shaders and materials are added and finally the lights are placed. When all of the elements come together, a high quality render is made and the image is applied to a low poly model to achieve the desired results [69], [70].

There are several approaches to creating good textures. The texture artist has to choose which method to use depending on the resources they have available and the quality they wish to achieve.

2.4 STANDARD UV PROJECTIONS VS UV MAPPING

There are two ways to apply textures to 3D models, standard UV projections and UV mapping. Standard projections are very useful for assets with simple geometry such as boxes, metal drums and other assets that lack organic shape. On the other hand, UV Mapping is mostly used for organic assets such as characters, animals or even trees but, it can also be applied to simpler models to provide a more accurate texture to the model [70]. This process consists in placing a 2D texture on a 3D mesh. The two-dimensional texture coordinates correspond with the vertex information of the geometry.

As the name implies, in projections, the textures are projected onto the surface of the 3D object. There are four different types of projections, planar, cylindrical, cubic and spherical. The planar projection is useful when recreating objects that we perceive as planar such as sheets of paper and posters but if the object is not completely flat, it will stretch the texture creating unrealistic results. Planar projections are also useful to mimic 3d objects using 2d planes. This type of projection is often used in video games to create grass and foliage that looks good but consumes very few resources (Figure 21 – A, B and C). Cylindrical, Cubic and Spherical projections are most suitable for objects with these shapes. Using any of these projections in other shapes will create distorted textures that will lead to undesirable and unpleasant results (Figure 21 - D) [70], [71].

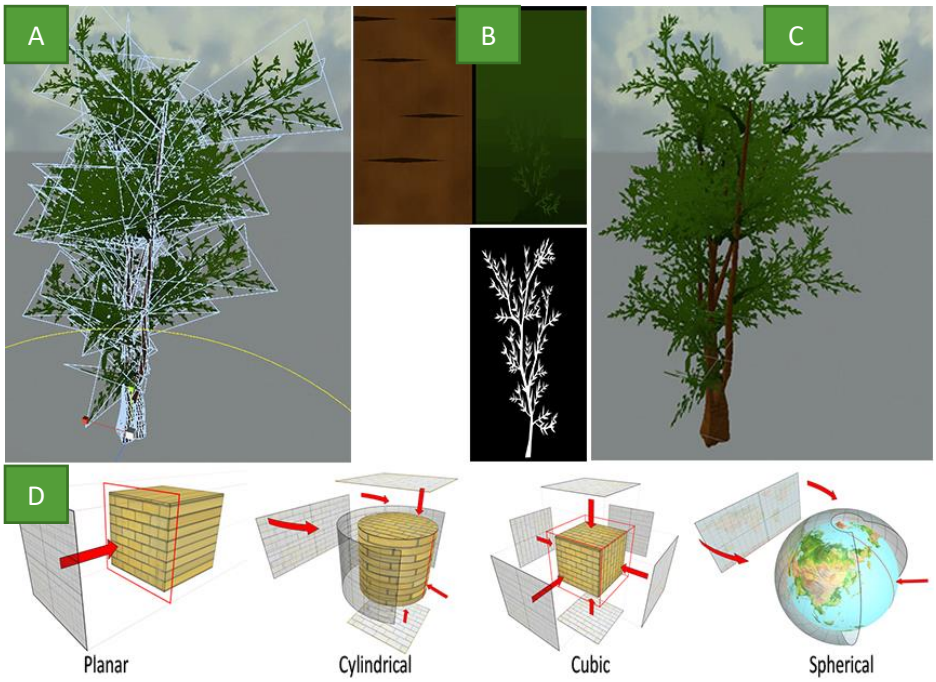


Figure 21 – (A) Tree wireframe on the game engine; (B) Color and alpha textures; (C) Final Tree on the game engine; (D) Example of each projection.

To overcome the limitations of UV projections, UV mapping is used. The first step to UV mapping is deciding whether to use the automatic UV layout created by the 3D application or a manual layout created by the artist (Figure 22 - A). The main limitation of automatic UV layout is its inability to accurately predict the best place to divide the 3D model. This often results in seams in undesired places such as the middle of the face instead of the desired back of the head, considered the standard [71]. Despite this, automatic UV layout is useful for models that resemble geometric solids. On manual UV mapping the artist divides the surface of the 3D model into several areas. The border of these

areas is called a seam and seams should be applied every time that there is a change in the material, the color of the texture or if there is a right (90°) angle. One very important consideration is that seams should always be placed in the least visible sections of the models regardless of how good they are. To improve the quality of the seams, padding should be used. Padding will allow the artist to blend the two sides of the seam in order to reduce its visibility [72]. A common practice when mapping the UVs is to overlap identical clusters to gain space in the map. This creates more space on the map for other objects allowing the overall map to be more detailed and saves the artist time since they only have to texture that cluster once (Figure 22 - B).

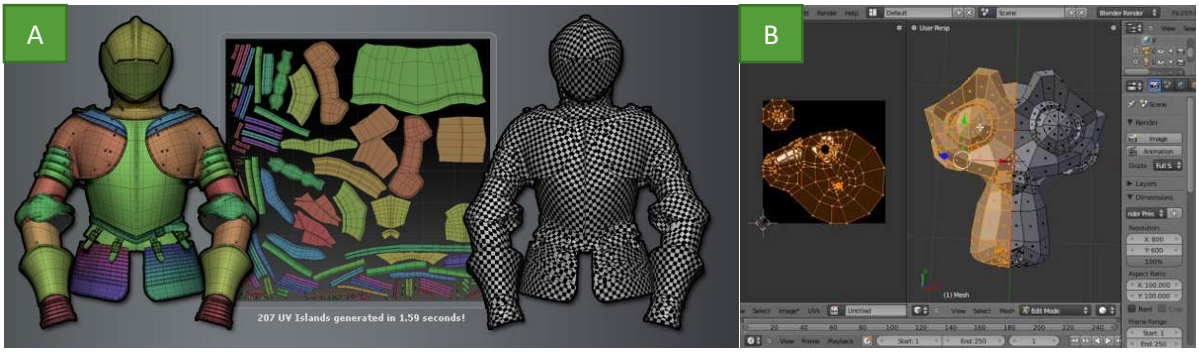


Figure 22 – (A) Model's unwrap map colored; (B) Similar clusters overlapped.

Keeping all of this in mind, texturing should only be applied to finished models as any change to the model will force the artist to repeat at least part of the texturing process.

2.5 ANIMATION APPROACHES IN GAMES

Animation brings 3D to life. Animation is typically approached in three different ways, key-frame animation, motion capture and procedural animation. In addition, these approaches can be combined to achieve better results.

In key-frame animation, animators decide how objects are animated in specific frames, the key-frames. This usually means that there will be at least two key-frames, the initial and final state. Additional key-frames are added to improve realism and to provide a smoother movement. These additional frames can be added manually or automatically by software. The main advantage of key-frame animation is that it can transcend reality which allows animators to produce animations that would be impossible to capture. On the other hand, this is still a very lengthy process despite major advancements in both technology and techniques [73].

Motion Capture is ideal to capture reality. It captures the movement of actors so that it can be mapped to virtual characters, making them move just like the actors [74]. Motion Capture can even record extremely detailed animations such as facial expressions. The main disadvantage of this type of animation is its cost, since it requires expensive equipment in addition to large spaces and actors. The main advantage of this type of animation is that it provides instantaneously results and with the advancements of technology there are some low-cost solutions such as the Microsoft Kinect [75]. Then again, these solutions lack in accuracy and require an animator to polish the animations [73]. Another disadvantage of motion capture is that actors are only part of the team for a short period of time, further increasing the cost of modifying or creating new animations. Despite common belief,

animators are still required when using motion capture in order to improve the animations themselves and to overcome the physical restrictions of motion capture [74].

Procedural generated animation is used to simulate the effects of the environment on an object. This new type of animation is used when it is not viable to animate every aspect of an object as it reacts to the virtual world such as cloth moving in the wind. This has proven to be extremely useful in video games since animators no longer have to animate a multitude of possible scenarios and it also generates slightly different results every time, reducing the possibility that the player (in games) will feel that the movement is unnatural or repetitive [73], [76].

In spite of the major advancements in animation technology, key-frame animation still remains as the most popular form of animation in the game industry [76]. While creating the animations for this game, all three types were used playing to each of their strengths. To keep players engaged and interested, procedural generated animations should be considered for unanimated objects and for any animations that are constantly repeated though out the game, such as the character's movement and execution of abilities. This can be achieved by using the game engine's physics on the character's clothes, hair and accessories.

2.6 GAME ASSET CREATION – STANDARD GAME INDUSTRY PIPELINE

In the game industry, there is a standard pipeline (Figure 23) to create a 3d model ready to be used in a game engine. The 3d model can be anything from a character to an environment asset. Although it is considered standard, different companies use different sets of software tools to achieve the same final product. This section is based on the information that game companies disclose and also in professional pipelines created by 3D game artists [77]–[80].

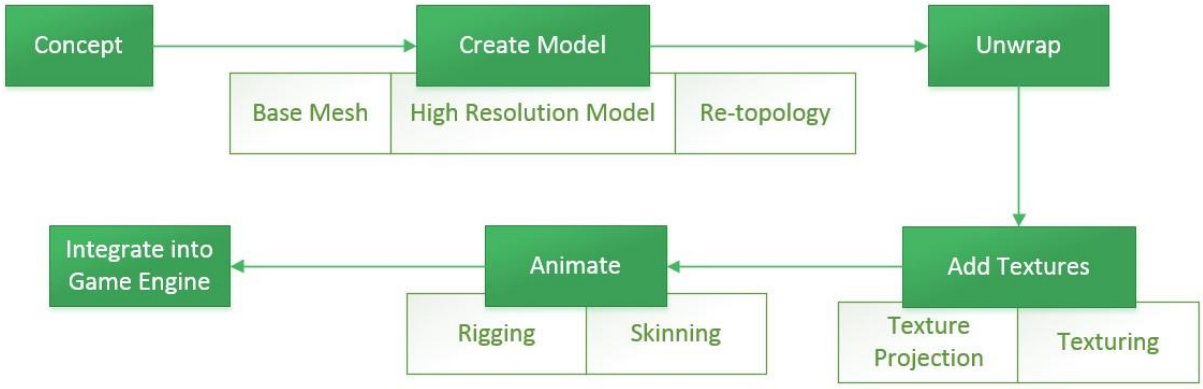


Figure 23 – Game asset creation standard pipeline.

2.6.1 CREATING THE CONCEPT

The first step in creating any 3d model is to have a concept which, is a series of sketches of the idea being developed. It is the most crucial phase because, if the concept is wrong then the final product will be wrong as well. In this phase, modelers, concept artists and creative directors work together to ensure that a shared vision is created quickly [77]. Concept design can be seen as a way of prototyping. Several concepts are created until the team is happy with them [78]. These concepts are usually made in paper and quickly converted to a digital medium. Here, Adobe Photoshop is one of the first choices

to begin coloring the concepts. It is also important to consider the type of surface the model will have, organic, hard surface or a mixture of both, as this will help to improve the quality of the concept [77]. When the concept is approved by the art director, it is sent to the 3D artist that will start to divide the concept into different parts while getting as many material and picture references as possible (Figure 24) [77], [78].

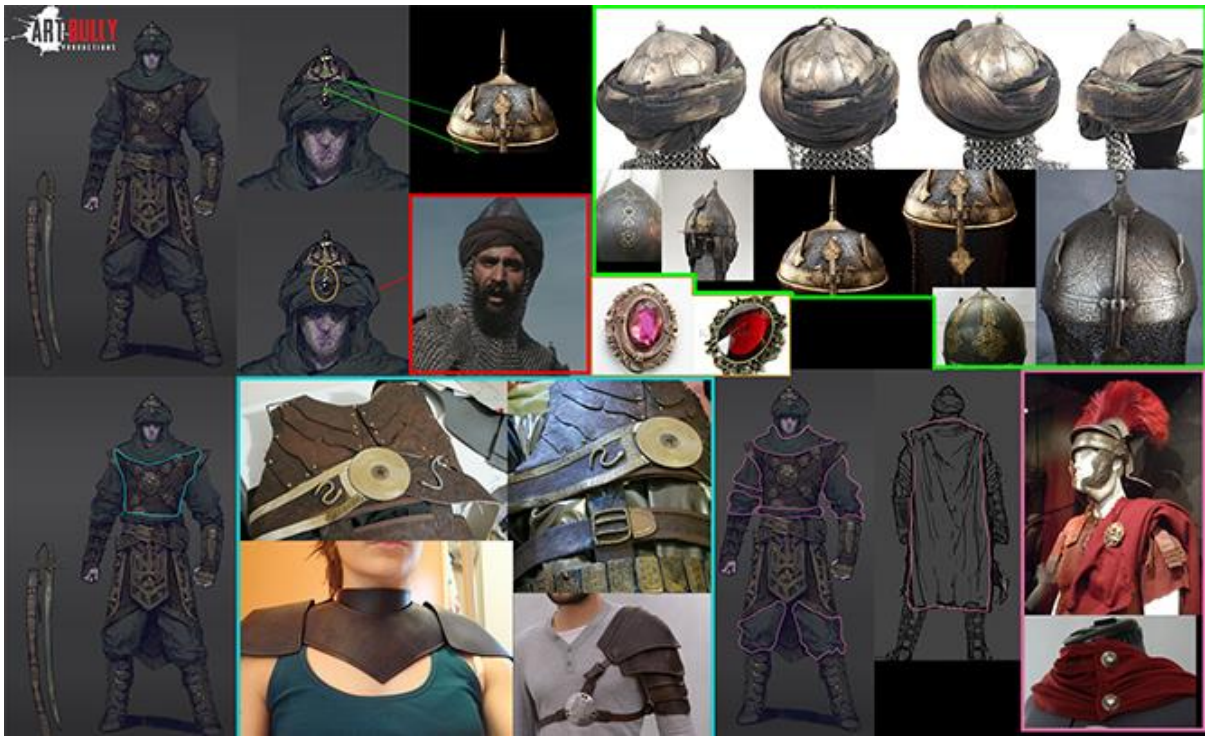


Figure 24 – Concept Art of a Persian Warrior from Art Bully Productions [81].

2.6.2 CREATING THE MODEL

The next step is creating the base mesh of the model. A mesh is a group of polygons and a polygon is any two-dimensional shape, the most common are triangles which are shapes formed by three vertices and quads which are shapes formed by four vertices. There are two ways to accomplish this, starting from a "primitive cube" or from a previously created base mesh (Figure 25 - A). This choice is up to the 3D artist. If the model is something similar to an already existing asset, the asset can be used as a starting point. If instead, the artist is starting from scratch, they will usually start from a primitive form, such as a cube or cylinder. At this stage, the artist uses Blender, 3dsMax or a similar program to create that base mesh. The most crucial rule during this step is ensuring that the base model is only made up of quad-based polygons (polygons with four vertices), no triangles or Ngons (a polygon with more than four vertices (Figure 25 - B)). This will ensure that in the next steps of the pipeline there will be no problems when subdividing the base mesh or rendering it [79], [82], [83]. Some companies already have vast libraries of base meshes that can be used as a starting point for new assets. This is convenient as it will significantly speed up this part of the process. For example, when creating characters, staring from a human base mesh will save valuable time and often resources that can be redirected to other areas of the development.

Once the base mesh is finalized, it should be exported as an OBJ file (a file that has all the data of a 3d geometry, such as position of each vertices and texture vertices) so it can be imported in Zbrush,

Mudbox or other sculpting program in order to create and provide detail to the high poly model. A sculpting software is any software that allows the user to sculpt on a 3D model as if they were sculpting clay in real life where the clay tools are the brushes inside the software. To produce the detail, the mesh will be subdivided several times, hence the need to make sure the mesh is only made of polygons with four-vertices. As each subdivision is made, more detail is added, as show in Figure 25 - C. As detail increases, so does polygon count, and several 3D applications cannot handle this many polygons like sculpting programs do. To prevent the 3D applications from running out of memory, the model is split in different parts to be imported later on 3DsMax, Maya or a similar program for the next step in the pipeline [77], [79], [84].

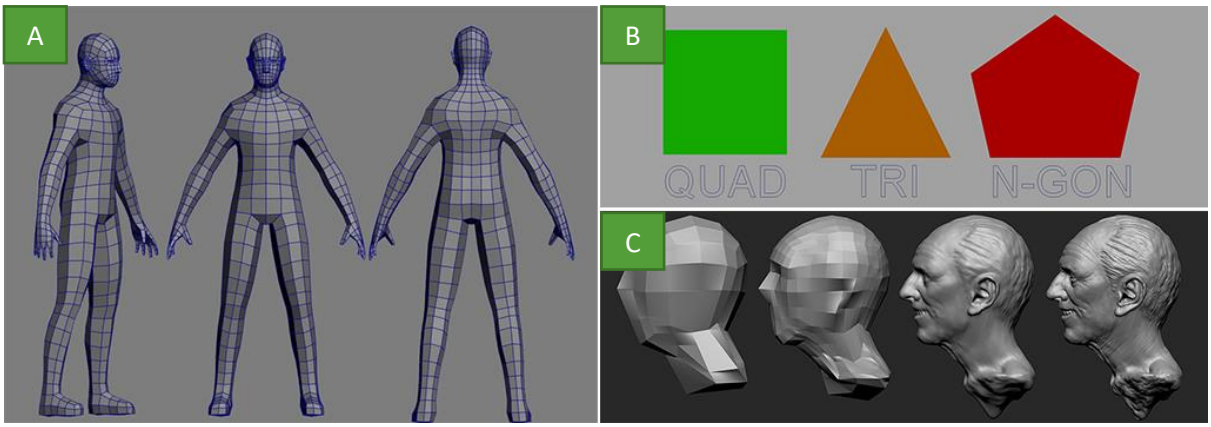


Figure 25 – (A) Base Mesh example; (B) Difference between quads, tris and n-gons [85]**; (C) Mesh subdivision phases in Zbrush** [86]**.**

Once all the detail has been added, the divided parts of the mesh are exported again as OBJ files, this time to be imported back to 3DsMax, Maya or a similar program.

This is the time to re-toplogise the model. Computers and game engines do not have the capability to render models with millions of polygons, nor can 3D artists afford to spend the extra weeks attempting to animate those kinds of models so, a more optimized mesh is required [77], [79]. Re-topology consists in creating the final low poly mesh that has the same silhouette as the high polygon mesh and that can then be used inside the game engine. There are different plugins and techniques to achieve the final product of this step. These range from creating a low polygon mesh on top of the high polygon model [79] or, using an automatically generated low polygon mesh from Polygon Crusher, a common plugin used in the game industry [77]. This final mesh will be used to animate and rig, so it is vital to consider how the topology (the way the polygons are connected to each other) is being formed. There are different guidelines that should be followed (Figure 26) to achieve good deformation and decrease the odds of having issues on rendering time [87]. A good understanding of the human body (when creating a human character) and how the bone structure affects its muscles is a plus when creating this mesh [79]. As previously mentioned in other steps, the models should not have any Ngons or triangles, since these will make the animating process far more complex.

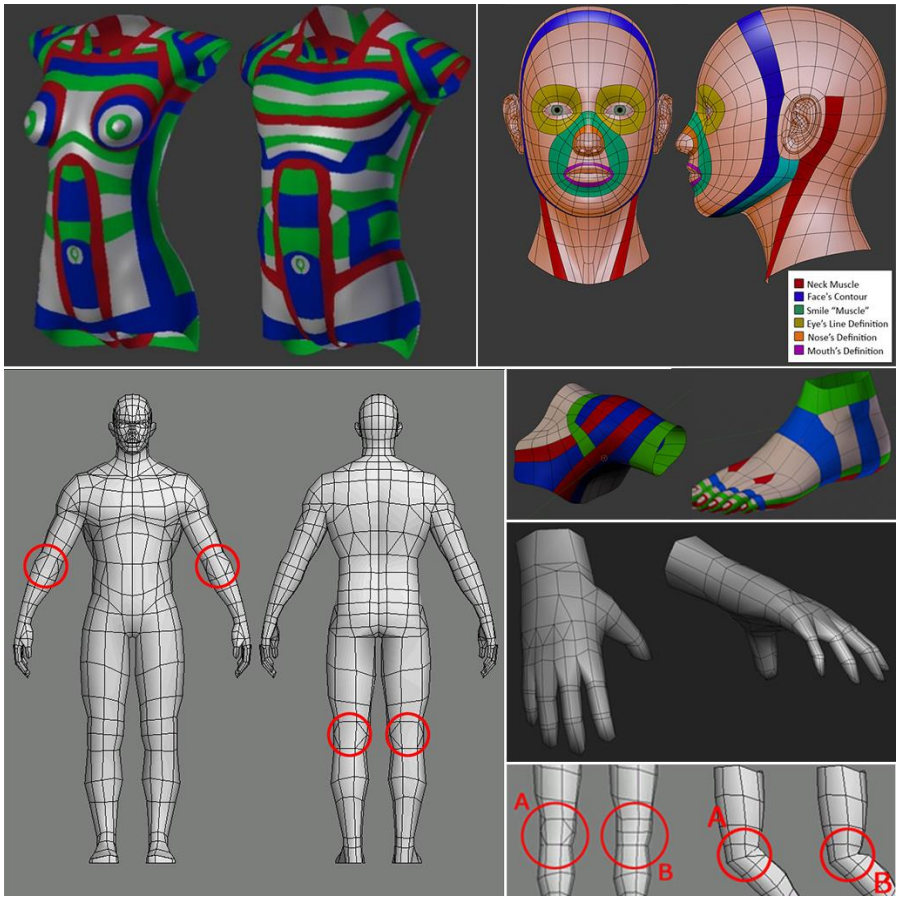


Figure 26 – Tips to get good topology while modeling characters [87]**.**

2.6.3 MAPPING THE UV'S

The next step of the pipeline is to unwrap the mesh. When creating a character for a movie, each section of the mesh can have a separate UVW. Unfortunately, game engines only allow a single mesh to have up to two UVW maps. As such, texture artists will have to consider which sections will receive more attention from players and give them more UVW space (more detail) at the cost of losing detail (UVW space) on other sections [77]. The unwrapping process consists of creating the UV maps for the mesh and making sure that all the polygons have similar sizes to be able to accurately texturize it in a future phase. As UV Mapping was already explained in detail in the Standard UV Projections vs UV mapping section it will not be approached any further in this section. Unwrapping can be accomplished by any modeling software, such as 3dsMax or Maya but also in a sculpting software like Zbrush.

2.6.4 ADDING TEXTURES

When the model has been unwrapped, it is ready to receive all the high polygon mesh's details onto it. To achieve this, it is important to ensure that the low resolution mesh has the same silhouette as the high resolution mesh in order to capture all of the details on the projection modifier [77], [79], [88]. Using a program such as 3dsMax or XNormals [89] it is possible to render the details of the high poly mesh to the low resolution mesh through textures, this process is also known as baking textures [78]. In other words, this process will create a Normal Map to save all the details, like stiches, wholes and cracks, an Occlusion Map to store all the information related to shadows and a Diffuse map for

the colors (Figure 27). Applying these three maps to the low poly model will completely change its detail without adding any more polygons, making it look like the high detailed mesh.

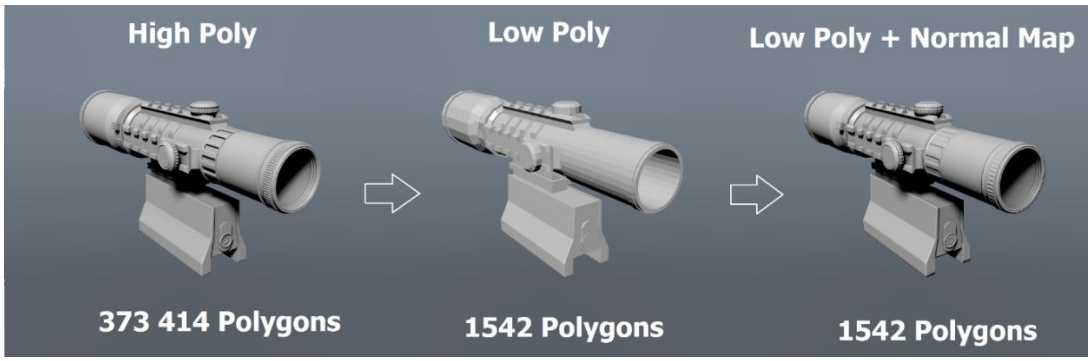


Figure 27 – How using Normal and Occlusion Maps can give a lot of detail on the final low polygon model.

At this point, the model can finally be textured. There are several routes at this point and this is where the pipeline diverges the most. Some companies prefer to use the common Photoshop workflow, where the baked textures are imported to Photoshop, the occlusion map is used as an overlay of the diffuse map and the detail is patiently added in different layers to the diffuse map [77]–[79]. Some companies prefer to texture the model in a 3D environment, like Zbrush, Mudbox, Quixel [90], Substance Painter 3D [91] or Deep Paint 3D [92] where it is possible to paint and see the resulting model in real time, so detail is actually painted on top of the model [77]. Other companies prefer to use a mix of both methods, they start by using Photoshop to correct any imperfections in the normal map, add textures to the diffuse map using the occlusion map as an overlay and even create new maps like specular and emissive. When they are happy with the base colors, they use the 3D model painting software to add additional color detail to it, making the model even more realistic [77], [78].

2.6.5 ANIMATING

When working with a static model, at this point it is ready to be exported as an FBX file (a file that has everything an OBJ file has and more, such as animations and skeleton data) and later imported into the game engine. However, when working with models that will be animated, such as a car or a character, there are still three steps to go. First, a skeleton must be created to be able to connect the mesh to an animation. Game companies usually have their own custom skeletons but some also use the defaults from software, such as the 3dsMax biped, CAT and Maya's HumanIK [77], [83]. These skeletons do not necessarily have the same number of bones as a human or animal skeleton. The standard humanoid skeleton has: a head bone, a neck bone, between two and three spine bones, a pelvis and for each side of the body there are also a hip bone, upper and lower leg bones, upper and lower arm bones. The bones of the hand are fused into a single bone as are the foot bones. On the hands the number of fingers can range from one to five and their number of articulations from one to three. Toes are generally represented by a single bone (Figure 28 - A) [93].

Rigging is the process of adding controls to each joint of the 3D skeleton, which can later be used by the animator to easily animate the model. There are 3D programs that can help character riggers achieve this faster, for example, in Maya, it is possible to create the bones with automatic controllers. When animating, if the controller is moved the bones follow. This makes the movement more realistic

and easy to execute. For example, raising a character's wrist involves lifting its shoulder, upper and lower arm and finally its wrist [83].

When the rig is created, with all the constraint limitations of how far it can stretch, how much it can bend and where it bends, it is time to tell each bone which part of the model will it affect. This process is known as skinning. For each bone, weights are assigned to the vertexes in the mesh. For example, in a character, to move an arm when the skeleton's arm moves it is necessary to assign the whole arm mesh to it. A higher strength needs to be assigned to ensure that the mesh completely follows the skeleton. On the joints, the weights must be evenly distributed between the two bones that create the joint so that when it is bent, it deforms properly (Figure 28 – B and C) [79], [83].

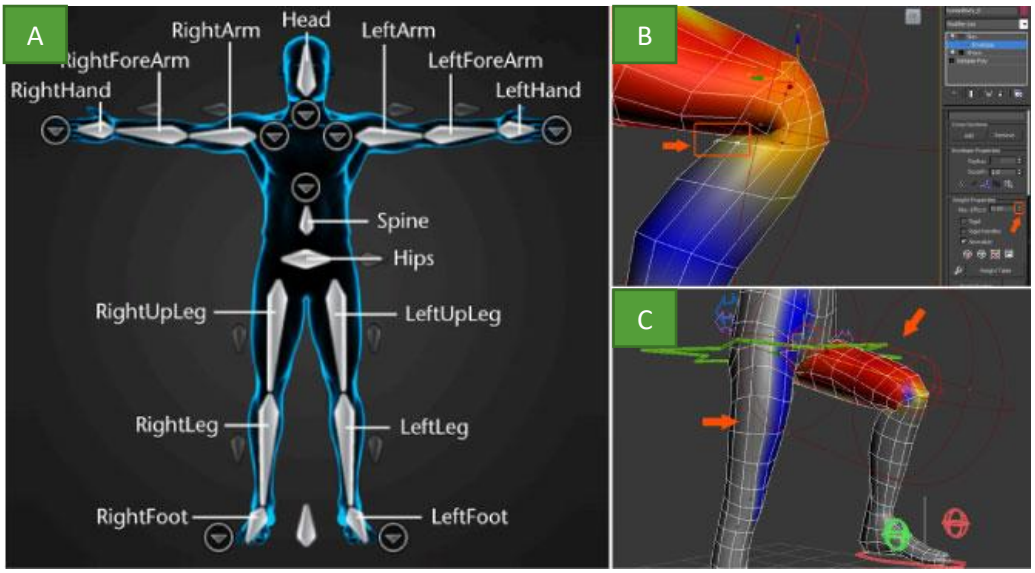


Figure 28 – (A) An example of a standard skeleton (downward arrows show where additional bones can be added); (B) and (C) Skinning Envelopes and how they should have their weights [94].

At this point, the model is ready for the final step, animation. Depending on the company's workflow, some will animate the model in a 3D program, some will animate it inside the game engine and others will use motion capture data on the character bones [79]. Animators usually create loopable animations to be able to use them inside the game without having to create long keyframed animations. For example, to be able to animate a character walking, all an animator needs to create is two steps of the character walking and then loop them for as long as needed inside the game engine [83]. When all the animations have been integrated into the model, it is ready to be imported into the game engine to undergo post-processing, such as lighting and animation transitions.

2.7 SOFTWARE TOOLS

Due to having to create all the game assets, research was required to better decide which software had the functionalities required to accomplish each part of the 3D pipeline process. The selected programs must be able to complete at least one of the tasks in the 3D asset production pipeline and be compatible with each other. Like any other tool, software's utility and performance is directly tied to the limitations of the GIP.

2.7.1 3D MODELING, RIGGING AND ANIMATING - 3DS MAX VS MAYA LT VS BLENDER

Currently, video game companies use different software in their game assets development pipeline. From modeling to sculpting, there are many tools from which they can choose from. Although there are many more tools available, the focus will only be in the three most popular for modeling, animating and rigging. They are 3ds Max [95], Maya [96] and Blender [97].

3ds Max and Maya belong to Autodesk and are the tool of choice for several companies. There is a version of Maya targeted for indie companies, Maya Lite (Maya LT) [98]. The big difference between them is that Maya can also be used for rendering high quality images, dynamics and physics but it also integrates python scripting while Maya LT is aimed at game development since it contains almost all the tools needed to produce game assets [99], [100]. Considering that Maya has all the functionalities of Maya LT and more, only Maya will be considered.

3Ds Max is known for its powerful modeling tools and vast modifiers library, that allow anyone to create complex 3D models with a fast workflow [101]. Furthermore, it has an easy to learn User Interface (UI), making it easier for inexperienced users (Figure 29 - A). Along with its UV editing tools, combination of unlimited number of textures and different projection methods, 3ds Max is a great tool when looking for an application specifically for asset creation. On the other hand, Maya is less forgiving when it comes to modeling complex assets and has a more complex UI (Figure 29 - B). Although Blender is not used often by AAA companies as the other software, it is used by many indie game developers since it is one of the only free tools available. Blender is known for its very intuitive and easy to understand modeling workflow [102], but it lacks in its UI (Figure 29 - C), being very unintuitive for a new user. The only downside of 3dsMax, contrary to Maya and Blender, is that it is a Windows-application only, without any cross-platform integration, making it impossible to use on Linux or OSX [101], [102].

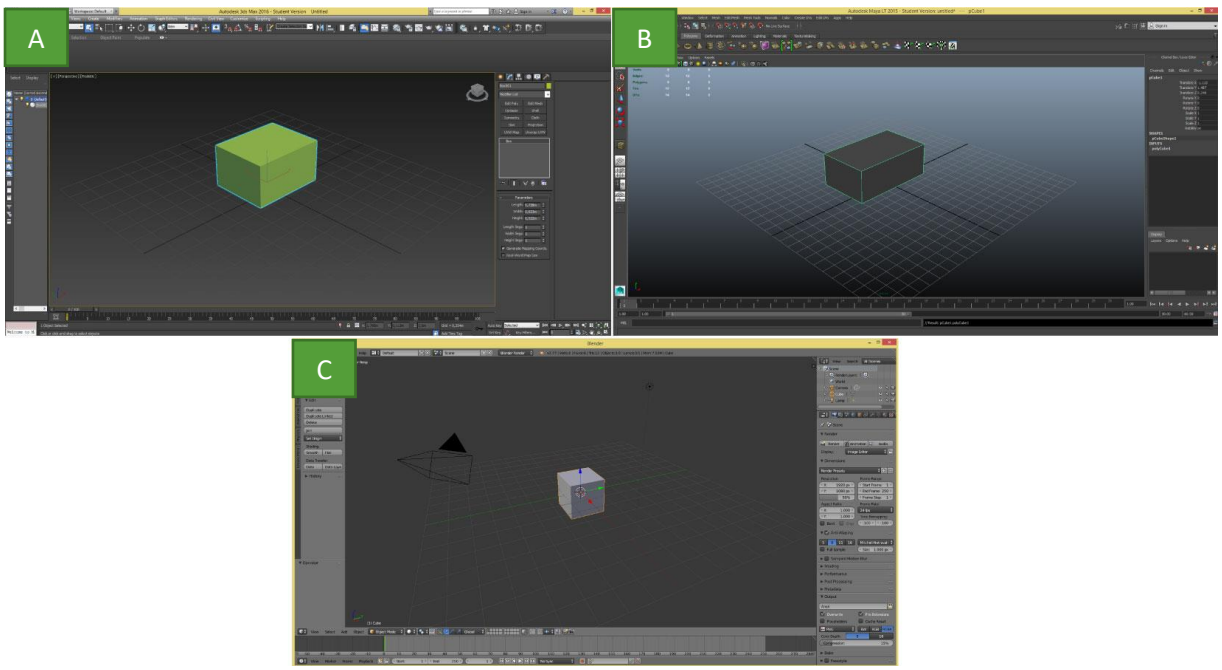


Figure 29 – (A) Autodesk 3DsMax 2016 interface; (B) Autodesk Maya Lite interface; (C) Blender interface.

When it comes to animating and rigging characters, 3dsMax has one of the best character animation tools available. The user can choose between Character Animation Tools (CAT), Biped and MAX customizable bones [103]. Between the three, Biped is the easiest to use, since it is an automated way of creating a bone system. CAT was created in an attempt to replace the biped, providing a bigger toolset and more animation controllers and finally, MAX bones, can be used when creating a custom rig to be fully animated by hand, which is not the case in the DAP [103]. Both systems (biped and CAT) allow the layering of animations and the use of motion capture data, which will both be used in this work [104]. While Biped is an old system, it is very good for two-legged characters but it falls short when trying to rig animals (even though it is possible to add additional bones to it). CAT excels both in animal and humanoid rigging and even includes a muscle system, adding some realism to the animations (Figure 30 - A). On the other hand, CAT is extremely unstable and will likely crash 3dsMax without any reason [104]. Maya has the HumanIK skeleton, similar to the Biped and CAT in 3DsMax, which creates a standard skeleton and lets the artist map all the bones so they can use the bone's controllers later. This feature along with Motion Builder allows the artist to use motion capture data later (Figure 30 - B). This is one of the big reasons why Maya is preferred during the rigging process and why so many game development companies use Maya. Unfortunately, Blender is the one that falls behind here, since the only possibility is to create a skeleton from scratch, the armature, which works as the MAX bones feature allowing the integration of motion capture later (Figure 30 - C) [101].

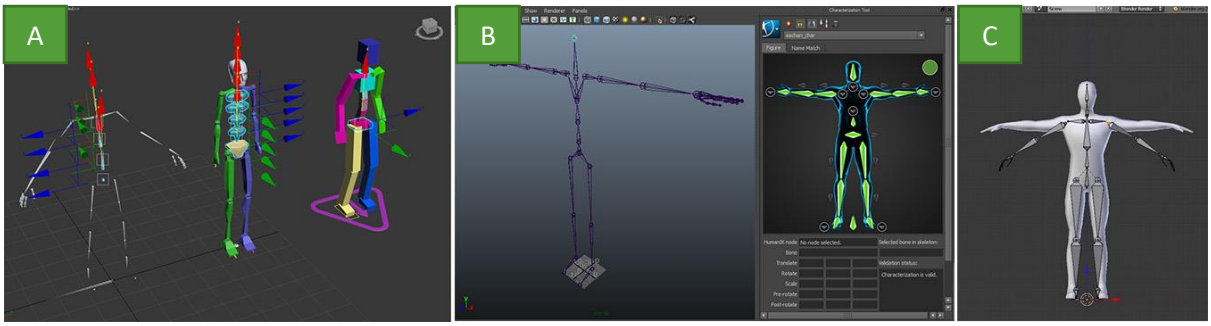


Figure 30 – (A) Maxbone, biped and CAT in 3DsMax; (B) HumanIK in Maya; (C) Armature in Blender.

As for skinning tools, 3dsMax has two modifiers, Physique and Skin. Physique is a very old modifier, created when the first biped plugin was created and has not been updated for several versions. The Skin modifier has surpassed Physique in how easy it is to use, the number of additional functionalities and in its reliability. Currently, nearly everyone skinning a character will use the Skin modifier in 3ds Max (Figure 31 - A) [103]. Maya and Blender have similar tools for the Skin modifier, named Smooth Bind (Figure 31 - B) and Skin modifier respectively (Figure 31 - C). In the end, the three tools work in the same way, the bones apply weights to the mesh [105].

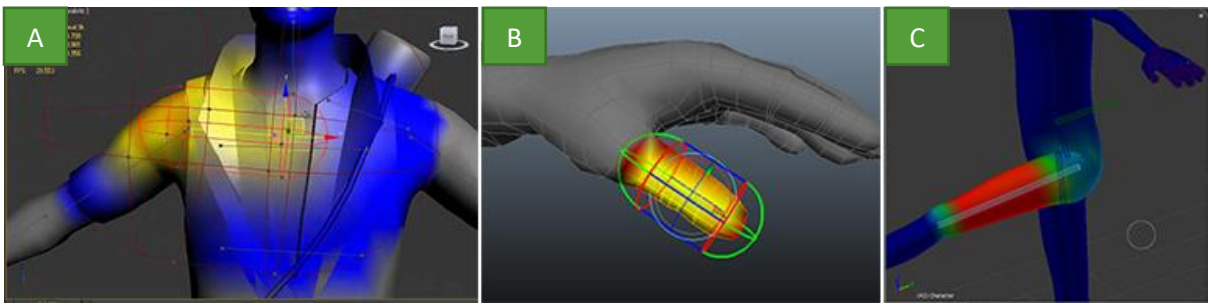


Figure 31 – (A) Skinning in 3DsMax; (B) Skinning in Maya; (C) Skinning in Blender.

Although 3ds Max is a good tool for animation and rigging, Maya is the tool of choice for those specific functionalities. Still, any animation can be accomplished on either software. Maya will still be more powerful when it comes to animation and rigging since it provides a superior amount of tools including MEL and Python scripting which will make these functionalities even more customizable [102]. Since 2014 it has been evolving and bringing with it great re-topology tools and a faster workflow [101]. Maya's strengths are in its scripting languages, rigging and animation tools, while its weaknesses lie in the lack of polygon work creation workflow due to its limited UI and repeated creation of N-gons.

Being a free software, the Blender community has grown and there is a considerable amount of information and support about this software on the internet. Blender is an all-in-one tool, it can be used to model, animate, sculpt, rig and even create a game because it has a rudimentary game engine on it [102]. However, since it has every single functionality, it lacks a bit on everything. Despite this, its modeling workflow outshines both 3ds Max and Maya. As for animating and rigging, Blender has a bit more depth than 3ds Max when working with keyframes.

Table 1 compares and ranks the three software packages through their characteristics and functionalities from 1 (very bad) to 5 (very good) considering the information gathered on them.

Table 1 – Comparison between the three most popular software for modeling, animating and rigging (1 - very bad; 5 - very good)

Software/Functionality	3ds Max 2016	Maya 2016	Blender 2.77
Ease of use	**5**	3	3
Modeling capabilities	4	3	**5**
Animation	4	**5**	3
Rigging	3	**5**	4
Skinning	4	**5**	4
UV Unwrap	4	4	**5**
Available Information	**5**	**5**	**5**
Cross-Platform	2	**5**	**5**
Price	**5**	**5**	**5**
Total	36	**40**	39

Considering that all of Autodesk's Software has a three-year Education free license and it would not be required to learn how to use 3ds Max due to one of the master's courses and since learning a software takes its time, 3ds Max 2016 was used for modeling, animating and rigging.

2.7.2 3D SCULPTING - ZBRUSH VS MUDBOX

3ds Max's modeling capabilities can only do so much. When planning to create a hard surface armor or objects, 3ds Max is a good choice, on the other hand, when creating detailed characters with details such as cloth stiches or skin imperfections, 3ds Max starts to fall apart. To accomplish this, it is better to use a Sculpting software that can handle working with millions of polygons at a time without running out of memory. The most popular are ZBrush from Pixologic [68] and Mudbox from Autodesk [106]. Both work on Windows and OSX.

Zbrush is a very powerful tool but it has a very complex interface (Figure 32). After getting used to it, it is easy to get lost in the vast amount of brushes available (both inside the program and created by the community), offering the look and feel of creating a real sculpture. The pen pressure (is a system that imitates a fountain pen, more pressure is translated into a wider and deeper line) works perfectly both in sculpting and painting and being able to create custom brushes adds a new layer of creativity that Mudbox does not allow. In Zbrush, when the topology of the mesh is modified or when it has to many polygons, by using Zbrush's ZRemesher feature it is possible to recalculate the topology of the mesh. This reduces the number of polygons and makes them all of the same size, allowing the mesh to have a perfect topology, as shown in Figure 33 [107]. This process vastly decreases the amount of time spent on fixing modeling issues.

Even though ZBrush does not belong to Autodesk, they have created a plugin, GoZ, that connects Autodesk programs to it, allowing the modeler to save a file inside ZBrush and refreshing that same file in 3ds Max, for example. The only down side of ZBrush is that there is no student or free version, there is only a 90-day trial version available.

One of the most important parts of the sculpting pipeline it to create a low definition model with optimized topology. This new topology will be used as the final mesh inside the game and this process is known as re-topology. Both ZBrush and Mudbox have methods that support this feature [107].

Since Mudbox belongs to Autodesk, it has a very similar interface to 3ds Max (Figure 32). There is no need for any plugin to integrate with the other Autodesk applications but unfortunately, it does not allow creating a base mesh from scratch. Another downside is the lack of support for different pen pressure tablets, as it only supports Wacom tablets [108].

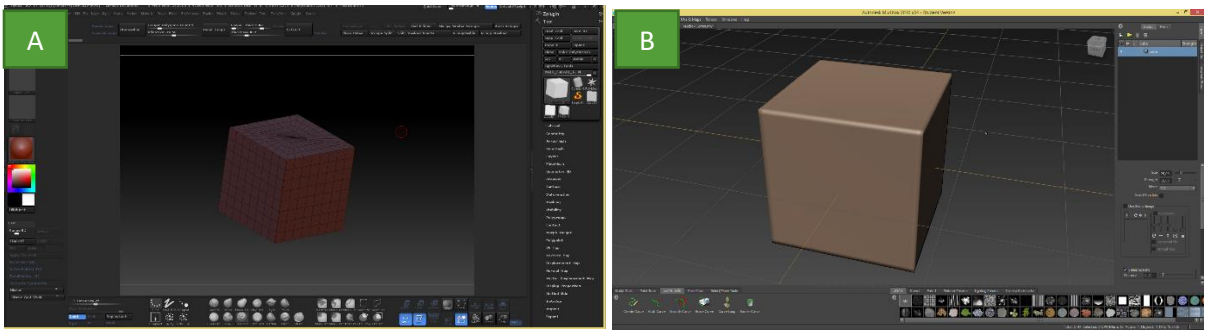


Figure 32 – (A) Zbrush 4R7 Interface; (B) Autodesk Mudbox 2016.

Mudbox really shines is in its texture painting features. It supports the creation of different layers for different texture maps where each map is assigned to a material and each one can have multiple layers blended together, like in Photoshop which can be seen in Figure 33. Since it is based on layers, it is possible to see the effects of maps like emissive and specular in real time. Another big advantage is being able to use textures that do not depend of the model resolution, like in ZBrush, where a low resolution texture will be resized to paint a high definition sculpt, distorting the texture image [109].

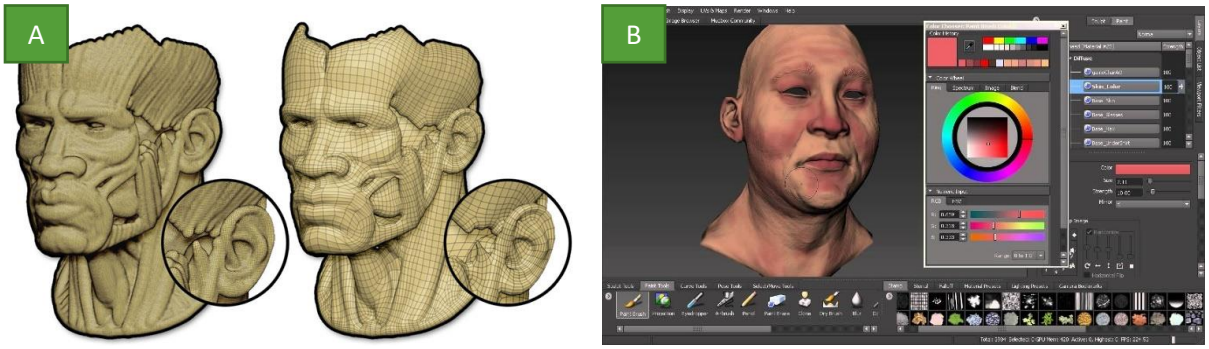


Figure 33 – (A) Zbrush Zremesher [110]**; (B) Mudbox texture layers** [111]**.**

In the game industry, nearly all companies use a mix of ZBrush and Mudbox to be able to create their high detail models. There are even some that only use ZBrush, since it allows everything Mudbox does except for the high quality texture maps [107].

Based on the information reviewed previously, Table 2 compares and ranks the two software packages through their characteristics and functionalities from 1 (very bad) to 5 (very good).

Table 2 - Comparison between the two most popular software for sculpting (1 - very bad; 5 - very good)

Software/Functionality	ZBrush 4R7	Mudbox 2016
Ease of use	2	4
Base Mesh Creation	**5**	N/A
Sculpting	**5**	3
Texture Painting	4	**5**
Re-topology	4	4
Peripherals Support	**5**	2
Available Information	**5**	4
Cross-Platform	**5**	**5**
Price	1	**5**
Total	**36**	32

Since there was no previous knowledge in either software and that the pen pressure features on Mudbox did not work with the drawing tablet being used, Zbrush4R7 was selected to create all the high detail models.

2.7.3 TEXTURING 3D OBJECTS – ZBRUSH VS PHOTOSHOP

Since Mudbox was discarded due to the fact that it did not support the drawing tablet used (see section above for more information), only ZBrush and Adobe Photoshop [67] will be compared regarding their painting features.

Models have many sides and the computer does not have the knowledge to correctly add a 2D texture to it, it is important to create a UV Map for each model (for further information check Texturing section) [112]. That is the reason for adding Photoshop to this comparison.

Photoshop is an image processing software. Among many other things it allows the user to create high detailed textures from scratch. The fact that it supports layers, considerably improves the workflow, as it allows the use of clipping masks that ensure that only the desired areas can be manipulated. One application of this is to create multi-colored models. This provides a quick and easy way to modify and/or update texture maps. This is also useful to create the same model with different color schemes which is very popular in videogames, to customize and add the player's personal touch to their model (Figure 34).

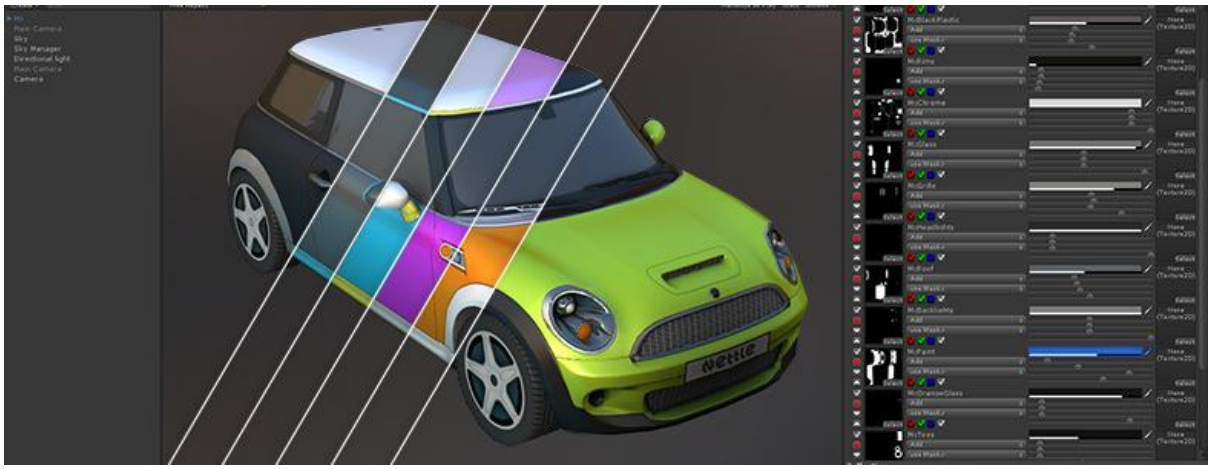


Figure 34 – Different color Maps for the same model.

On Zbrush it is possible to paint directly on the 3D model. This is also possible on Photoshop but it is much less intuitive and requires an additional software pack, Adobe Photoshop Essentials. This is very useful as the artists can see their end product in real time while they work on the textures. However, the lack of texture layers, considerably hinders the ability to maintain or modify textures once they are finalized. To perform this task, the artist is required to have additional third party programs. The lack of texture layers, also forces the artist to fragment their work into several files as they cannot create different texture maps in a single file (diffuse, specular, emissive, among others).

Considering the previous experience with Adobe Photoshop and that a student license account was available, Photoshop was selected for this part of the production pipeline. In Table 3 the two previous software packages are compared and ranked from 1 (very bad) to 5 (very good) considering the information gathered.

Table 3 - Comparison between the two most popular software packages for texturing (1 - very bad; 5 - very good)

Software/Functionality	ZBrush 4R7	Photoshop CS6
Ease of use	2	4
Use of Texture Layers	N/A	**5**
3D Model Visualization	**5**	3
Ease of modification	2	**5**
Price	1	**5**
Total	10	**22**

2.7.4 POPULAR GAME ENGINES – UNREAL ENGINE 4 VS CRYENGINE 3 VS UNITY3D 5

One of the most important things to consider while developing the DAP was, which game engine to use. The most famous are Unreal Engine 4 (UE4) [113], CryENGINE V (CE5) [114] and Unity3D 5 (Unity) [3]. At the end of this section, Table 4 compiles and compares the features of each game engine and awards each one a score according to the needs of the DAP.

Since none of these game engines had been previously used in other courses or projects, ease of use was one of the main selection criteria. In this aspect, Unity shines the brightest with its easy to use interface, well documented API and many official courses and tutorials available for free. Unity also has an easy to use drag and drop system to create the game interface [115]. Despite the usability improvements that came with UE 4, it still falls behind when compared to Unity [116], [117]. CEV is an advanced game engine recommended for users that are already experienced with other game engines. It has a very steep learning curve making it an unreasonable choice as a first game engine [115]. While all of these game engines support the same features, one of the main differences is that while in UE 4 and CEV the user has instant access to all the properties of an asset, in Unity the asset only has the necessary properties to use it as is and all the other properties can be added as is shown in Figure 35 [117]. When compared to Unity, UE 4 and CEV also fall behind on their documentation.

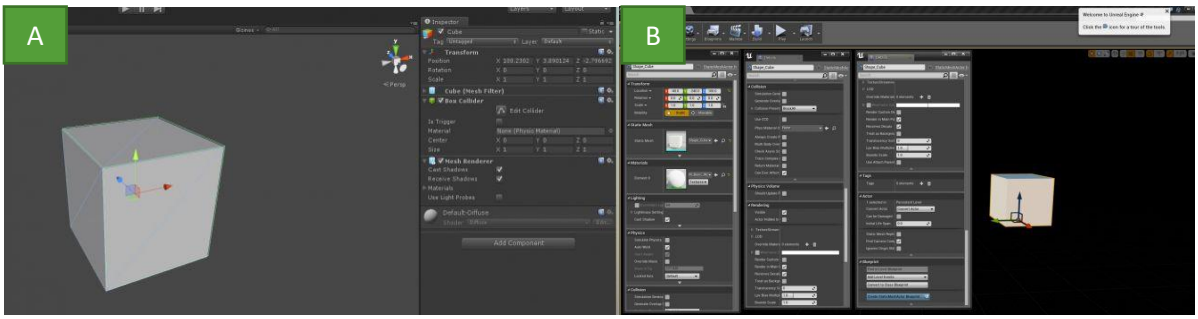


Figure 35 – (A) Unity3D cube properties; (B) UE4 cube properties [117].

Until the release of Unity 5, graphic fidelity was one of the main strengths of the other two engines. Although Unity is still behind, its current version can compete head on with the other two engines. To demonstrate how far Unity 5 has come, the Unity team built a short video entitled 'Adam' where they showcase the power of Unity 5 [118]. In Figure 36 it is possible to see the quality that each game engine can achieve. Considering how powerful UE4 and CE5 are, it is no surprise that their editor also requires far more resources than Unity [119]. While Unity is the current standard for 2D and small 3D games such as mobile and indie games, UE4 and CE5 are still the standard for large open worlds and triple A quality games [115], [120].

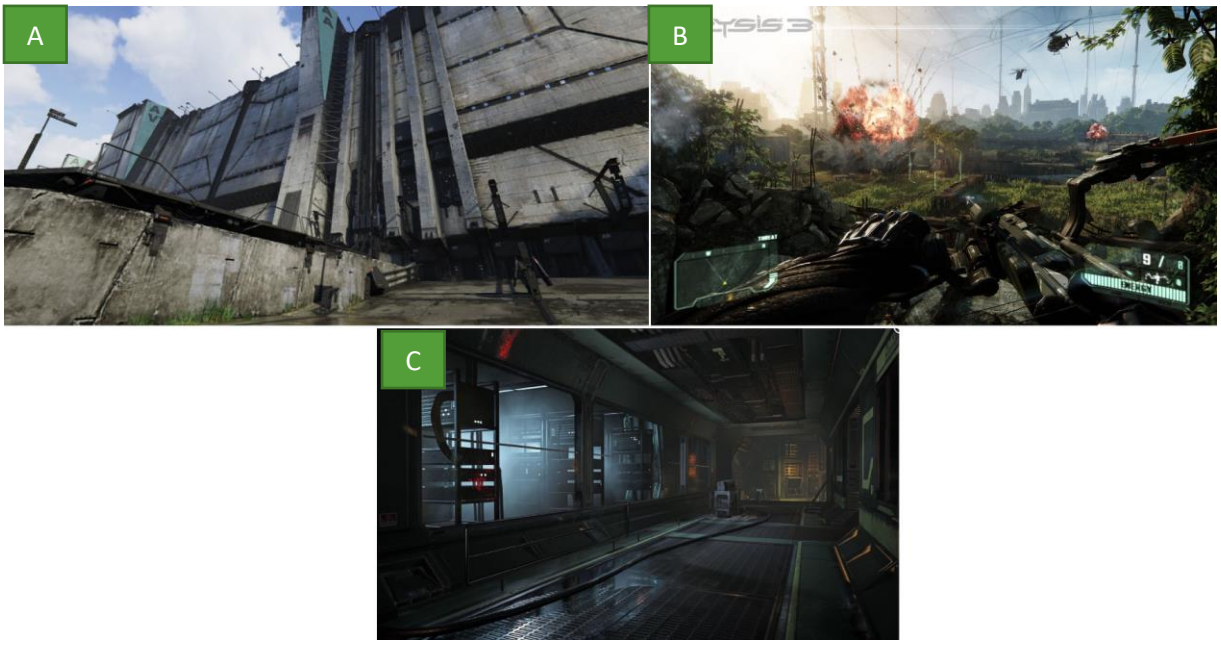


Figure 36 – (A) Adam made in Unity3D 5 [118]**; (B) Crysis3 made on CE 5** [121]**; (C) Doom 2016 made on UE4** [122]**.**

All three game engines have an asset store. This is very useful as the assets in these stores can be used to significantly reduce the production time of video games. Although all of these online market places have an extensive catalog, the Unity Asset store is the biggest and usually the most affordable [119].

Each of these game engines has employed a different monetization strategy. UE4 is free and developers have no costs unless they earn more than 3000USD (approximately €2677.85) each quarter per game at which point, they have to pay a 5% royalty to Epic Games on all of their sources of revenue [123]. CE5 was a new milestone for Crytek (developer of CE5) as it was released in a 'pay what you want' plan [124]. This allows everyone to access CE5 without having to invest into a software license they might not use and the fact that it has no royalties makes it more appealing than UE4. Finally, Unity3D changed their monetization strategy but not their plan. The full game engine is still available for free for yearly revenues of up to 100 000USD (93 622.4394 €) while additional functionalities and services can be purchased using a paid plan. Like CE5, Unity does not charge royalties.

When choosing which game engine to use, at the start of the GIP, Unity was instantly favored since at that time it was the only game engine which had no costs. Another motivation to choose Unity was the release of Hearthstone: Heroes of Warcraft by Blizzard Entertainment [52] which was developed in Unity and shared many mechanics and systems with the game developed in the GIP. Finally, Unity natively supports several 3D file formats instead of being restricted to FBX, which greatly improves the workflow [115].

Table 4 - Comparison between the three mentioned game engines (1 - very bad; 5 - very good)

Software/Functionality	Unreal Engine 4	CryENGINE 5	Unity 5
Ease of use	4	3	**5**
3D Programs Integration	4	4	**5**
Graphics Quality	**5**	**5**	4
System Requirements	3	3	**5**
Asset Store	3	2	**5**
Documentation	3	2	**5**
Price	4	**5**	**5**
Total	26	24	**34**

With all the information that was reviewed so far, it is possible to understand the importance of narrative and aesthetics in games, how games display the information on their interface and what types of prototyping techniques should be used. The game asset creation pipeline for the game industry, which was reviewed, will be used throughout the DAP. 3DsMax 2016 will be used for modeling, animating and rigging, while Zbrush 4R7 will be used for digital sculpting, and Photoshop CS6 for texturizing. Unity3D will be the game engine used to develop the digital game.

3 THE BOARD GAME

As mentioned previously, the board game was used as the foundation of the digital game being developed. The board game is played between two teams, with one to three players. Each team controls three heroes and their goal is to destroy the enemy towers. Lastly, each team has a pool of resources shared between all their heroes.

The game takes place in a hexagonal map which does not cover all of the available area. This map is symmetric and has four towers, two for each team (Figure 37 – A). The map's tiles are not all the same. There are tower area tiles, normal tiles and spawning tiles (Figure 37 – B). Each tower is represented by an object that has information about its health, damage and the team to which it belongs.

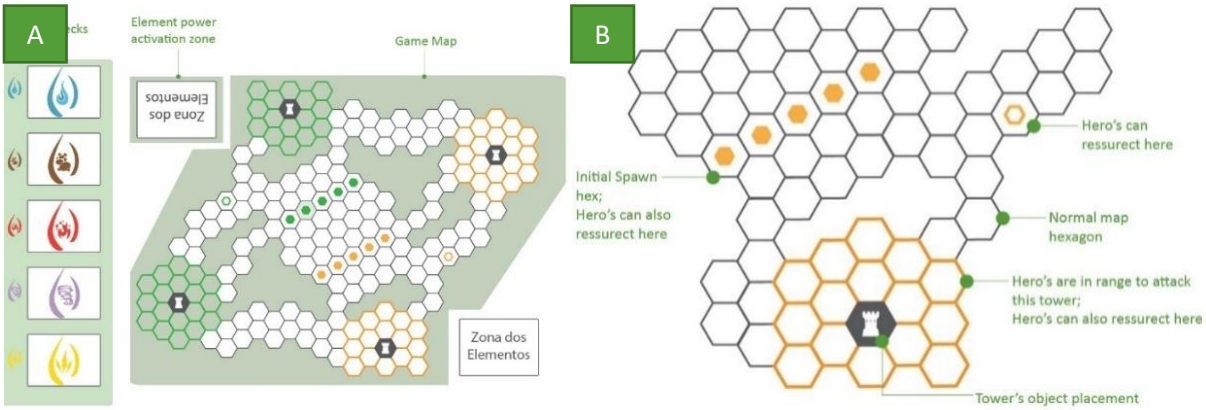


Figure 37 – (A) Board game's map; (B) Different hexagon tiles in the map.

Each hero is represented by a token on the map. All the information related to it is contained in its hero sheet. The hero sheet has the hero's name, health, tower damage, a portrait, a token that shows if the hero has already played, the types of abilities that the hero can unlock (polarities), which unlocks award resources and which abilities are unlocked, have been used and can still be used. Each hero can have a maximum of four abilities unlocked at any time, although each hero can choose from a total of eight abilities. The hero sheet also contains the element that empowers the hero and which element is currently affecting the hero (Figure 38- A). Hero abilities are divided into three polarities, damage, utility and ultra. They also have a title, a description, a cost, a minimum and maximum range and, if they are a cycle, a counter or a normal ability (Figure 38 - C).

Heroes' elements grant access to one of the five different elemental power decks, each with a corresponding theme and purpose. Each card has a title, a description, to which element it belongs, a rarity and when it can be used, which affects if the card can be kept or not (Figure 38 - D). Each team has an area where they can activate elemental powers, which will affect the map and the objects on it. This can only be done at the beginning of each round. Every time a hero uses an ability, he performs an elemental combination. At the end of each turn (after a hero has finished playing) elemental combinations are calculated and these can be positive, negative or neutral, depending on the elements combined (Figure 38 - B). Positive combinations give the team an element power card, negative combinations give the enemy team an element power card and neutral combinations award each team a card.

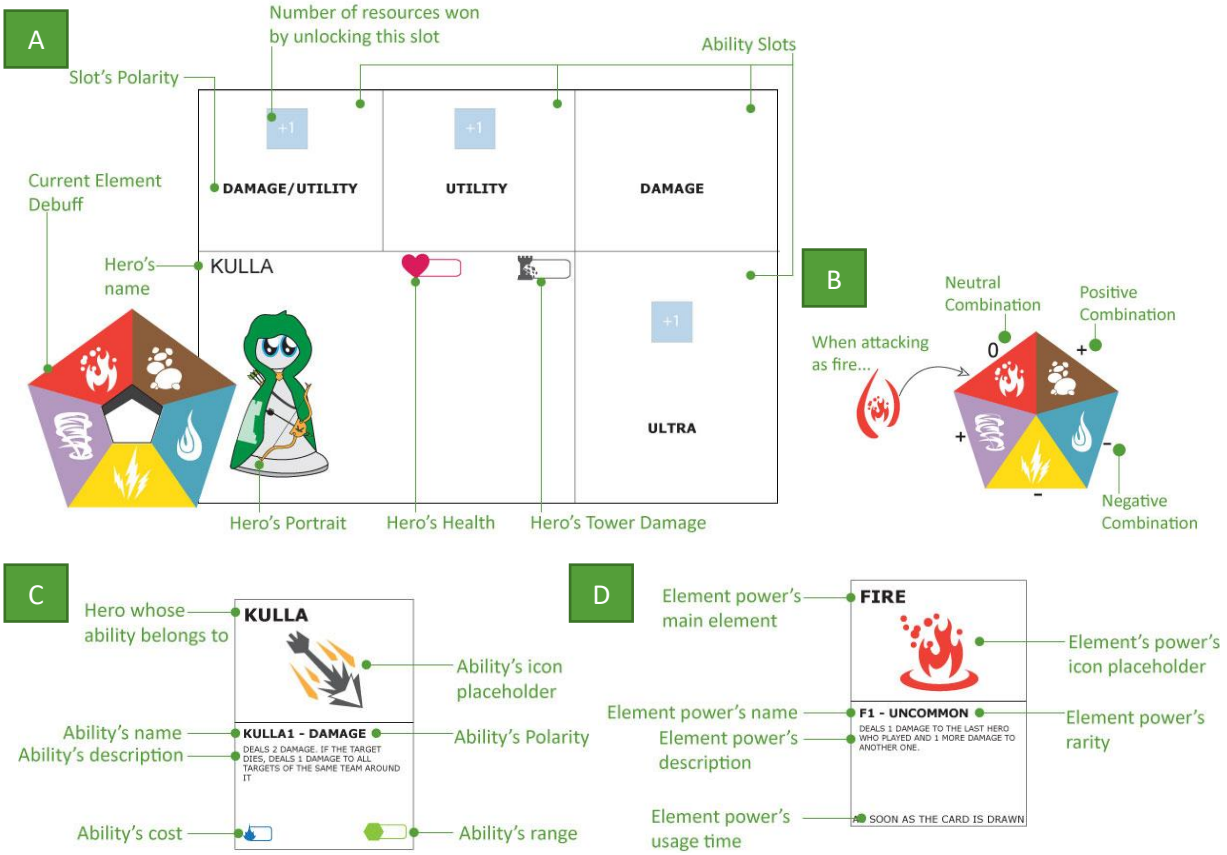


Figure 38 – (A) Hero's board sheet; (B) Element combination system; (C) Ability Card explained; (D) Element power card explained.

The game takes place over several rounds. A round is completed when all the heroes from both teams have played. The actions of a hero can be movement, to use abilities and/or to attack towers. An ability is unlocked each time an enemy hero or tower dies. When heroes die, they can resurrect on their next turn. Resurrecting uses all of their actions in the current turn, ending it.

After setting up the board, each team chooses the starting ability for their heroes and which element will empower them. To start the game, the teams flip a coin to choose who goes first. After the coin toss each team will place its heroes alternatively on the spawning hexes. Once all heroes have been placed on the map, the team that won the coin toss will begin playing. The order in which heroes play is not defined and can change between rounds. To remove the first turn advantage, the last team to play on one round will be the first team to play on the following round.

For further information on how to play the board game see its rulebook [125]. The main difference between the game implemented and the board game is that each a player controls a team of three heroes instead of having each player control a single hero.

PART II:
GAME DEVELOPMENT

4 GAME STORY DEVELOPMENT

In this section the creation of the game's background story will be explained along with the story of all the characters and clans.

There was no background story on the board game. There were five different classes: Assassins, Pathfinders, Wizards, Menders and Guardians. Each class had one or two heroes and each hero had been given a name and a region where they came from in the real world. In games, it is important to have stories (reasons) for everything. Why a certain object exists on the map is just as important as a character's background to make the player feel engaged and keep them interested.

A background story for the game had to be created, each hero, class and clan also had to be defined along with their aesthetics. The board game's map was only a set of hexes without any images or background of how the actual playing field was and this also had to be defined.

The first step was defining the game's universe, time period and the world regions from where each clan and consequently each hero came from. This was followed by defining the world where the game took place, what existed there and why.

4.1 GAME'S UNIVERSE AND STORY

A brainstorm was made in order to define why and where the battle would take place. Then, a story was created using the hero's journey [126] as a foundation, from the call to action to the completion of the adventure.

There are two main galaxies, Red and Blue. From time to time (around 50 000 years) the galaxies' tails touch each other and it is possible, during a small window of time, to travel between the two galaxies. The people from the Red galaxy are technologically superior. The inhabitants of the Blue galaxy have the same level of knowledge that existed in the real world between the 10^{th} and 16^{th} centuries and have no idea that the Red galaxy exists. When the galaxies' tails touch, the people from the Red galaxy travel to the Blue galaxy and attack its planets to harvest their resources, which they need to maintain their life. When they attack the planets, they kill nearly everyone and leave them in ruins. This is a cycle that happens every 50 000 years. On one of those cycles, Lenam, was the scout on the mission from Red galaxy and he was the first to arrive to Theris. He had some problems landing and crashed his ship. A woman comes to his aid and eventually they fall in love. Lenam is now faced with a terrible choice, he either does what he was ordered to do or he protects his love. Lenam chose to protect his love and her world using the last of his ship's energy and barely manages to hide Theris just in time as the ships from the Red galaxy attack. The people of the Red galaxy leave a trail of devastation in their wake. Every planet except for Theris is devastated and in ruins. Lenam decides that he does not want future generations to be in harm's way and decides to take a stand. He starts by travelling through Theris to find each of its clans and to recruit their best warriors. Using the last of his magic, he opens portals to other planets and sends the chosen warriors of each class to train there. On these planets the warriors can hone and master their skills so that on the next cycle they are able to defend against the people of the Red galaxy.

The most relevant aspects for the digital game's story is where the heroes come from, where and why they fight each other. The most important planets on the Blue galaxy are Theris and Intheris, where

the game takes place. Theris is the planet where all the heroes were born and where their clans live. Intheris is where the game's fight happens since it is the first planet to be harvested on every attack. The world of Intheris will be explained in further detail on the next section.

4.2 GAME WORLD - WHERE THE GAME TAKES PLACE

The board game had already defined the map's hexagonal grid layout, so the world around it had to be built. To understand why the game's world has its assets, it is important to know how Intheris is and what events shaped it over the years. This planet has countless trees and green plains. Many battles were fought throughout the years and countless civilizations have come and gone. Like the people of Red Galaxy, there was once a time when people of the Blue galaxy could control all of nature's elements at the same time. As time went by, this knowledge and power were lost. The people of the Blue galaxy have lost their connection to the elements and can only ask for the help from one element at a time.

Regarding the requirements for the map, in addition to the hexagonal grid layout, the only other requirement was to create a place where the heroes could go when they are defeated in battle since they are only training. Figure 39 provides a detailed explanation of every important monument (3D asset) on the final game's map.

So far only the game world has been described but the heroes and their clans have also been mentioned. The following sections will go into further detail on how each part of their story and concepts were created.

Figure 39 – Game World Assets explained.

4.3 CHARACTER CLASSES

As previously mentioned, there are five character classes on the board game. The assassin class which can only attack at melee range but are normally know in other games for their ability to manipulate and deceive. The Pathfinders are very good at incapacitating their enemies, only have ranged attacks

and are very limited in melee range. Wizards on the other hand control magic and are very powerful but also very fragile. Menders also use magic but they use it to manipulate life. While some heal their allies, others specialize in stealing health from their enemies. Guardians are on the front lines. With their enormous shields, they are there to protect their allies and take all the incoming group damage.

Defining the classes alone was not enough for the digital game. These required a background story and a design concept. For each class, a world concept was developed and their colors were defined. The inspirations that should be used when creating new characters of a certain class were also set. Each class was assigned a weapon style, so that inside the game it would be possible to easily distinguish between the five classes (Table 5). The weapon style was also based on how the world where they trained was and how each class is already seen in other games. For example, a wizard is known for having a staff or a one-handed sword, but never a shield. The assassin class, to which Malik and Wang both belong, will be explained in detail while further information on the remaining classes can be seen in annex 9.3.1.

The world where the assassins travelled to improve their skills is filled with ruins which have been taken over by nature. There is a lot of green from large trees and vines covering the stone ruins. It is a lonely world and its biggest threat is how hard it is to get around on it. This world was envisioned using concepts from the Aztec ruins, Indiana Jones and the Tomb Rider worlds but also the levels of Assassins Creed 2 that unlock Altair's armor (Figure 40). For the assassins, small but sharp and deadly weapons were chosen, such as daggers or chains with sharp edges. These needed to be small since assassins are going to be jumping around walls and using vines to go from one place to another in order to solve the world's tridimensional puzzles. As such, they could not be carrying heavy or big weapons (Figure 40 - E).

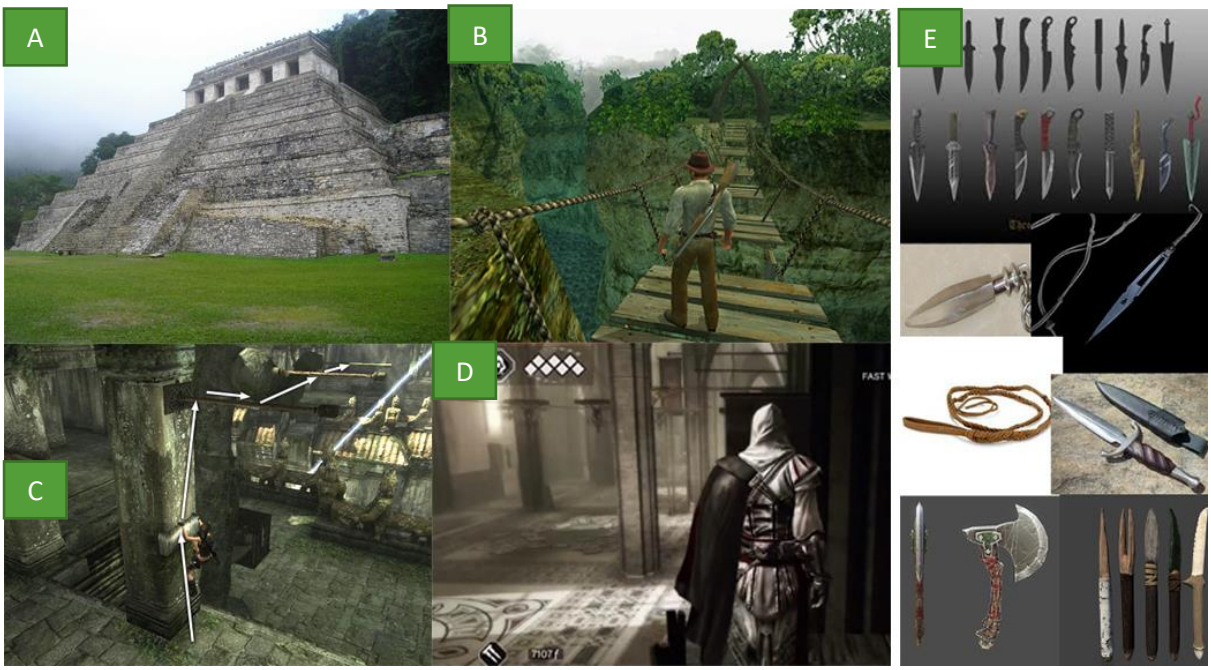


Figure 40 – Assassins world and weapon inspiration: (A) Aztec ruins; (B) Indiana Jones and the Emperor's Tomb ; (C) Tom Raider: Underworld [127]**; (D) Assassin's Creed 2: Altair's armor unlocking level** [37]**; (E) Assassins weapon inspirations.**

Table 5 – Game Classes Process

Class	Weapon Style	World inspirations	World colors	Concept Inspiration
Assassin	Small bladed weapons	Aztec and stone ruins, nature growing inside buildings	Brown, green and stone grey	Assassin's creed2 [37],Indiana Jones, Tomb raider[127]
Pathfinder	Bow, crossbow, gun	Steampunk, victorian era technology	Rust brown and metallic grey	Bioshock [128], dishonored 2 [129], Steampunk game/worlds
Wizard	Staves	Floating towers/buildings, magic, pointy buildings, small houses, big towers	Purple, blue, beige	Dalaran (WoW) [43], sky cities, elf cities, crystal cities
Mender	Alchemy flasks (different colors for different types of menders (death/life)	Life: round edges and lively forest Death: dark forest with spiked edges	Life: Green, strong colors; Death: Dark, grey, black, dark purple	Twisting nether legion (WoW) [43], Forbidden forest (harry potter), Mirkwood forest (LotR), maleficent forest (both life and death)
Guardian	Big Shields	Mountains with some valleys, snow on top of the mountains, Castle on top of mountains, desert and barren ground	Brown, green, stone, dark grey, white	Jurassic Park, Minas Tirith (LotR)

4.4 GAME CLANS AND ITS AESTHETICS

As mentioned before, the only thing that had been defined on the board game was each of the character's names and which region they were from. Each clan's name, ethnicity, beliefs and spoken language also had to be defined along with, their colors and aesthetics (Table 6).

Table 6 – Board game's already existent background

Character's name	Class	Region	Spoken Language
Malik	Assassin	Arabia	Arabic
Ghora	Guardian	India	Hindu
Lipp	Wizard	Estonia	Old Norse
Kulla	Pathfinder	Albania	Albanian
Wang	Assassin	China	Chinese
Loper	Mender	Africa	Africans/Zulu

4.4.1 CLAN CREATION PROCESS

For this part, information from the board game (character's name and region) was used. This limited some decisions, and research was performed on each country/region in order to find what existed there at the time where the game takes place. A table was created (Table 7) with the ethnicities, traits, beliefs and most commonly known values of each clan. Each clan was given a name according to the last column of this table, 'Traits, beliefs and mostly know values'. This process was done using Google Translate and some other dictionaries like Old Norse Dictionary [130] to translate these words to the spoken language previously defined for each clan.

Table 8 shows this process for Malik's clan, the other clans can be seen on annex 0 - Table 26.

Table 7 – Clan definition process end product

Clan's Character	Ethnicity	Traits, beliefs and mostly known values
Malik	Arab people	Religious, lonely, generous, honored
Ghora	Indian people (Sikh)	Equality, sacrifice, humility, generous, honest
Lipp	Vikings	Brave, courageous, strong, ancestors
Kulla	Albanian people	Free people, heroic, resilient, martyrs
Wang	Chinese Monks	Discipline, many people, strategists, technologically advanced, virtuous
Loper	Zulu	Value life, ancestors, animal force, animal spirit, community

Table 8 – Possible names for Malik's clan –final choice in bold

Clan's Character	Traits, beliefs and mostly known values	Translated Traits, beliefs and mostly known values
Malik	Religious	Almutadiinin
	Lonely	Wahidaná (reads Whahoieda), muta Wahid
	Generous	Sakhi, **Karim**
	Honored	Sharaf, fakharr

4.4.2 CLAN AESTHETICS

The aesthetics for each clan were also created. These were defined using colors based on their region, known traditions and other games that had characters that shared the same story or inspiration, such as the Zulu tribes from Civilization V [20] and Loper's clan. Their skin color was set according to their ethnicity and their heights were defined using a list of average human height worldwide [131]. For inspirational purposes, a concept inspiration was created as well. All of these can be seen on Table 9. For the digital game, it was decided that there would also be a way to easily distinguish which characters were from the same clan. It was defined that characters from the same clan would all share a head accessory which can be seen in Figure 41. This accessory was created based on the clan's

inhabiting region and their known traditions. The concept inspiration for Malik's clan (Karim) can be seen on Figure 42 and all the other clans concepts can be seen on annex 0.

Table 9 – Clan Aesthetics Definition

Clan	Clothes Colors	Average Heights		Concept Inspiration
		Male	Female	
Malik - Karim	Beige, blue and brown	1,70m	-	Arabian Warriors
Ghora - Immandari	Red and blue	1,64m	-	Sikh warriors
Lipp - Hungrakur	Brown, white and dark orange	1,70m	-	Vikings
Kulla - Deshmor	Brown, green and gold	-	1,61m	Albania traditional vests
Wang - Mingzhi	Gray, beige and dark grey (leather)	~1,67m	-	Shaolin warriors
Loper - Voorváders	Yellow straw, dark brown, brown	~1,68m	-	Zulu warriors

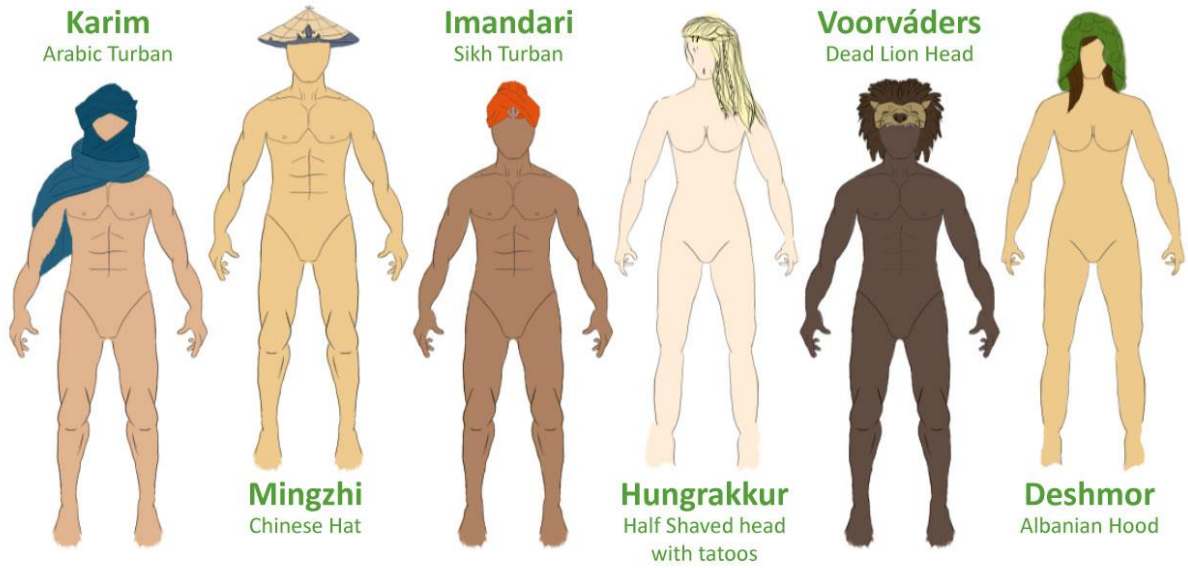


Figure 41 – Head accessory that represents each clan.

Figure 42 – The Karim Clan, where the character Malik comes from.

4.5 GAME CHARACTERS' BACKGROUND AND AESTHETICS

As mentioned previously, the names of the six existing characters, their class and the mechanics of their abilities were obtained from the board game. For each character, a small story of how they became who they are in the game and how they arrived to Intheris was defined. Their school of magic, the inspiration for their weapons of choice and particles' visual language were also defined (Table 10). Since in the board game each hero had eight abilities and only their mechanics were described, it was also vital to establish a fantasy for each ability in order to properly animate it inside the game.

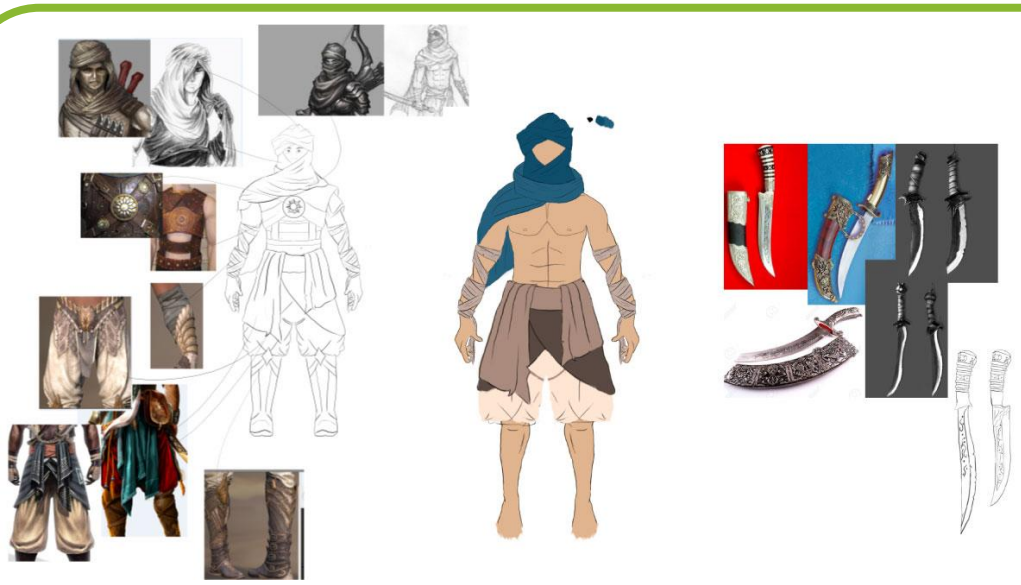
Table 10 – Game character's aesthetics process creation

Character Name	Weapon Inspiration	School of Magic	Particles' visual language	Some Accessories
Malik	Arabian Daggers	N/A	Poison & smoke (green/gray)	Balloon pants made of linen; arm bandages
Ghora	Sikh swords & shields	N/A		Sikh bracelet; Sikh symbol in his clothes
Lipp	Wooden staves and magic crystals	Evocation & Conjuration	Arcane Magic (purple)	Drinking horn; warm clothes made of animal pelts
Kulla	Mechanical bow	N/A	Leaves (green)	Green hoodie; clothes with dirt;
Wang	Chains	N/A	Incense (white)	Socks and shoes like shaolin monks; orange robe;
Loper	Alchemy flasks	Necromancy & Transmutation	Death and Blood Magic (black/red)	Zulu arm and ankle/leg bands; naked body with white tattoos on one side;

After defining each character's concept, their concept design was started. This defined the types of clothing, its colors and materials to use later on the 3D pipeline.

The background story and the fantasy for each of the character's abilities mechanics were also established. Malik's concept design can be seen below on Figure 43 followed by his background story

and the fantasy of his abilities. The concept design for the other five characters, background stories and ability fantasy can be seen on the annex 9.3.4.



Malik's Background Story:

"Malik was a thief for a long time. His family was killed and he chose to avenge them by killing everyone responsible. After enacting his revenge, he became an assassin 'for hire'. After undergoing the assassination trials he arrived at the Intheris arena. He uses two Arabian Daggers, one smaller than the other. Like other assassins, he learned to brew poisons from some exquisite herbs. He now uses his knowledge to make his daggers even deadlier. One of the skills learned while being a thief was the ability to pickpocket."

Malik's abilities:

- Malik1 - Malik can hide in the shadows. By throwing a smoke bomb on the ground Malik disappears, making him almost invisible in game. When leaving the shadows, Malik sneaks behind his target and attacks them in the back
- Malik2 - Malik attacks with his main weapon, if he does not know how to hide in the shadows he also attacks with his secondary weapon.
- Malik3 – Malik attacks once or twice, depending if he had to move to reach his target or not.
- Malik4 – Malik Pickpockets the target. / Malik creates an impenetrable smoke cloud around him making him immune to abilities.
- Malik5 – Malik throws a rock to set off a trap. / Malik wraps one of his bandages around his arm, healing his wounds.
- Malik6 – Malik creates a thick smoke cloud, which protects everyone around him from abilities but also prevents them from using abilities.
- MalikU1 – Malik attacks his target with both weapons. If anyone is around him, Malik will steal their weapons for a moment and attack the target with all of the stolen weapons.
- MalikU2 – Malik poisons and sharpens his daggers, making them glow green. When attacking his target, Malik attacks with both weapons at the same time.

Figure 43 - Malik concept inspiration and story.

5 GAME INTERFACE

This section will explain how the game interface was developed, implemented and tested.

To easily implement the game user interface (GUI), the work load was divided into two parts, the first before starting a game (menus) and the second inside the game (in-game). The menus consist of the part of the game where the player is not yet playing. Players can choose their team, change their avatar, check which heroes they own and their information, buy new heroes in the game's shop, change the game settings and invite friends to play among other things. The in-game consists of the playing field, with the hex grid map where the heroes play against each other and all the current match information that is displayed such as: health bars, timers, important messages, score and any other relevant information.

The menus section was implemented first since, Unity3D had not been mastered at that time. This part was considered easier to start off since, it was mostly navigation between panels instead of complex game mechanics such as timers and health bars.

5.1 GAME INTERFACE REQUIREMENTS

Using the board game to find the game's interface requirements, all of the functionalities needed to implement the game on the computer were raised. There are some requirements that cannot be raised from the board game alone, but from the way the game was idealized in the first place.

In this type of games, players have an account with all of their information, from statistics to their unlocked heroes, this is known as their profile. It is also common to purchase and customize heroes and all of their information can be accessed before entering a game. Players are also able to prepare their team before joining a new game.

The list containing both functional and non-functional requirements can be reviewed in the Annex 9.2.

All of these functionalities must be implemented in the computer game in order to achieve an accurate representation of the board game. In addition to these functionalities, others will be required in order to improve the player's quality of life while playing since for example, in the board game players have to remember every action that has happened.

5.2 MENU

The first step in the development of the menus was a brainstorm to identify how the navigation between each screen was made and where each button would lead the user (Figure 44). By researching how the most popular games displayed this same information, it was possible to create a paper prototype of the entire menus interface. This was accomplished with a horizontal prototype combined with a task-oriented strategy. The first strategy was useful to find issues and lack of consistency within the game screen's navigation. After creating the prototype, specific tasks were used to stress test the interface and ensure once again, that all the interface requirements were met. It is important to note that this part of the interface was not present on the board game and that it was created in the way that most fit the game.

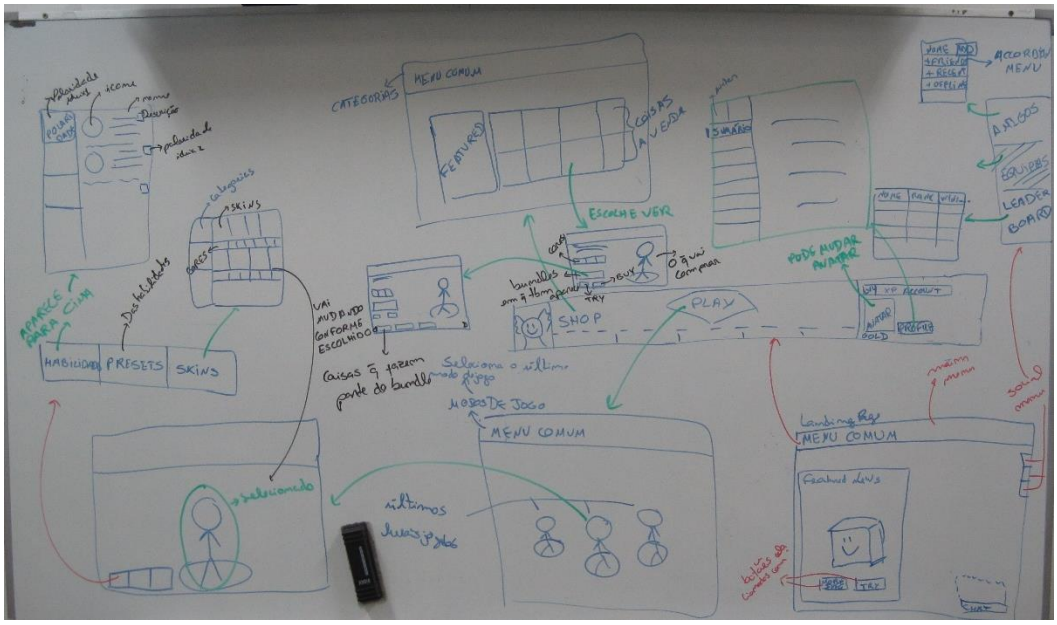


Figure 44 – First sketch of the menu navigation screens depicting their interaction.

A complete menu navigation tree (Figure 45) was created for the menus of the game. It was based on both the brainstorming and paper prototypes created previously. This was then used to assist in the design of user tasks to test the interface later in the development process which, can be seen in more detail in sub section User tests.

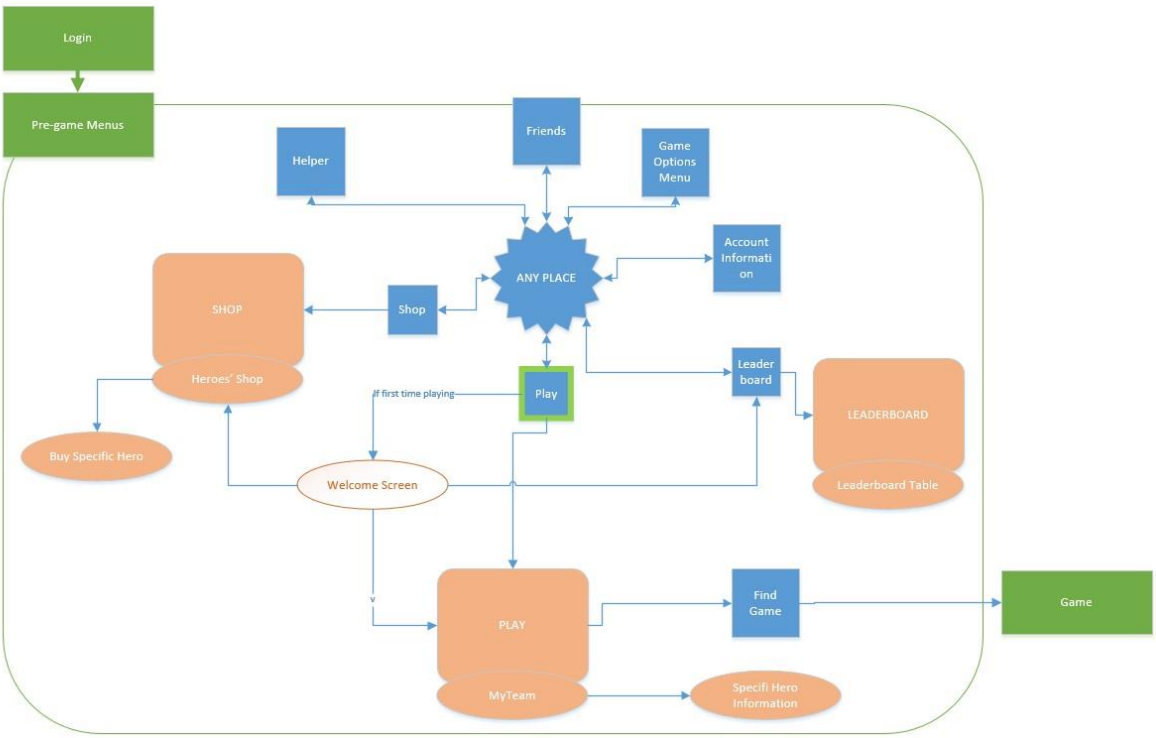


Figure 45 – Menus navigation tree – small version (for a full view of the tree check annex 9.4.1).

After achieving a satisfactory low fidelity prototype, it was then implemented in Unity3D using its interface development features. Unity3D's interface features allow the creation of pre-made buttons, text boxes, radio buttons and check boxes which, can then be dragged to their position around the

canvas. This main purpose of this new prototype was to create an automated navigation system. The board game did not have any icons aside from range and resources. As such, there was a need to create icons for the different classes, for each hero's ability and other game elements. This was done using the Game Icons open source library [132] along with Adobe Illustrator and Photoshop (annex 9.4.2).

Considering the interface requirements and the previously created navigation tree, the menu's interface was divided into six categories: Login, Play, Shop, Profile, Leaderboard and settings. These were also sub divided into smaller categories to ease the implementation process (Table 11). The leaderboard screen was not developed.

Table 11 – Menus interface divided by sub categories

Login and Main Menu	Play	Shop	Profile	Settings
	• Team • Hero's Information • Hero's Presets • Game Modes	• Different Item lists • Specific Item	• Summary • Avatar • Game Heroes • Match History	• Options Menu • Video settings • Audio settings • Help

5.2.1 LOGIN AND MAIN MENU

This type of games only allows the player to create an account on their website. Considering this, the login screens only ask for the information required to login. Below (Figure 46)), it is possible to see the login screen used as inspiration (Warframe [133]) and the corresponding prototype.

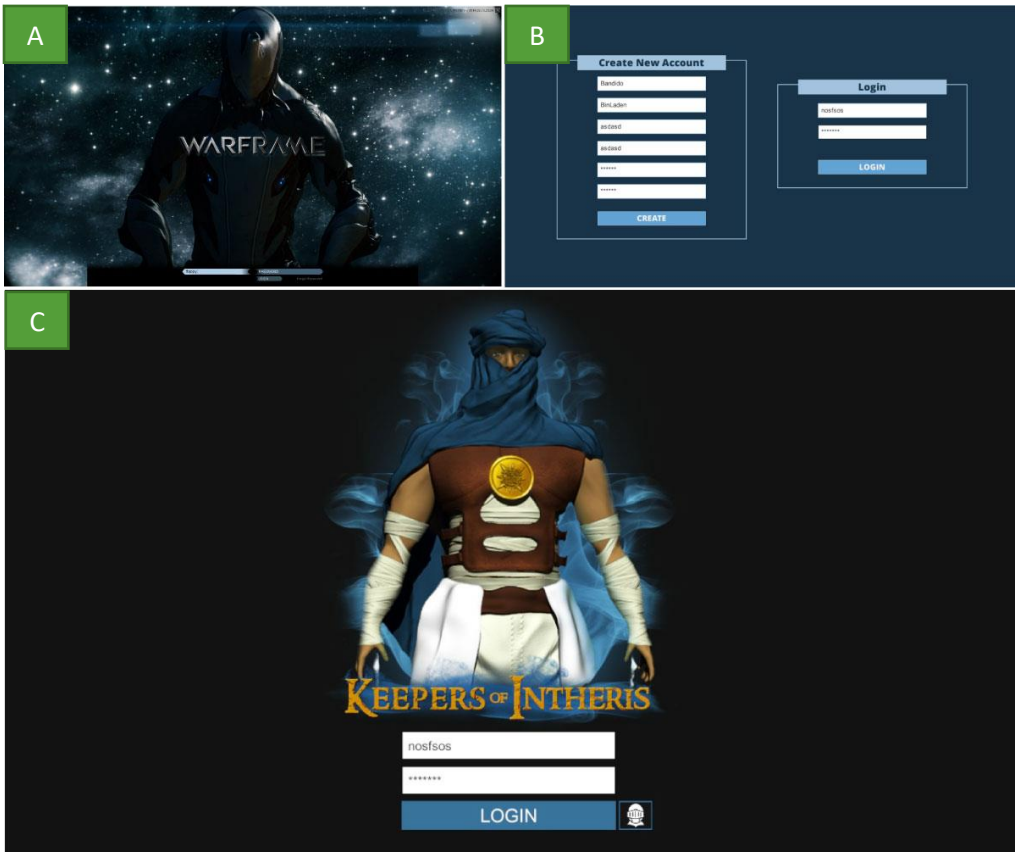


Figure 46 – (A) Warframe login screen; (B) First digital prototype of the login screen; (C) Final login screen.

Regardless of where the player is inside the menu's screen, the main menu will always be visible. Several games have big colored buttons on the top or bottom of the screen that are always displayed. This design leads the player's eyes to the play button which in turn incentivizes them to play more. The same method is used in the interface of keepers of intheris. A big button on top of the screen with a Home title with a small button on each side to be able to go to the shop and check the leaderboard easily. On both top corners, the game settings and player's profile buttons are displayed, using the same GUI design approach as Magicka and SMITE (Figure 47).

Figure 47 – (A) Smite main menu; (B) Magicka: Wizard Wars main Menu; (C) First Iteration of the main menu's paper prototype; (D) Final Game Menu.

5.2.2 PLAY

It is not common for a player to have a team of heroes in this type of games, especially MOBAs, since each player controls a single hero. Despite this, XCom is a good example of a game where the player controls a squad. With the lack of resources on how to show and edit the current team, a similar approach was used to show the player their current team composition (Figure 48). The player can select their team and choose each hero's in-game element.

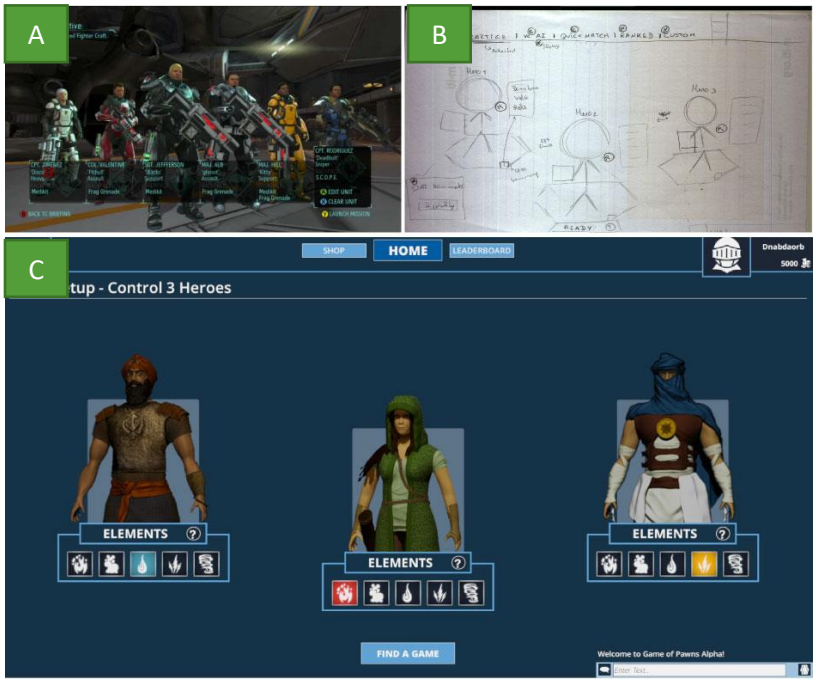


Figure 48 – (A) XCom2 team screen; (B) Paper prototype of the team screen; (C) Final team screen.

On the other hand, most MOBAs have a hero's screen. This screen shows hero information such as health points, abilities, difficulty, small background story, available skins, currency cost, and other information. There is also a place in this section that shows all the heroes inside the game. In some games these two screens are separated while in others they are combined. The game's hero screen was created using the second approach. On one side, there is a panel with all the available heroes inside the game and on the other, all the information of selected hero is displayed. In addition to all the information mentioned previously, each hero also has a list of Presets and their ability slot polarities. This interface design was based on the Heroes of the Storm hero selection screen (Figure 49).

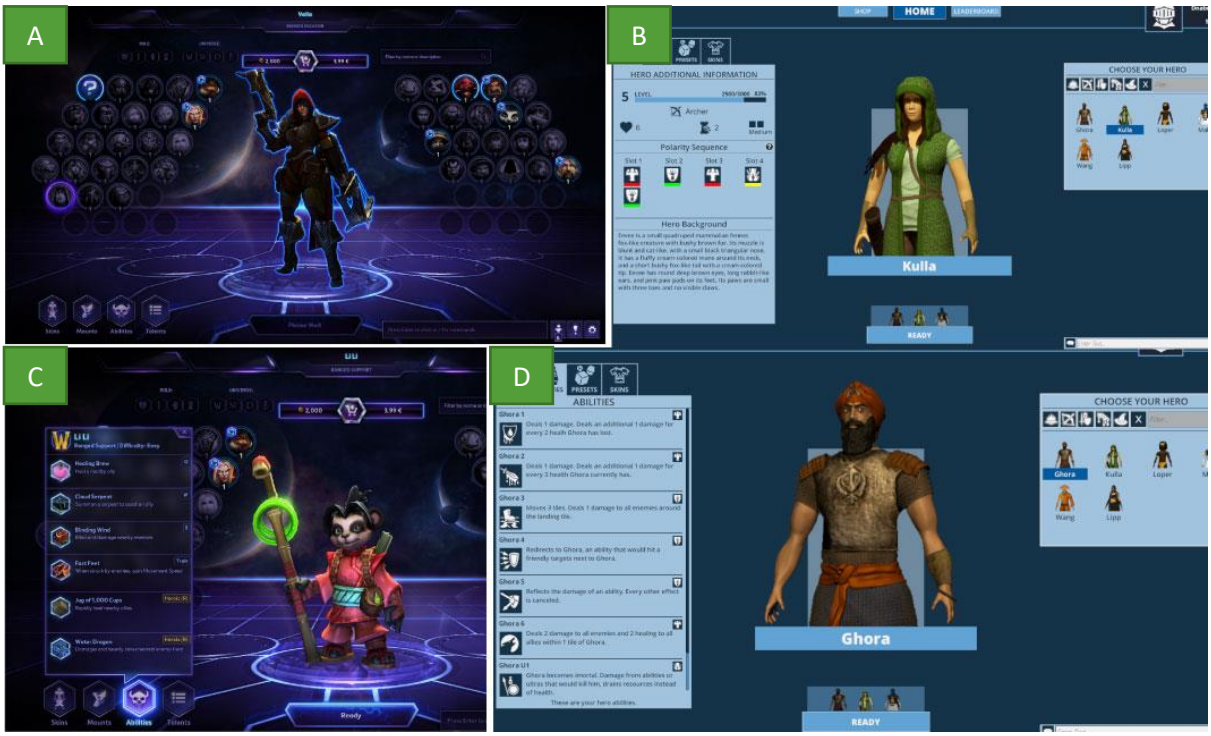


Figure 49 – (A) Heroes of the Storm Hero selection screen; (B) Final hero selection screen; (C) Heroes of the Storm abilities panel; (D) Final hero's abilities panel.

To help new players and, to simplify heroes' in-game progress, each hero has a list of presets that the player can use/add. Each preset is made up by fours abilities (based on the ability slots available to that hero), a weapon skin and a hero's color pallet. This specific information is not usually seen in this type of games but, MOBAs are known for hero progression through item acquisition/upgrade during a match. Using this mechanic, although different from the game's presets, a player can create item sets for each hero, so the same interface design approach can be used to create the hero's preset list. A good example of this is design approach of League of Legends'. The game's preset list was created using the same idea (Figure 50).

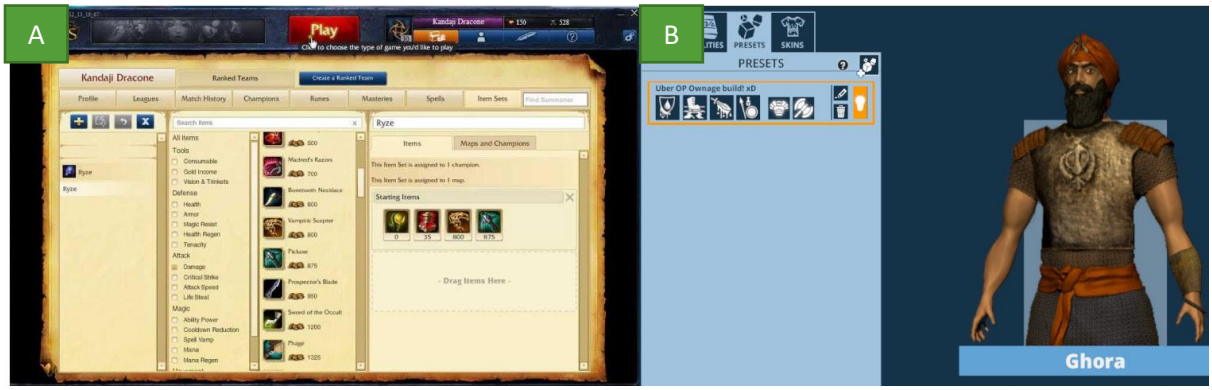


Figure 50 – (A) League of Legends Item sets screen; (B) Game's final preset list screen.

Players can choose between different game modes when they want to play a game. These game modes are presented to the player in several ways. Some games have a game mode selection screen which is the last step before going into a match. Other games allow the player to choose the game mode and then let them navigate throughout the game menus until it finds a match to join. This second option is usually used when several players are required for each team to create a game. Considering that the game being developed only needs two players to start a match, the first approach was chosen. Its design was based on the game interface of Magicka Wizard Wars which has three choices: Practice mode, versus and custom (Figure 51).

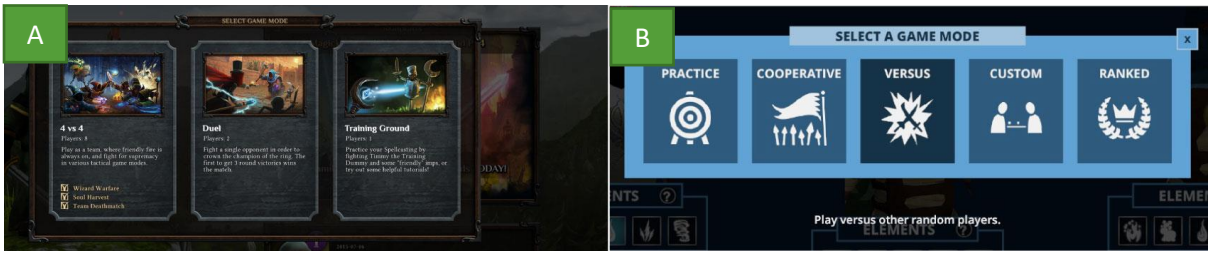


Figure 51 – (A) Magicka: Wizard Wars Game Modes screen; (B) Final game modes screen.

5.2.3 SHOP

Since many games have an in-game shop that allows players to buy game items, a shop menu was also created. Like game interfaces, shop interfaces are similar to each other within the same game genre. The game's shop was inspired by the MOBA subgenre and is very similar to the Heroes of the Storm shop (Figure 52).

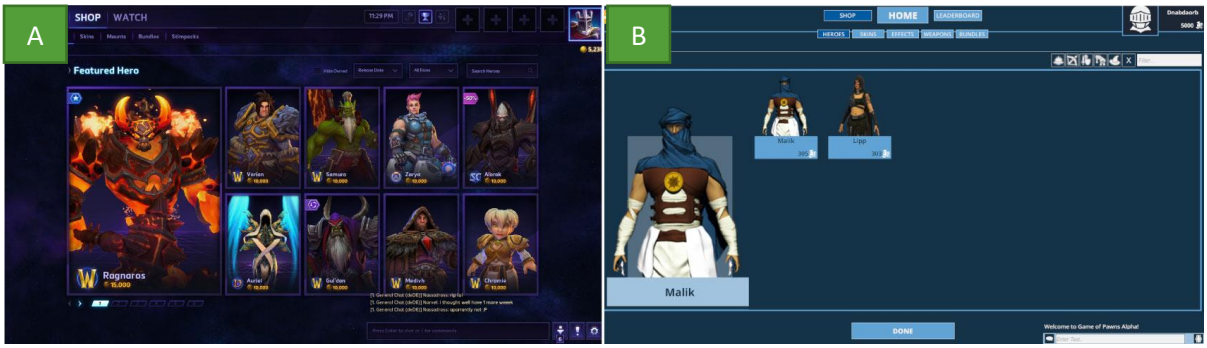


Figure 52 – (A) Heroes of the Storm Shop Menu; (B) Final game shop screen.

When buying an item, especially a hero, a lot of information needs to be displayed. Heroes of the Storm is again a solid example on how this information can be displayed. Its interface design concepts were used in the DAP. Each hero has information about its abilities, background story and currency cost as shown in Figure 53.

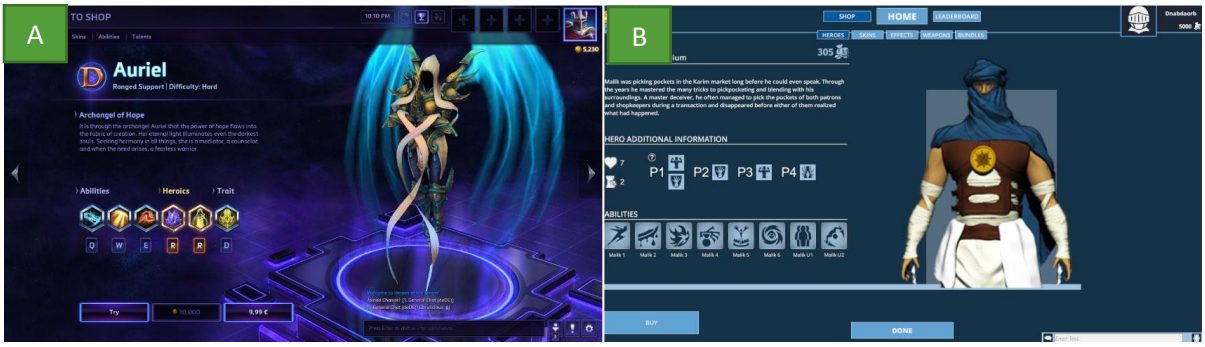


Figure 53 – (A) Heroes of the Storm Shop specific item screen; (B) Game shop specific hero screen.

5.2.4 PROFILE

The profile is a common piece of information in any game. Players enjoy having their account progress, statistics and all other information in a single place. MOBAs in general show this information very well (Figure 54). They have a tab with statistics and recently played heroes, a tab with information on all the heroes in the game and another tab with information about every match the player has played (Figure 55 – A and B).

Figure 54 – (A) Smite profile summary; (B) Game final profile summary screen; (C) Heroes of the Storm profile hero list screen; (D) Game final profile hero list screen.

Something else that is also very important for players is the ability to 'show of' their recent game successes. Part of this is through their account's avatar. In the profile, there is a place where players can change between avatars that they have unlocked (Figure 55 - C and D).

Figure 55 – (A) Smite match history screen; (B) Game's final profile match history screen; (C) Heroes of the Storm avatar list screen; (D) Game's final avatar list screen.

5.2.5 SETTINGS

Like in all other games, a game settings panel is required. In general, all games display it in the same way and, the only deviation is in the amount of settings provided. On the audio panel, the volume can be changed and all the audio channels can be turned off and on. The video settings panel is where all the game quality settings can be adjusted and the resolution can also be changed (Figure 56). In addition, almost all games also allow players to change the game's default key bindings and language. In the DAP, all the game's interface was designed and prepared for later implementation, including all the settings mentioned above. This was one of the screens that was not implemented in the final version of the game since, this was a proof of concept of the board game.

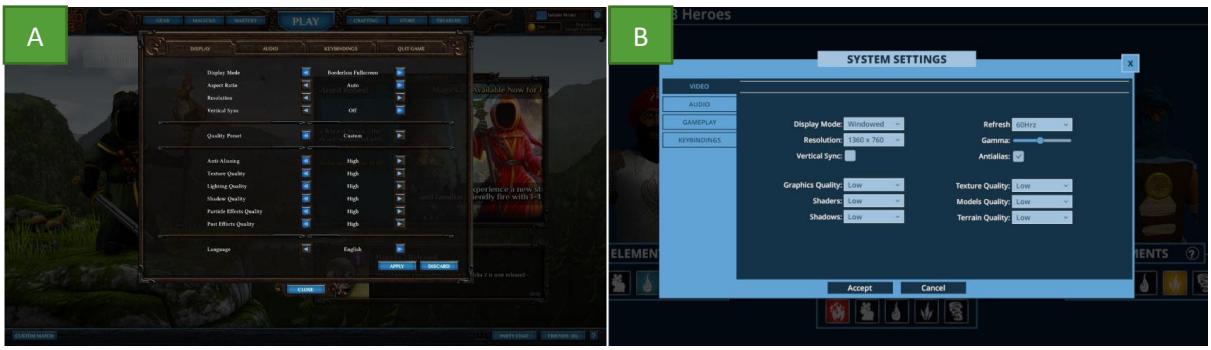


Figure 56 – (A) Magicka: Wizard Wars settings screen; (B) Game's final settings screen.

Players also enjoy it when the game explains everything through tooltips and helpers. The Civilization series is known for its civilopedia menu, which works as an in-game encyclopedia. Although the game being developed is much smaller, it still has a fair amount of complex information. A similar system was created and named 'Keepopedia'. Throughout the game's interface there is a help button that players can click and they are redirected to the 'Keepopedia' section (Figure 57).

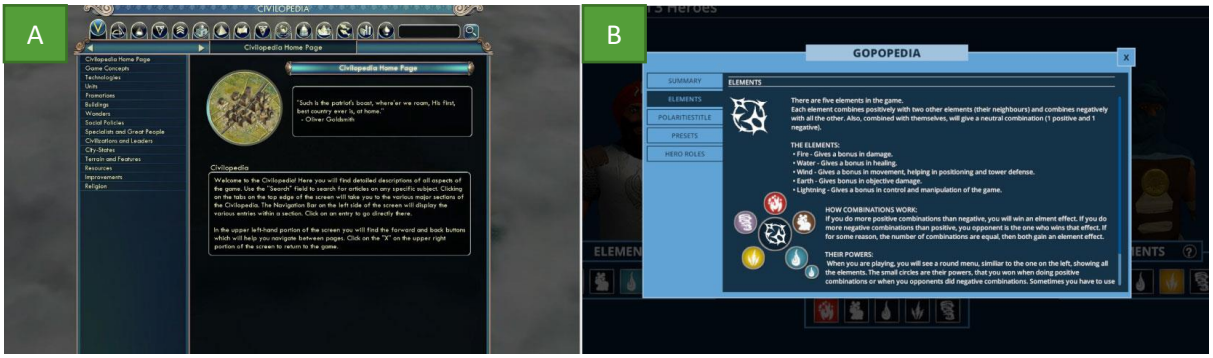


Figure 57 – (A) Civilization V helper screen; (B) Game's final helper screen.

5.3 DEVELOPMENT OF THE IN-GAME INTERFACE

A brainstorming session was also done for the in-game portion of the game interface. This helped to decide which positions were most suitable for each part of the game's functionalities. A paper prototype was created to verify if all the game elements required were being displayed but also to experiment with different layouts. These prototypes were created using the same approaches as the Menu screens' prototypes, by using the previously researched games and how they displayed their information (Figure 58). The only difference between the creation of the two prototypes was that for this part of the interface, the board game already provided all the requirements. It was simply a matter of finding the best way to position and display information to ensure that players felt that the interface was familiar.

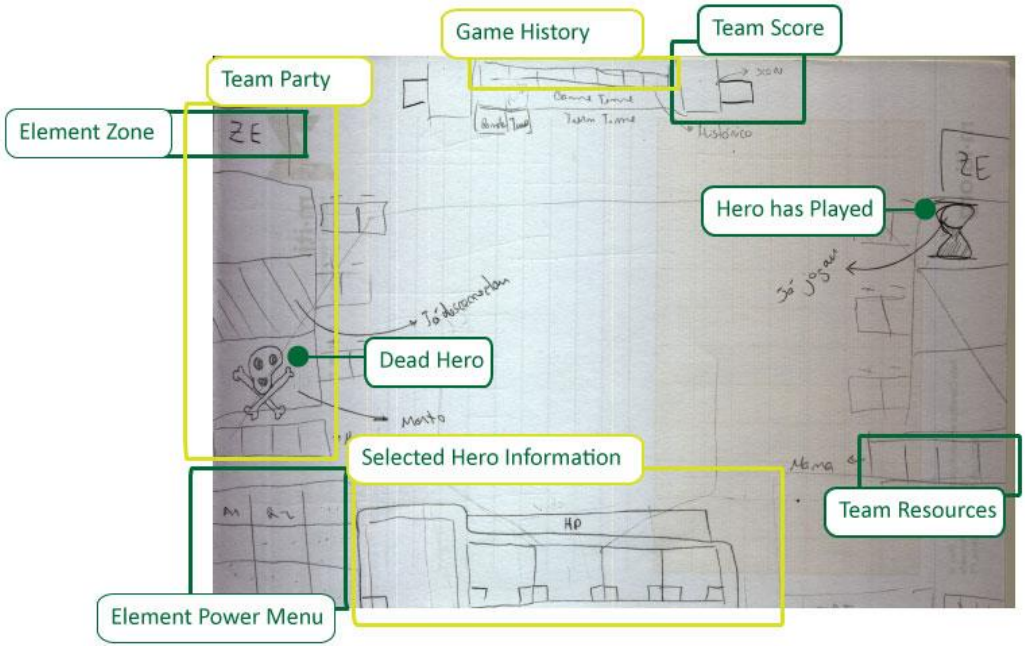


Figure 58 – First paper prototype of the in-game screen.

There was no navigation tree since there was only a single screen with a lot of information about the current state of the game. Although it is a single screen, it is ten times more complex than the menu's screen and so, it was also divided into sub-categories to ease and speed up the implementation process. Aside from the options menu, which was already implemented in the menu's screen, this screen needs to display information on: the currently selected hero (Hero Information Bar), ability and/or element being used (Targets Window), both teams (Player and Enemy Party) and their information (Team Panel), game history, element power's system and information (Element Power), messages and error systems and, game timers (Figure 59). In addition to all this, there were still the spatial interface elements such as, floating health bars and combat text for each hero and tower, the map feedback and all the particle effects that were used to provide feedback to the player on what had just happened. Throughout the game, both players can always see each other's actions and any information related to each team except which specific element powers they own.

Figure 59 – Brief explanation of the In-game Screen.

5.3.1 HERO INFORMATION BAR

The hero information bar shows all the necessary information to be able to describe the hero's game state at any time. In games, such as MOBAs, a player only controls one hero, so that information is always visible (Figure 60 - A). In the game being developed however, a player controls three heroes and, needs to be able to change between them. Using MOBAs GUI's approach, a hero information bar was created on the bottom of the screen that has information such as the hero's health points, all its active abilities and their states (used, cycle is on/off, not unlocked), buffs and debuffs, element power information and its combinations, hero's polarities and their portrait which also represent the hero's state (has played, is dead, can play) (Figure 60 - C). Every time a player chooses one of their heroes, the bar is updated with the selected hero's information. Since this is a turn based game, it is also necessary to be able to start and end each hero's turn. Therefore, a Turn button was also added to show that the hero can/cannot start the turn, similar to Hearthstone's end turn button (Figure 60 - B).

Figure 60 – (A) Dota2 Hero Bar; (B) Hearthstone end turn Button; (C) Game's hero bar implementation with a brief explanation of each component.

Heroes can unlock or change abilities as the team progresses in the game. The closest implementation of this system can be found on Heroes of the Storm when a hero gains a new level (Figure 61 – A The same method was used to implement this system on the game being developed. (Figure 61 – B) When a player selects an ability, it becomes highlighted similarly to what happens in XCom (Figure 61 - C). This opens the ability information window (Targets Window) (Figure 61 - D).

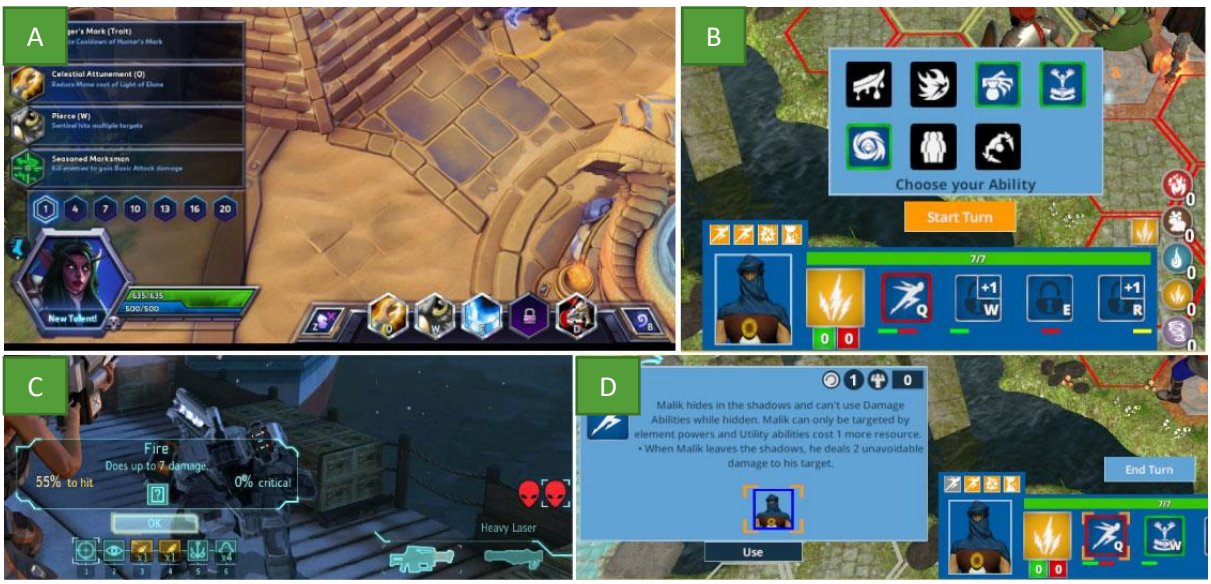


Figure 61 – (A) Heroes of the storm when a hero gains an ability; (B) In-game screen showing a hero gaining an ability; (C) XCom when a unit uses an ability; (D) In-game screen of a hero using an ability.

5.3.2 TARGETS WINDOW

As mentioned before, the system that allows the use of abilities in a turn based game is slightly different from real time games. In this game, there are abilities and element powers that target/affect heroes, towers and even tiles on the map (hexes). Everything must be done in a certain order to prevent confusion to the player but, at the same time, to provide all the necessary information. Although XCom has a good targeting system GUI, it was not enough for what this game required. In the game being developed, there is a target's window like in XCom which shows information of an

ability such as then name, icon, description, when it can be used, range, cost and type. If there are any targets that the player can choose, they are highlighted on the map so the player knows what are their choices. As the player fills in the targets by clicking on the 3d targets, the 'Use button' is activated and the player can then use the ability or element power (Figure 62).

Figure 62 – Brief explanation of the Targets Panel for the ability selected.

5.3.3 PARTY AND TEAM INFORMATION

Team information in MOBAs is shown as small portraits of each team's hero. Each team also has their score displayed on a score board with the number of kills, assists and deaths. This approach was not sufficient to display all the information required by the board game (Figure 63 - A). On this case, only the team had resources instead of each hero, similarly to Hearthstone's player mana GUI approach (Figure 63 – B and C). In the game being developed, each portrait has the hero's health, their states (dead, has played), buffs and debuffs, element power, its combinations and all the hero's current abilities. The information regarding the abilities can be toggled to free up screen space (Figure 63 - D). Like heroes, towers also have a portrait to display their current game state (dead or alive) and their current health points. In addition to being able to target each hero or tower by clicking on their 3D object, it is also possible to target them by clicking on their portrait.

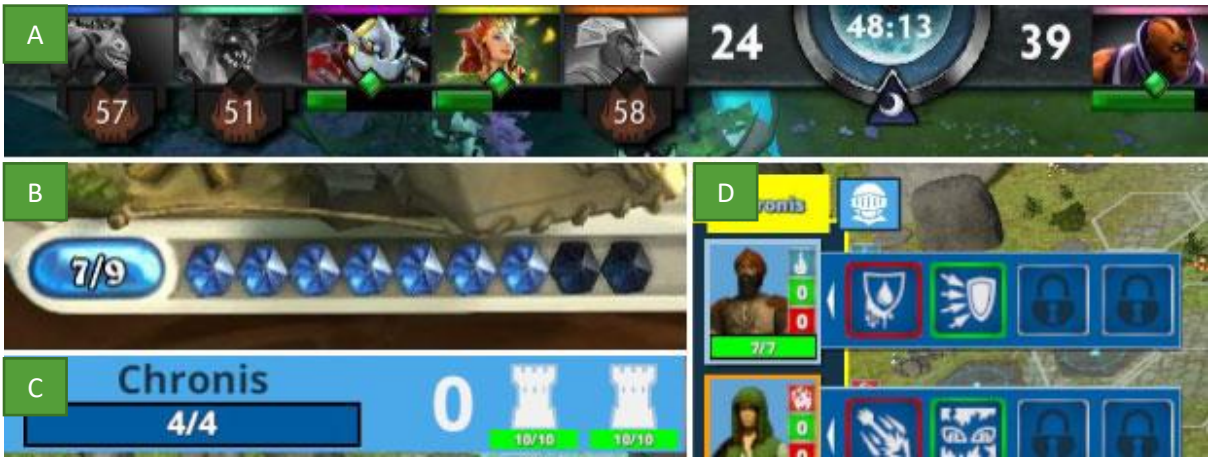


Figure 63 – (A) – Dota2 party portraits; (B) Hearthstone mana bar/gems; (C) In-Game team resources bar; (D) In-Game team hero information.

5.3.4 ACTIONS HISTORY

As previously mentioned, the actions history is a very important aspect of turn based games. This was an interface requirement for the game being implemented, and it was based on the Hearthstone's game history GUI. The player's actions that needed to be documented are the use of abilities and elemental powers. In addition to their icons, both the caster (hero or team) and the targets are shown, along with any health modification (damage or heal). If the action is an ability, the hero's combinations are also shown (Figure 64).

Figure 64 – Game history panel showing the last ability used.

5.3.5 ELEMENT POWER SYSTEM AND INFORMATION

A large amount of information must be displayed to provide feedback on the use of elemental powers. Both teams can hold a certain amount of elemental powers which on the board game translated to elemental cards. It is also possible to draw cards and use them. Any digital card game, such as Hearthstone and Magic: The Gathering Online [134] can be used as a good starting point for the GUI since they all play with cards and this specific system is mainly a card game (Figure 65 - A).

In the game being developed, the player can only see their cards and only knows how many cards of each element the other team has (Figure 65 - D). This is shown near the team's party. Every time a player wins an elemental power, a big screen is shown with additional information on that specific elemental power (Figure 65 - E). When the enemy team gains an elemental power, the player can see a small bubble going into the team's party.

Players also needed feedback when they use a card inside the game, so particle effects were created to represent the use of each elemental power (Figure 65 - F). Fire and water are represented by a stream/spray of themed particles, earth is represented by an earth meteor, wind is represented by tornados and lightning is represented by a cloud of lightning.

The player has also a small menu that shows all the elemental cards that they currently have. It is possible to go through these by selecting each elemental bubble (Figure 65 - C). There was an additional interface requirement that the game had to accommodate, the elemental activation zone. At the start of each round, players can activate elemental powers that will last the entire round. These will then be displayed in each team's Elemental Zone, on their side of screen (Figure 65 - B).

Figure 65 – (A) Hearthstone gameplay screen showing the cards of both teams (one team at the top and another at the bottom); (B) Element Power Activated; (C) Element Power Menu; (D) Element powers owned by the enemy team; (E) Wind element power won (F) Fire element power used (particle feedback).

5.3.6 GAME TIMERS

Many games have timer bars, not just turn based ones. For example, MOBAs, have a resurrection timer that starts when the hero dies and lasts until they can spawn again while, turn based games often have turn timers to display how much time a player has left to perform all their actions during that turn (Figure 66 – A and B).

For the game being implemented, apart from a turn timer, an elemental zone activation timer and an ability counter timer were required. All bars work the same way, they start as a full horizontal bar and

decrease size as time goes on. For the latter two bars mentioned, an additional panel was added from where the player could choose the elemental powers or ability counters to use (Figure 66 – C and D).

Figure 66 – (A) Heroes of the Storm resurrection timer; (B) Hearthstone turn timer; (C) Element power use timer; (D) Hero turn timer bar at the bottom of the hero bar.

5.3.7 SPATIAL INTERFACE ELEMENTS

As in most games, there are also some spatial interface elements in this game. Every hero and tower have a health bar associated to their 3D model, like in most RPG games. There are also particles associated with some of the map objects to provide feedback on different states. For example, the hero that is currently playing has a small orange circle beneath its feet with particles around it and, as the player uses the hex grid, different hex states are shown, such as the movement path and all possible hero movements, similar to XCom's movement phases.

Figure 67 – A) XCom movement grid; (B) Map grid showing hero's movement path (yellow first phase, red second phase); (C) Map showing all possible movements for the hero (yellow first phase, red second phase); (D) Tower's 3D in game model with its health bar; (E) Hero's 3D in game model with its health bar.

5.4 IMPLEMENTATION OF THE MENUS INTERFACE

Unity allows scripting in three different languages, Boo (similar to python), Unityscript and C#. The game in the GIP was developed in C#.

In Unity, scenes contain the objects inside the game, from menus to individual game levels. Every object inside a Unity scene is considered a GameObject. These objects can have components or scripts that add additional properties to them. Scripts are classes in C# and by deriving the class from MonoBehaviour, it is possible to add it to an object. Every time a script with the MonoBehaviour derivation is added to an object, Unity considers it as another component of that GameObject. These objects can also be converted into prefabs which, act as templates that can be cloned to new objects with the same components and properties.

Unity also allows the user to add Interfaces to a GameObject as long as, they also derive from MonoBehaviour class. Interfaces will work as they normally do in C# coding.

As mentioned previously, the interface was implemented early in the game development cycle. To design the GUI, the first step was to define how many scenes were required. The game was divided into three different scenes: Login, Menus and In-game. Later, more scenes were added such as the preload scene that is used as a place holder for the game's loading screen and the map creator scene which was used in the MTP [2].

Subsequently, the GUI's hierarchy in Unity was created for the Menus scene using the sub categories defined previously as a starting point (play, shop, profile and settings) (Figure 68 - A). Every time dynamic information would be received, a prefab was created. The prefab's parent was also prepared to receive as many prefabs as required without having to prepare it in the code every time. This was done to ensure that there was no need to create each GUI asset from scratch in the code.

Using this hierarchy, a Model View Controller (MVC) pattern was used to separate different game responsibilities. The Model is where the Data structures are, the view is formed by the interface input detection, rendering and actual interface layout and, the controller is an intermediary between the model and the view that has all the logic to respond to the user's commands. In the GUI implementation, the Models were the data structures of the game, which were first created with only the information for the game's GUI and were updated later in the MTP [2] to be able to contain all the information in the database. The views were the screens in the menu's scene with their interactor classes associated to each category along with their controllers.

For example, in the Play category, there is a Hero information screen, where all heroes are listed. When the player chooses one hero, all the information is updated on that screen because there is a controller (PlayGuiController.cs) which controls and decides what to show to the player and an interactor (PlayGuiInteractor.cs) which is associated to each Play screen button that detects the player's choice and, alerts the controller so that it can update the view with new information. If, for example, the player decides to change the current team composition then, the controller has to communicate to the model so it can be updated and then communicate back to the view that there is new information to display (Figure 68 - B).

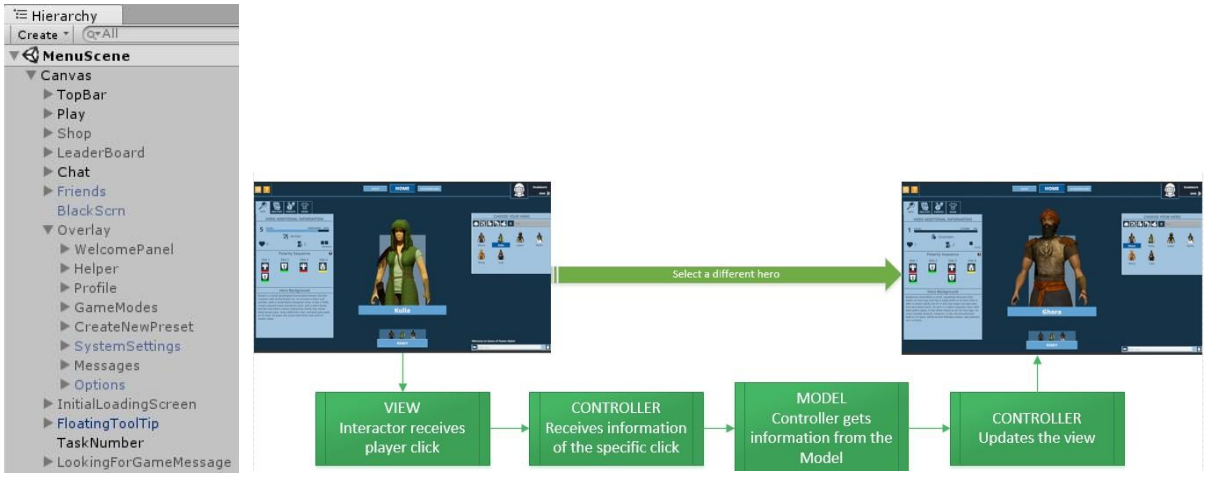


Figure 68 – (A) Menu scene hierarchy; (B) MVC pipeline in the implementation of the game's menus.

5.4.1 GUI DATA STRUCTURE

When the interface was implemented, the database structure and implementation were still under development on the MTP [2]. To be able to fully test the functionalities implemented, data was required. A database was created using only dummy information to fill in the data for the Models and to be able to see that information in the GUI.

Since all the development was done before having the real data but all the GUI's data structures were already created, an Interface was created to be used as template for the other database implementations used in the MTP [2]. This way, when the real database was complete, there was no need to change the GUI's implementation since it was using the same Dictionary and Lists. Only the data inside of these was coming from a different database (Figure 69).

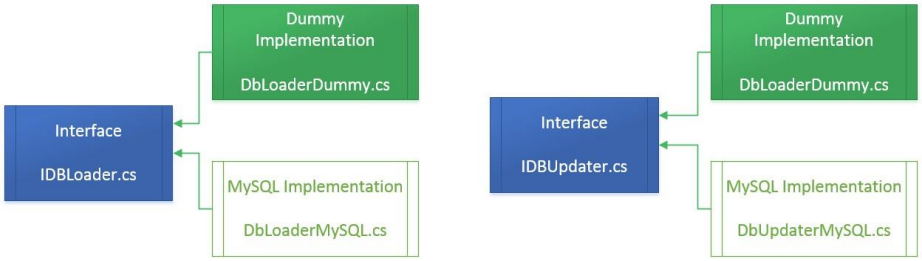


Figure 69 - Both dummy implementations, which were implemented in the DAP, are represented in green. Blue represents the interfaces and white represents the MTP's implementation.

Every GUI item was required to have a code name, which was assigned through a component added in Unity3D (CodeName.cs). This code name was comparable to an id in a database, but it was used to identify every item of the GUI which, could then be identified by the controller. This component along with XML was the core of the interface. This was due to the fact that a major part of the features implemented relied on them such as, the tooltips, multiple languages and the icon systems.

5.4.2 XML INTEGRATION – MULTI LANGUAGE AND ART SYSTEM

Due to the type of systems required to implement the interface, most of the implementation was done relying on XML. XML is simple and, each tag can have unlimited attributes. This is useful to store

a large amount of information without losing its context [135]. A multiple language system was also implemented using XML. The XML data was stored in different Dictionaries at the startup of the game and then using the interface managers together with the codename information, each button, input box and text field were filled with the information from the XML file and shown on the interface. When a player changes the game's language, the game needs to be restarted and will then load the language specific XML data with the correct language (Figure 70 - A).

XML was also used to map the interface icons. Instead of changing an icon every time the interface evolved or new icons were created, an XML icon list was created. Using the same approach as the multiple language system, a Dictionary stored all the icon's codenames and their file names inside Unity3D. Data structures, such as the Ability Core, which held all the static information of an ability in the game had the icon's information stored and, at the start of the game, Unity would show the proper icons according to the information in the XML file and the ability's constructor (Figure 70 – B and C).

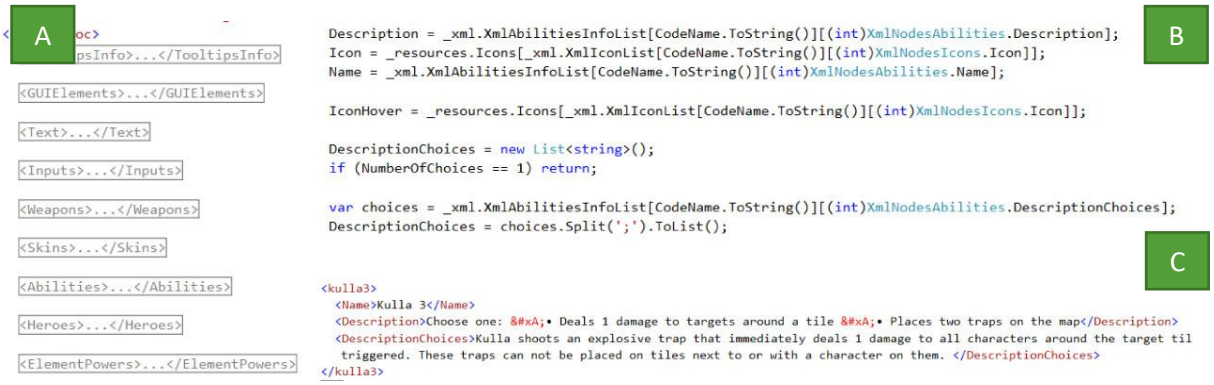


Figure 70 – (A) Xml base structure; (B) Ability constructor which fills in all of the static variables from xml; (C) Xml ability structure.

Art managers were also created. These were responsible for rendering the button's colors, icons and font styles, in other words, the interface's item style. These art managers were used together with a button manager which, was similar to Unity's toggled button group standard feature but had additional information such as the XML data and the art manager's information. Since there were so many art managers, an Interface was created (IArtButtons.cs) which was used to communicate with the button manager so that it knew which art manager it should communicate with (Figure 71).

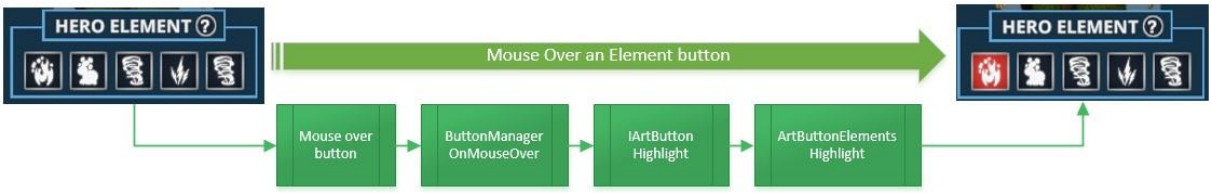


Figure 71 – Pipeline showing how the GUI knows what color to show for each element .

This system was very helpful in streamlining the interface implementation since, any new interface items created only required a size, position and the corresponding scripts. Then, at run time the item would be modified in the code making its style match the other items. This also provided the ability to change the color pallet of the GUI by simply changing the script's base colors.

5.4.3 XML INTEGRATION – TOOLTIP SYSTEM

One of the goals of the game's interface was to get players quickly acquainted with the interface, and to accomplish this a tooltip system was implemented. This system consisted of two components, TooltipManager.cs and TooltipInitializer.cs. The former was added to any interface item that had to have a tooltip and the later was added to the tooltip object. Using an XML file and the interface's item Codename, the ToolTipManager.cs is able to fill the tooltip with the item's information (Figure 72).

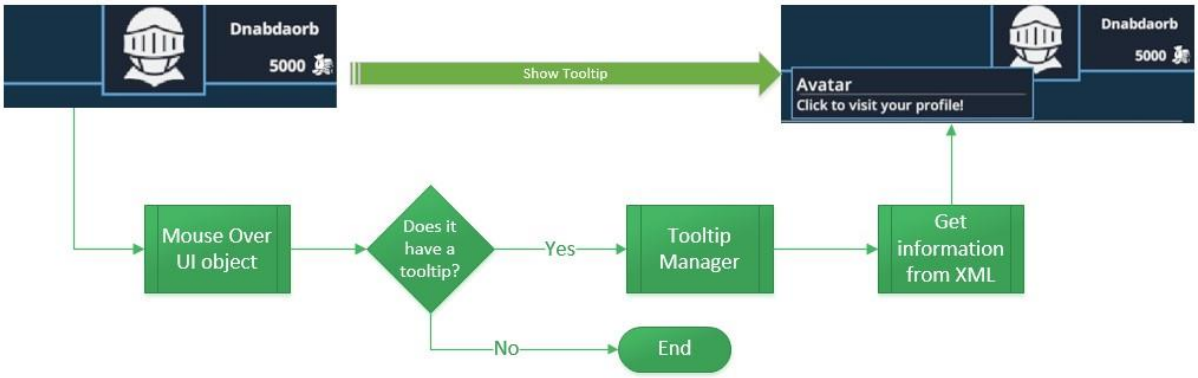


Figure 72 – Pipeline of how the GUI knows when to show a tooltip and what to show on it.

5.5 IMPLEMENTATION OF THE IN-GAME INTERFACE

Using the same approach as in the implementation of the menus, the MVC pattern was used to separate different game responsibilities. As mentioned before, there was only one screen in this part. However, since the number of functionalities had increased, the different parts of the screen mentioned above had a different controller and interactor each, following the same reasoning as in the menus' implementation. However, each section of the screen has its own responsibilities. For example, the party controller script (PartyPortraitGuiController.cs) is responsible for populating and updating the hero's information in its party portrait at all times. Every time a player clicks the hero's party portrait, the interactor (partyPortraitGuiInteractor.cs) handles the click and communicates with its controller which takes care of the rest.

There is a global script that initializes everything related to the GUI, 3D models and map colors (GameInitializer.cs). This can be considered as a game loader that instructs all the controllers to initialize their responsibilities. This controller, then populates Unity's hierarchy that has all the information about both players and their team heroes in the game (Figure 73 - A). This hierarchy is the core of the game and from there, any game state information can be obtained. This can range from the hero's position on the map to any current effects applied to them. For example, each hero is represented in the game by the class Monohero.cs, which has a reference to the hero's core information (maximum health points, base tower damage, name, polarity sequence, list of abilities (HeroCore.cs)) and to the instance of the specific hero (IngameHero.cs) which has information on the current state of that hero at all times (current health, unblocked abilities, current hex). The Monohero class also has a reference to its party portrait controller, its 3D model in the game, its animation and particle controllers and, all the other elements that are related to a hero in the game (Figure 73 - B).

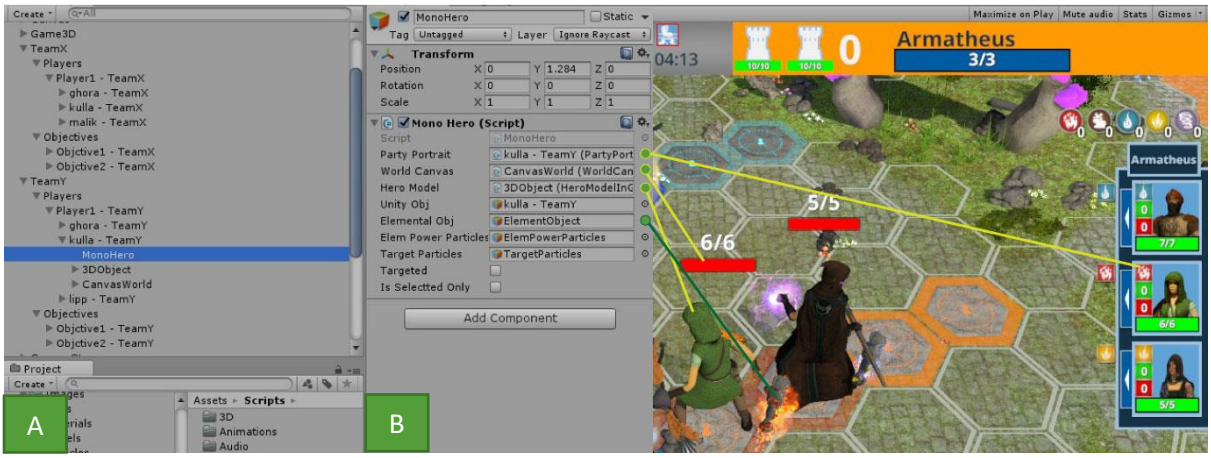


Figure 73 – (A) Unity3d In-game hierarchy; (B) MonoHero script with a reference to everything about a hero in the game.

Unity does not support multi thread processing, so, to be able to wait for player's inputs in different parts of the game, Unity's coroutine system was used. This is very similar to a busy wait but does not allocate additional resources every time the coroutine yields or resumes, making it very useful as long as the developer limits the number of coroutines while the game is running [136].

Unity allows the creation of custom editors which can be considered as Unity's plugins. These custom editors were developed to implement all the timers and, the bars for resources and health. This allowed the use of a simple Observer design pattern where every time the bar value was changed, the bar size and color were updated to reflect this change. In the future, these custom editors can be used in any other games created in Unity simply by adding the class to the proper Unity project.

5.6 USER TESTS – MENUS INTERFACE

User tests were performed on the Menus Scene of the game. These were used to assess the usability and satisfaction with the prototyped interface. Using the navigation tree (annex 9.4.1), a list of tasks was created to ensure that all the different menus were explored (Table 12). From this list of tasks, a user test guide was created to guarantee that all tests were consistent (annex 9.4.2).

To be able to know if the user was performing the tasks properly, a Task Log script was created. This script saved every mouse click to a line in an excel file. Each line had the task number, the time, the button/blank click and the X and Y coordinates of the mouse position (Figure 74). The script made it possible to go through each task as the test progressed by adding a new line for each task's beginning and end. With this information, it was then possible to calculate the time between clicks, the time it took to perform each task, number of clicks for each task and even, the number of wrong clicks since the buttons that had to or could be pressed to be able to perform a certain task were known. This Task Log script can be used in the future on additional tests on this game's interface or, other interfaces to be able to collect the same information since it was made using Unity3D's component system.

Table 12 – List of tasks used in the user test

Task number	Task Brief Description
1	Change your resolution to 1920x1080 and mute the volume
2	Filter the heroes by only showing assassins
3	Check the abilities of one of the assassin heroes
4	Buy an assassin hero
5	Buy 2 other heroes
6	Check the helper
7	Create a preset for one of your heroes - add 4 different abilities, a weapon and a skin
8	Change your avatar picture
9	Check the hero you created the preset to
10	Create a new preset and Activate it
11	Delete the first preset
12	Go to the shop and see which heroes are still available for sell
13	Get your hero team ready and give each hero a different element
14	Play a game vs. AI

StatTester42 - 13-04-2016 09:48:21

TASK N	TIME	TIME MINS	BTN	TAG	CURRENT HERO	MOUSE X	MOUSE Y
Task 1		09:48:54				41	258
Task 1	1,640503	00:00:02	WelcomeSkipTutorial	Untagged	no hero	0	104
Task 1	11,20935	00:00:11	SystemSettingsBtn	Untagged	wang	-665	355
Task 1	22,79578	00:00:23	OptionsVideoBtn	Untagged	wang	31	144
Task 1	25,72565	00:00:26	OptionsVideoBtn	Untagged	wang	2	73
Task 1	27,08881	00:00:27	OptionsVideoBtn	Untagged	wang	-26	26
Task 1	30,80817	00:00:31	AudioBtn	systemMenu	wang	-318	137
Task 1	32,79175	00:00:33	AudioBtn	systemMenu	wang	-83	98
Task 1	36,15021	00:00:37	OptionsAccept	Untagged	wang	-34	-191
Task 1		09:49:34				-36	-180

	Min	0:00:02	9:48:54
	Max	0:00:37	9:49:34
Task 1	ExecutionTime	00:02	0:00:40
	MissCliques	0	
	CliquesT1	8	
	wrongclicks	0	
	Min	0:00:10	9:50:06
	Max	0:00:35	9:50:41
	ExecutionTime	00:10	0:00:35

Figure 74 – Part of a task log.

A game dedication questionnaire was created based on the work of Ernest Adams [137] and was used to define the game dedication level of each tester. This would later make it possible to assess if the number of errors was due to the interface or due to the lack of experience on this type of games. This questionnaire uses the familiar Likert scale with five levels (1 – strongly disagree, 5 – strongly agree). To calculate the game dedication, the Factors of Classification and associated weighting table were used together with the answers from each user. The gamer-dedication score was calculated using the formula: $Game\ Dedication = \frac{\sum_{j=1}^{n} w_j s_j}{\sum_{j=1}^{n} 5 w_j}$ where n = 15 (number of questions), w =

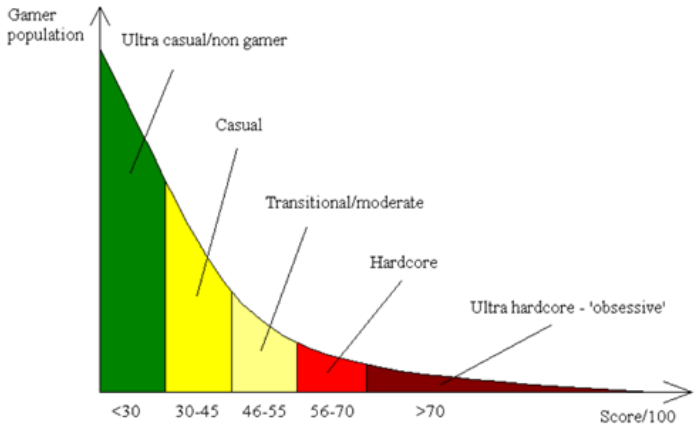


Figure 75 - From Casual to Core: A Statistical Mechanism for Studying Gamer Dedication – Definition of each score vs possible 5 categories.

weight and s = the user's answer. The data was then interpreted according to Figure 75 - From Casual to Core: A Statistical Mechanism for Studying Gamer Dedication – Definition of each score vs possible 5 categories. The higher the score the more hardcore/dedicated a player is.

A User interaction satisfaction questionnaire was also used based on [138] development to help to find additional problems on the interface. This questionnaire also uses Likert scale with ten levels and was only considered as a qualitative evaluation.

5.6.1 USER TEST SETUP

Before starting the tests, a pilot test was performed to find any bugs on the Task log script and to make sure that the test guide could be followed. A final test was then performed by an experienced player as part of the pilot test (this player only participated on this internal test). During this pilot test, a complete log of every task and the time it took to perform it was recorded. This time was then used as a as comparison for each iteration.

Since there was no art developed at the time and, to make sure that all the testers were familiar with the game's heroes, to each hero was given a Pokemon portrait. Two iterations were performed on the game interface. Both iterations had the participation of eight different users. For each test, an additional keyboard was used to change between tasks so there would be no interference in the user experience while testing (Figure 76) and every test was recorded with the consent of who was testing.

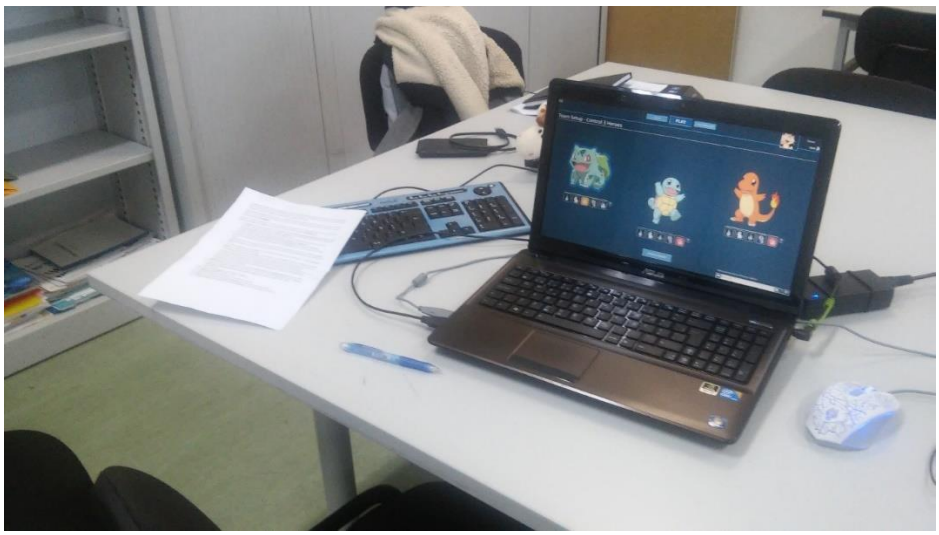


Figure 76 – User tests: laptop and keyboard Setup.

5.6.2 RESULTS – FIRST INTERACTION

The first iteration of the interface was the one that generated the most changes to the interface. Table 13 shows the game dedication questionnaire results for this iteration. Table 14 and Table 15 show the results of the first round of user tests for the number of clicks and time to complete the tasks respectively. To review the individual tasks please refer to Table 12.

Table 13 – Game dedication (GD) questionnaire results for each User who tested the interface on the first iteration. For more information on the analysis check Figure 75

Week 1	User1	User2	User3	User4	User5	User6	User7	User8	**Average**
GD %	62,6	93,6	86	81,6	76,2	75,2	83,2	57,2	76,95

Table 14 – Number of Clicks per task in the first iteration. The first row represents the task's number, the second row represents the number of minimum clicks needed to perform the task and the third row represents the median number of clicks that users performed to complete each task

Task Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Number of minimum clicks	8	1	2	3	9	2	17	5	3	15	3	2	4	2
Median number of clicks performed	8.5	1.0	2.0	3.5	10.0	2.0	19.5	4.0	2.0	16.5	3.0	1.5	6.0	2.0

Table 15 - Time taken by users to complete each task (seconds) in the first iteration. The first row represents the task number, the second row represents the minimum time required to perform the task and the third row represents the median time users took to perform each task

Task number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Minimum time to perform a task	9	2	3	2	8	3	12	5	3	11	4	2	5	2
Median time to perform the task	34	8	11	7	23	8	47	8	7	31	6	11	26	5

Despite having high game dedication scores, users still took a long time when performing each task. This lead to the conclusion that parts of the interface were not intuitive. The feedback provided was extremely valuable and led to improvements of the interface on every task.

5.6.3 ANALYSIS AND IMPROVEMENTS

To create presets, the player needed to choose an ability for each of the four ability slots. At the time, the different polarities were only represented by their icons. When testing, players did not know how each ability would fit into each polarity slot and this lead to ambiguity. As the game development progressed, polarities are now also represented by colors and this was also adopted on the preset creation process.

For each ability, a colored border was added with the same color as their polarity. The polarity's images and their respective colors were added to each ability slot Figure 77.

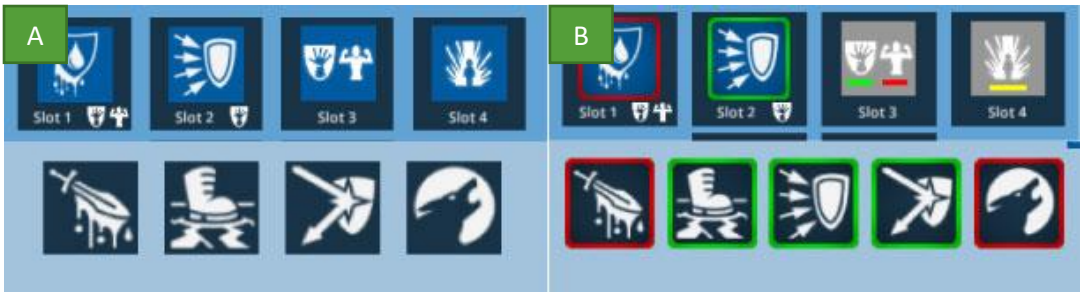


Figure 77 – Evolution of the Polarity Colors GUI– Before (A) and After (B).

When players hovered over any preset item, they would not receive any feedback from this action or what kind of tasks they could perform once there. To improve feedback, the unselected slots are now greyed out. If users mouse over these, they will become blue as if they were selected. Some of the testers did not know that they should name their preset which meant that the input field should be

highlighted further. To improve this, a label was also added near the input field. Every time a preset item is filled, it turns blue to show the difference between the items missing and the ones already chosen Figure 78.

Figure 78 – Evolution of the GUI for creating a new preset– Before (A) and After (B).

When a preset was saved with errors, a generic error message would appear informing the player that something was missing. Players would immediately dismiss it and just press ok to try to save it again. Now, after pressing ok there is a red border around the part of the preset that is missing, to inform the user of what is wrong (Figure 79).

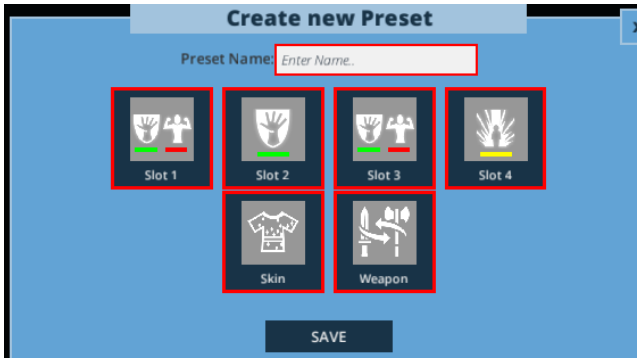


Figure 79 – Preset error notification.

It is expected that even if players do not read the error, they will now be able to know what is missing to complete the new preset.

To activate a preset, the player needed to click it and a blue background would appear, showing them that it was activated. The preset would also have the same behavior when the mouse pointer hovered over it. It took some users a considerable amount time between creating the preset and understanding how to activate it Table 16.

Table 16 - Time it took to activate the preset after creating it on task 10

User 1 time	User 2 time	User 3 time	User 4 time	User 5time	User 6 time	User 7 time	User 8 time
03	03	03	10	02	03	02	06

The feedback gathered, was that there should be some kind of checkbox or way to toggle and activate/deactivate the preset. There is now a way to toggle each preset that shows if it is active or not. The mouse hover continues to exist, but only to assist the player while scrolling through their preset list (Figure 80).

Figure 80 – Evolution of the GUI to activate Presets – Before (A) and After (B).

It is expected that during the next test phase, players will instantly know how to activate their presets.

Some of the players clicked the money bag near the account information when asked to go to the shop (Figure 81). This was taken into account and now there is a link there that will lead the player to the same place as the Shop menu button.

Figure 81 – Account's avatar with username and current gold (money bag has link to shop now).

Players knew how to go to the shop, but when asked to go back to their team, none of them knew what to do. They tried to click everywhere on the screen without knowing that the 'Play' button would lead them to where they wanted to go. This in turn, led to a large number of excessive clicks during this task (Table 17).

Table 17 - Number of clicks that each user performed that were not meaningful (required) to complete task 12

User 1 N Clicks	User 2 N Clicks	User 3 N Clicks	User 4 N Clicks	User 5N Clicks	User 6 N Clicks	User 7 N Clicks	User 8N Clicks
0	3	2	32	17	34	0	0

This behavior was not expected, since many MOBAs have this type of menu navigation and that is the reason behind this implementation. A series of interface problems rose from this. 'Play' was not the most suitable name to lead the player to their team selection and, the shop had no button to go back. Since in each main menu, there is a button on the bottom of the screen that took the player somewhere in the game, it was decided that on the shop there would be one as well, taking the player to the home menu (Figure 82).

Figure 82 – Evolution of the GUI to go back to the Team from the Shop – Before (A) and After (B).

It is expected that players will now know how to go back to their team from the Shop screen.

When asked to change their hero's element, many players went to their profile and tried to find a way to change it there. At that time, the only way to change it was to go to the team and choose the hero element there. Players provided some feedback about placing the hero's element in the hero selection screen or in the profile. Taking this into account, an element choice menu was added to each hero on their information page of the profile (Figure 83).

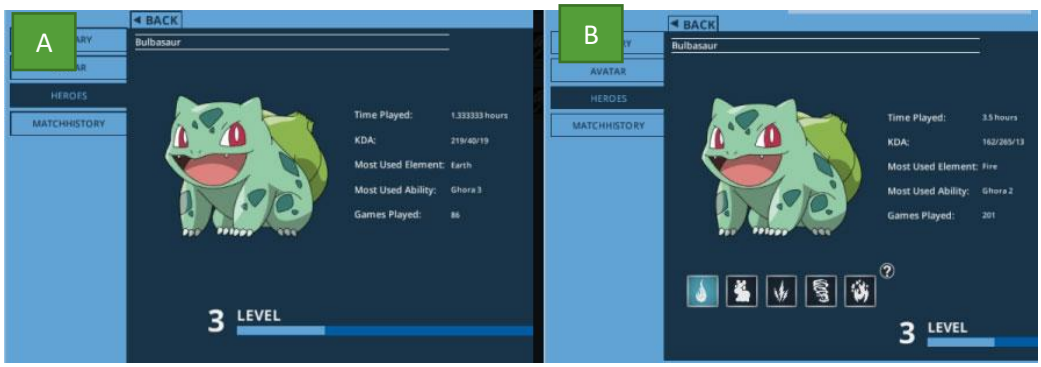


Figure 83 – Evolution of the hero's profile screen – hero's profile before (A); Hero's new profile screen (B).

As mentioned previously, the 'Play' button was misleading the players into thinking that if pressed, they would immediately go play a game instead of going to their team selection. The menu name was renamed to 'Home', leading the player to his team selection page (Figure 84).

Figure 84 - Relabeling of the top menu button, Before - Play (A) and After – Home (B).

In this test phase, some users took some time to find the settings button. They would look at the four corners of the screen trying to find the button, looking first to the top right corner and then, looking counter clockwise to the other corners. Some users even missed the button that was near the chat on the bottom right corner (Table 18).

Table 18 - Time each player took to find the settings button after clicking the start tutorial button on task 1 (seconds)

User 1 time	User 2 time	User 3 time	User 4 time	User 5time	User 6 time	User 7 time	User 8 time
16	01	09	11	13	39	04	02

Some players asked for the button to be moved to the top right corner and made bigger. After some consideration, it was decided that the button would not be moved to the top right corner, since it might mislead players into thinking that it was a button for account settings since, the account information/profile is already on the top right corner. The button was enlarged and moved to the top left corner along with the helper button as, everyone that tested the interface said they knew where the helper button was because they immediately saw it when they found the settings button (Figure 85).

Figure 85 - Evolution of the position of the Game Settings button on the GUI – Before (A) and After (B).

It is expected that in the next testing phase, players will find the settings button much faster.

While performing task 3, users took some time to find where the hero abilities were. Some of them were misled and believed that the polarity sequence on the information tab were the hero abilities (Table 19).

Table 19 – Time it took to find the abilities button on Task 3 (in seconds)

User 1 time	User 2 time	User 3 time	User 4 time	User 5time	User 6 time	User 7 time	User 8 time
09	05	06	16	23	03	24	08

There were a few of problems here that needed to be addressed. First, if players thought that polarities were abilities, either these concepts were not being properly explained or maybe, since it is a new concept, players did not yet fully understand it. Another possibility is that the polarity information screen was occupying so much space that it led the user to think that they were the abilities. Some users also said that the buttons should be on the top or bottom of that left screen and not so close to the border since, they would easily miss them while searching for the abilities button (Figure 86).

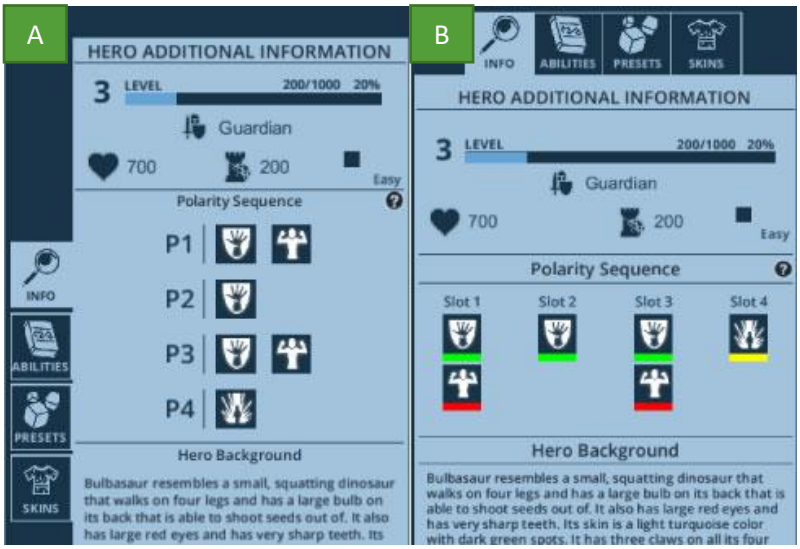


Figure 86 - Evolution of the Hero information screen – Before (A) and After (B).

All the buttons are now placed at the top of the screen and the polarity sequence is occupying less space. This makes it appear less important but still relevant. It is expected that players will be faster to find their hero abilities but also, that they will not get confused between the hero's polarity sequence and its abilities.

5.6.4 RESULTS - SECOND ITERATION

While in the first iteration nearly all the users were ultra-hardcore players, here the users are still hardcore but almost moderate players (Table 20). On the second iteration, players completed each task with more clicks than on the first iteration (Table 21) but, in some tasks they took less time (

Table 22). The main factor that contributed to these results was the level of game dedication these users had. Despite being accurate, these were not very helpful since most users got a high score from their dedication to other genres of games. Another reason for this was that the users were more interested in reading about the hero's abilities than in performing the task as fast as they could.

Table 20 - Game dedication (GD) questionnaire results for each User who tested the interface on the second iteration. For more information on the analysis check Figure 75

Week 2	User1	User2	User3	User4	User5	User6	User7	User8	
GD %	49	59,8	68,2	67,4	87,2	56	65,4	75,2	66,025

Table 21 - Number of Clicks per task in the second iteration. The first row represents the task number, the second row represents the number of minimum clicks needed to perform the task and the third row represents the median number of clicks that users performed to complete each task

Task Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Number of Minimum Clicks	8	1	2	3	9	2	17	5	3	15	3	2	4	2
Median number of clicks performed	9	1	2	3	10.5	2	19	4	3	16	3	1.5	10	2

Table 22 – Time required to complete each task (minutes : seconds) in the second iteration. The first row represents the task number, the second row represents the minimum time it took to perform the task and the third row represents the median time users took to complete each task

Task number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Minimum time to perform a task	0:09	0:02	0:03	0:02	0:08	0:03	0:12	0:05	0:03	0:11	0:04	0:02	0:05	0:02
Median time to perform the task	0:28	0:10	0:06	0:06	0:25	0:08	1:04	0:14	0:07	0:31	0:05	0:05	0:28	0:10

In tasks 2, 4, 5, 8, 9, 11, 12 and 14 the GUI did not change from the first iteration to the second. Any changes to the number of wrong clicks or the time it took to complete these tasks cannot be used to draw any conclusions. In task 1, task 3 and task 6, between the two iterations, the settings button, the hero's abilities button and the help button had their positions changed. It took less time for the users to complete all tasks on the second iteration.

In task 7, between the two iterations, both the preset error feedback and the connection between abilities and they polarities were improved. Regardless of these changes, it took more time to perform this task. It is believed that this happened for two different reasons. First, players still need help to know what to do which can be solved by adding a tooltip to each polarity slot, that explains to the player that they should click the button to choose an ability. Players were also interested in reading what each ability did which can be confirmed in the recorded videos. In task 10, along with the changes mentioned above, the preset activation method was also changed but the performance remained the same in both iterations. Some players tried to activate the preset by clicking on it. Perchance it is better to have both options, clicking anywhere on the preset or on the small lamp on the side of the preset to activate it.

In task 13 every tester knew exactly how to leave the shop. This was no longer a problem. Almost every one used the 'Done' button at the bottom of the screen. Unfortunately, the second part of the task did not run so well. Two out of the eight testers used the profile to change their hero's default element. They went through the team selection page and did not know or realize that could change the elements there. Although it took more time for the testers to perform the task, there were significantly less wrong clicks. It lead to the conclusion that the elements on the 'team selection' were not drawing enough attention. A title was added to each hero's element choice to show that this is where players have to change the elements (Figure 87).

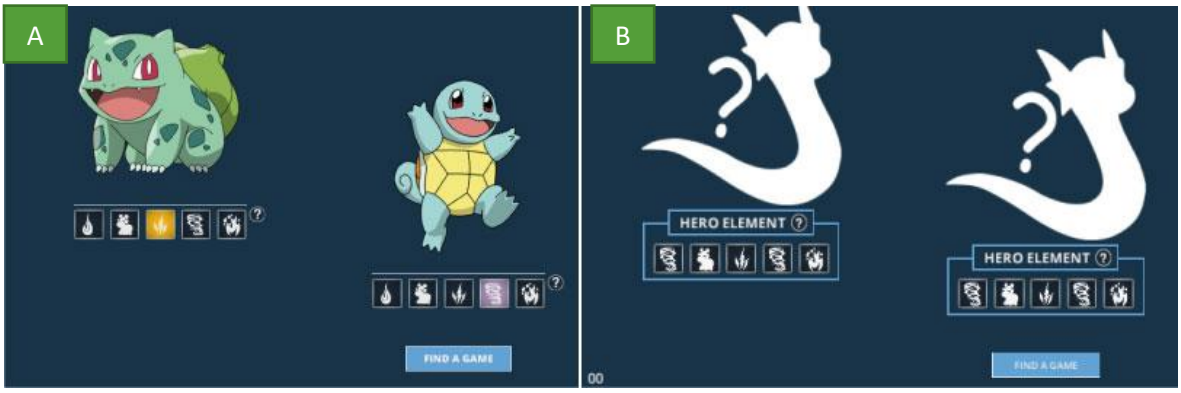


Figure 87 – Evolution of the game's team screen – Before (A) and After (B).

User tests were vital to the development of the interface and greatly helped to improve it. Even users that were not familiar with this type of games, were capable of providing valuable feedback that translated into interface usability improvements. In the future, further user tests should be conducted with players experienced in this genre to fully validate the layout and navigation of the interface.

6 3D ASSETS CREATION PIPELINE – FROM CONCEPT ART TO UNITY3D

Before starting the creation of any 3D asset, a production pipeline was established based on the pipelines used in the game industry. This pipeline was followed during the production of all 3D assets of the DAP (Figure 88).

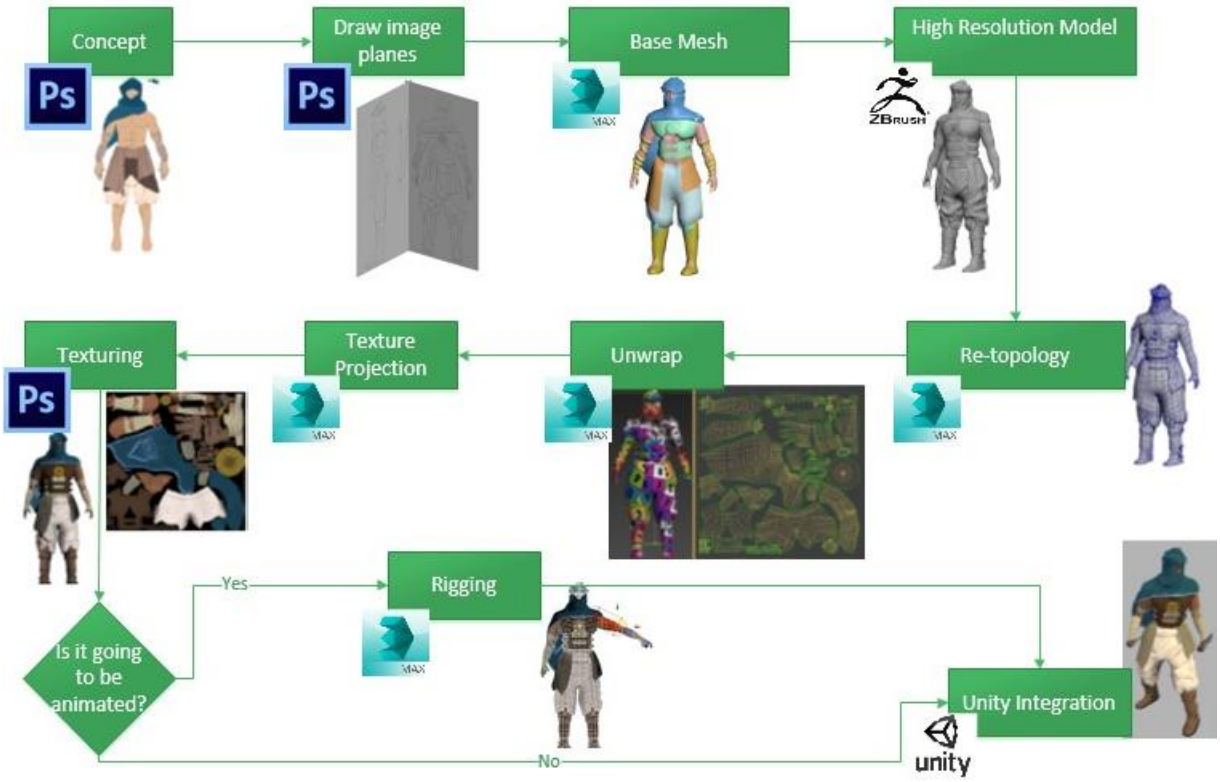


Figure 88 – 3D Asset creation pipeline.

The process to create any asset starts with its concept. Therefore, for each model created there was first a concept design from various inspirations. From pencil to Photoshop, each concept was developed while keeping in mind what material, type of surface and color each piece of the model had to be made of. Once the concept was complete (check section 4 for more information on the concept development), front, back and side views were created to serve as guidelines during the creation of the 3D models (Figure 89).

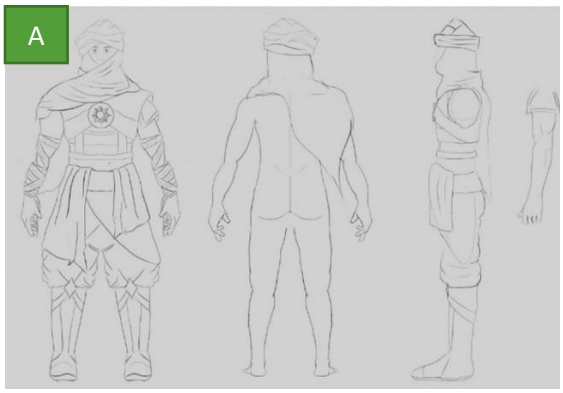


Figure 89 – (A) Malik Image Planes in Photoshop.

Considering that, more than half of the process was similar regardless of the type of asset being modeled (character or environment/weapon asset), the differences in each step will be described in detail in the corresponding process' sub-section.

6.1 CREATING THE BASE MESHES

For each game asset, a base mesh was created in 3DsMax. This mesh would provide the basic topology for the high polygon model and the only precaution to take at this point was to make sure that there were no triangles or Ngons. As previously mentioned in the state of the art, this makes it possible to fully divide the mesh for the next steps of the pipeline. To ensure that there were no triangles of Ngons on the models, a 3DsMax plugin named Non-quad selector was used [139]. After choosing a mesh and using the option to count the number of Ngons, this plugin would then count and highlight them in the mesh.

Since there would be four male and two female characters, a basic body mesh was first created. This mesh was important to define the basic anatomy such as: arm length, base bone structure, the character's height and basic facial features such as the position of the nose, eyes and mouth. It was important to have all of these base meshes in a T-pose or with the arms open in a 45 degree angle so that in the future sections the model was easier to work with. These body meshes saved time when creating each character since there was no need to create six different bodies, instead there was only one male and one female base (Figure 90). In the following steps, when higher detail was required, unique features could be added to each character, giving them different traits and personalities. Both body meshes were modeled from a cube using the front, back and side concepts as guides. While modeling, a Symmetry modifier was used. This modifier allows the modeler to create only half of the body while 3DsMax creates the other half as a mirror image, considerably improving the workflow. Once both base meshes were completed, the development of each character began. This process was very similar between the six characters, with only small changes to the way some parts were modeled or to the base mesh used (male/female).

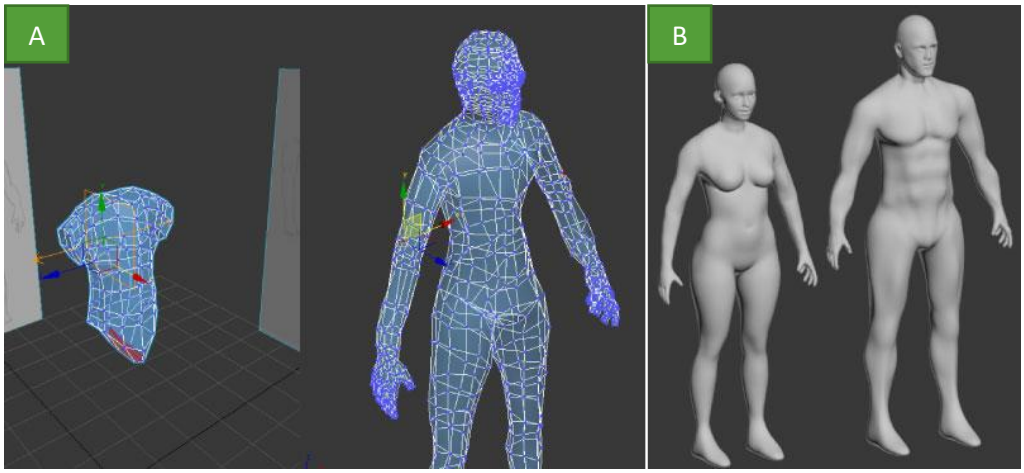


Figure 90 – (A) Starting point of the base meshe; (B) Final female and male base meshes.

After having a basic body mesh, the characters' clothes were added. 3DsMax allows dropping polygons over a surface to create a new mesh. This process is called edge modeling, which in 3DsMax can be achieved by using the freeform modeling tools. Most of the character's clothes were created using

this process with the body mesh as the surface (Figure 91 - A). For example, Malik's chest armor and arm bandages were created by using this process while, his pants, turban and boots were created from cylinders with an edit poly modifier (Figure 91 - B). When all the base clothes were created, characters were at the same stage as the other game assets in 3DsMax and could now be exported as .obj files with the Zbrush preset. Later they would be imported into Zbrush to add further detail.

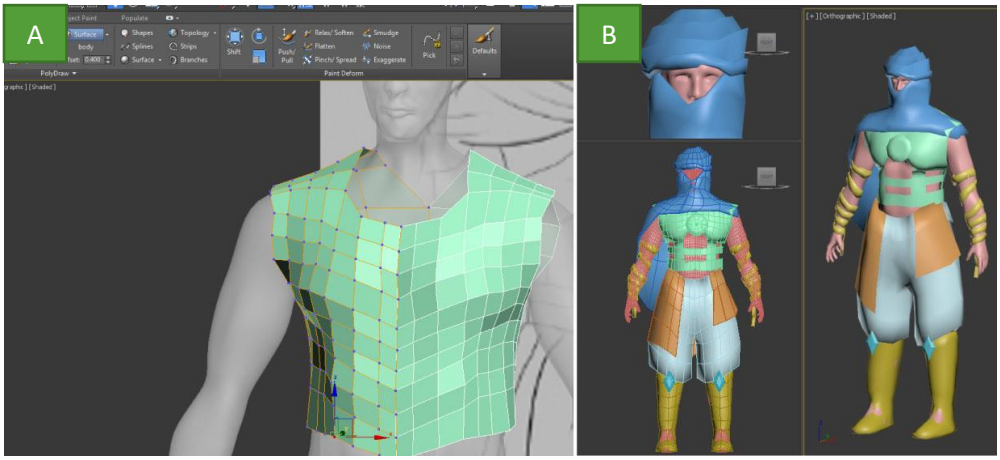


Figure 91 – (A) Malik's torso created using edge modeling with the base mesh as its surface; (B) Malik base mesh before going into Zbrush.

6.2 FROM THE BASIC MODEL TO THE HIGH POLYGON MODEL

This section explains in detail how the Malik character model was created since, it was the most complex to model. The evolution of all other assets can be seen on the annexes (9.5).

To create the high polygon models, Zbrush 4R7 was used. Before going into detail about sculpting in it, there are some important base brushes and tools to be considered. Firstly, the Clay brush will make it possible to add/remove volume to a mesh in small amounts at a time. On the other hand, the ClayBuild brush will add/remove volume as long as it is pressed which, is very useful when creating muscles if there is no brush alpha selected. However, when adding smaller details like carvings, the DamStandard or the Slash3 brush should be used. To redefine facial features the Standard brush is a good choice. Using all these brushes while using the Smooth brush will yield good results. Yet, the most useful brushes are the move and move elastic brushes as they allow the modeler to move the mesh and extend it. These brushes combined with the ZRemesher tool will make it possible stretch any mesh and it will remain with a good topology for further division. Another valuable tool is the masking brush that allows panting (masking) mesh zones which will then ignore any sculpting. This is very useful when adding small details but only in certain parts of the mesh. While creating the assets for the DAP, all meshes started with a single division and as additional detail was required, additional divisions were added. A mesh was never divided if, the specific detail being added could be added without increasing the polygon count as, additional divisions would require more computing resources for Zbrush.

Since it was the first character to be created, the male base mesh had no muscles or facial features. This first steps were to define his muscles and base bone structure such as the cheek bones and forehead size. As mentioned previously, the ClayBuild brush is a good tool to create muscles combined with the Smooth brush (Figure 92 – A and D). Another important detail was his fingernails which were all created based on the approach of Willyam Bradberry [140]. This technique consists in masking each

fingernail, creating a polygroup for each mask and then moving each poly-group slightly into the finger creating the illusion of a fingernail (Figure 92 – B and C).

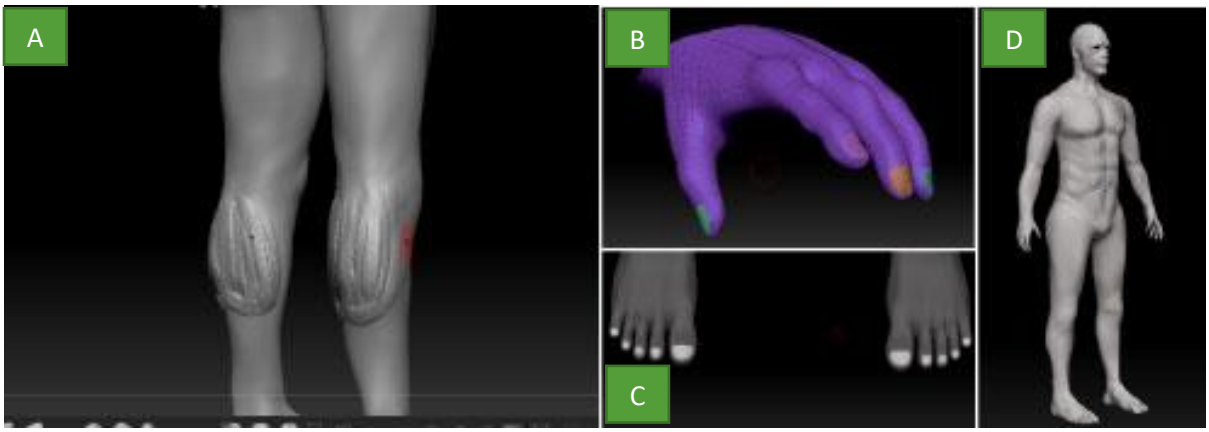


Figure 92 – (A) Leg muscles being created in Zbrush; (B) Finger nails; (C) Toe nails; (D) Final mesh with muscles.

Once satisfied with the details of the base mesh, it was time to add detail to Malik's clothes. Every mesh detailed in Zbrush first passed through the Zremesher tool to create equally sized polygon models and simplify the sculpting process. Malik's turban was the first thing to be modeled. Using the Clay and ClayBuild brushes together with the Move brush, folds were created to give it an organic look. The cloak was the next object to be detailed. In the beginning, it was divided in two separate meshes, the part covered his shoulders and face and the rest of the cloak. Unfortunately, when two separate meshes are modeled together as if they were a single mesh in Zbrush, the vertices will not be welded, making it difficult to keep the two meshes together (Figure 93 - A). Considering this, another single mesh base cloak was created in 3DsMax. The details to the cloak, pants and the pieces of cloth hanging from Malik's pants were added by creating many folds that, produced a feeling of movement and realism. These folds were all created using two brushes, the Standard brush to add the folds' volume and then the Pinch brush. This brush is used to move all the vertices inside the brush's area to its center (Figure 93 - B).

To create the boots and chest details, the inflate deformation option was used. This is a very useful tool when used together with the masking brush. For example, by masking part of his boot and then inverting the mask, only that part of the boot will be editable. Then, using the inflate option, the editable part of the mesh will inflate according to the chosen axis. This is similar to extrude by local normals in 3DsMax and Maya (Figure 93 – C).

Figure 93 – (A) Malik's cape wrong mesh division; (B) Malik's cape folds; (C) Malik's boot inflate mask.

Zbrush has a feature which allows a user to create different meshes and add them as parts of a brush. These are known as Insert Mesh (IM's) or Insert Multi Mesh brushes (IMM's). There are numerous open source libraries on the internet [141] and it is also possible create one very easily in the brush tool window. All the fine details of the six characters were created by using IMM's from a library or by creating a new one. For example, Malik's chest and boots have leather straps on the sides (Figure 94 – A), his daggers have gems on their grips. Lipp's hair braids were also done with an IMM. The IMM's are very easy to use. When drawing on a surface, it immediately starts adding a seamless mesh as the line is drawn (Figure 94 - B). Another example of an IM used was the border stiches of Malik's pants. Using only an alpha image of a stich with the standard brush it was possible draw on the surface of his pants and create a realistic cloth seam (Figure 94 - C).

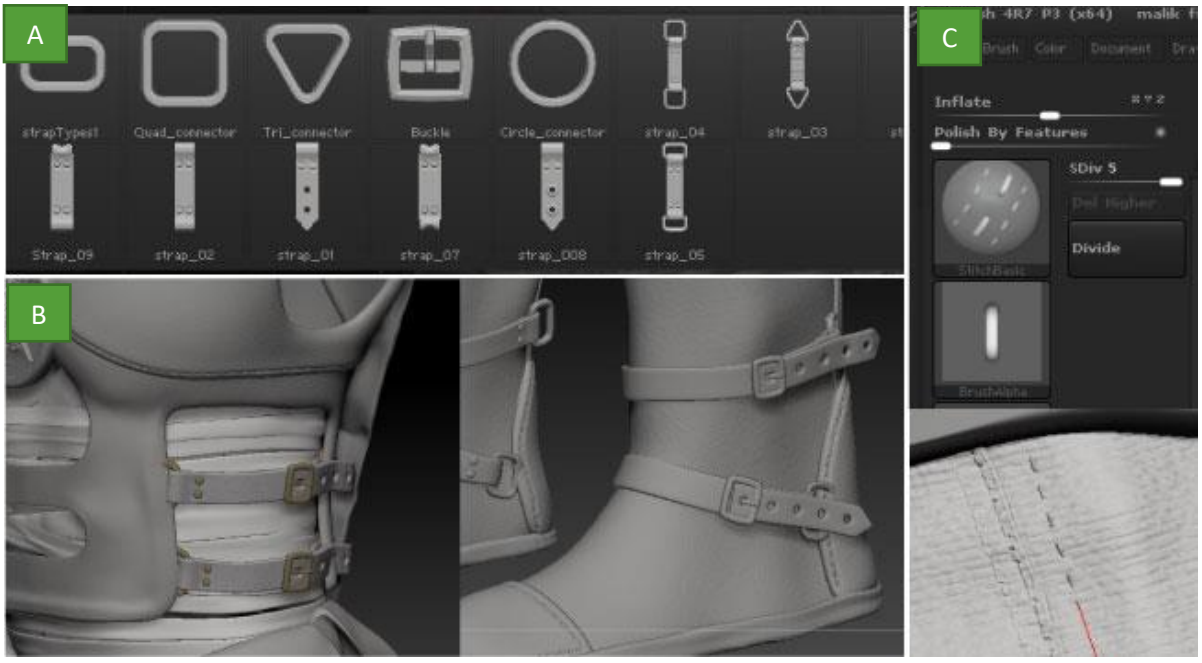


Figure 94 – (A) Zbrush IMM brush for straps; (B) Malik's torso and boot straps created in Zbrush; (C) Zbrush Stich brush and final result.

To create Malik's arms and torso bandages, a technique based on an approach by eFerrari [142] was used. This technique consists in using the SliceCurve brush to slice a cylinder splitting it in different poly-groups. On this case an extract of his arm and torso were used as the base. When satisfied with the number of slices, a group split is performed which will create a sub-tool for each group. As a last step, the thickness is increased using panel loops and many small cylinders that resemble bandages or straps have been created. After performing this technique two or three times with different sliced groups, folds and smaller details were added to create a realist look (Figure 95 - A).

To create Malik's detailed medallion, a similar process to Martins' [143] engraving brush technique was used. In this case, only the first three steps were used. Using the Standard brush and an alpha image together with the DragRect stroke option on, the alpha image was carved on the surface. It is important to note that to prevent the engraving from passing through the other side, the 'back face mask' option had to be turned off in the brushes pallet (Figure 95 - B).

To add the final detail to Malik's clothes such as the linen or leather textures and skin imperfections the NoiseMaker tool was used. This tool uses a seamless texture as a pattern as if it was sculpted on the mesh. Firstly, for each mesh that would have a noise texture, an automatic UV map was created. Then, the mesh was divided several times depending on the level of detail being added with more detail requiring more divisions. When pleased with the noise scale and irregularities, these were applied to the mesh so they became permanent as part of the mesh's detail (Figure 95 - C).

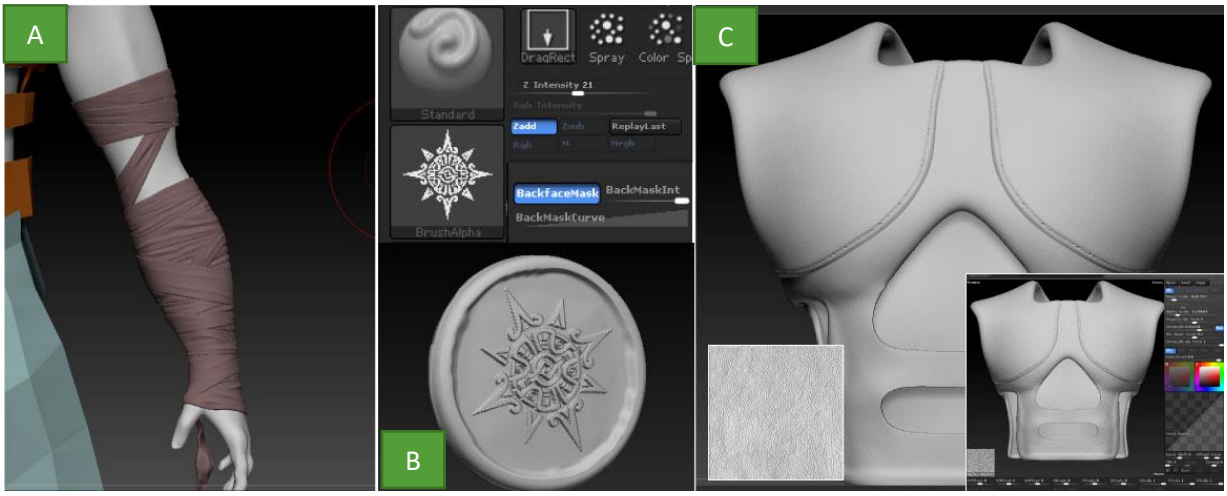


Figure 95 – (A) Malik's bandages sculpted in Zbrush; (B) Malik's medallion brush setup and final result; (C) Malik's torso armor noise texture and final model.

Different character's details and other assets in the DAP used the same techniques explained both above and in the following section.

6.2.1 ADDITIONAL TECHNIQUES USED IN ZBRUSH

In addition to the methods mentioned before, there are others that are extremely useful and were used as other models were created.

Some parts of the characters' clothes were created from extracting a new mesh from their original body, such as Kulla's hoodie and vest (Figure 96 - A). This was done by masking what was going to be extracted, inverting the mask and then extracting it with a thickness of 0. Then, to add thickness the

panel loops option was used with a thickness of 1. This created an interior and exterior mesh where the extra detail could be added. As mentioned before, clothe stiches are a good way to add additional detail to the character's clothes and a bit of realism as well. This, combined with the inflate technique, can be used to create clothe borders as shown in Figure 96 - B.

To create Ghora's chainmail, the Micromesh tool was used. With this tool, it is possible to replace each polygon of a mesh with equally sized polygons with another seamless mesh. The seamless mesh used on this case was one of the chainmail's links (Figure 96 – C) which, had been previously modeled in 3DsMax. If the polygons are well distributed inside the mesh, the combination of all the links will form a pattern giving the mesh a realistic feeling and look.

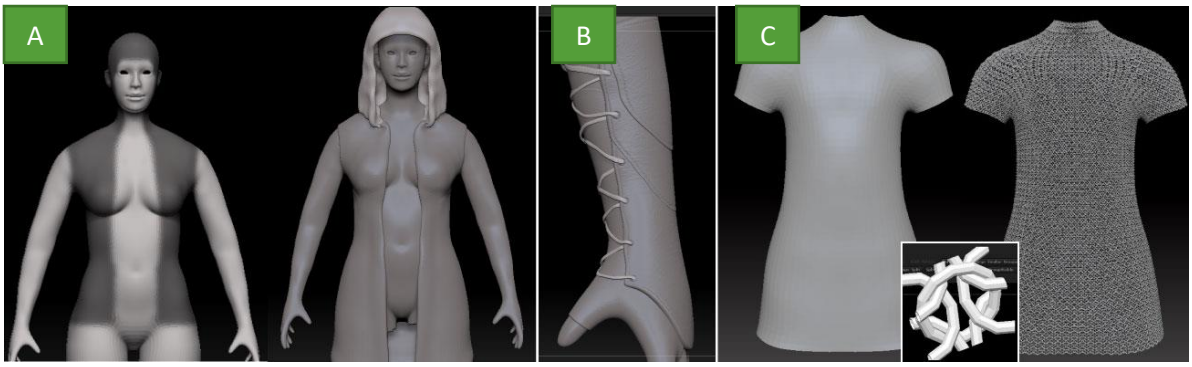


Figure 96 – (A) Creating Kulla's hoodie from a mask; (B) realistic detail on Kulla's bracer; (C) Ghora's chainmail using the Micromesh feature.

Three different techniques were used to create hair/fur. For long sculpted hair, a brush was used from the Zbrush's brush library [144]. To create an illusion of real hair, the Pinch brush was used in the bottom tip of the hair and the DamStandard brush was then used to create the illusion of more hair (Figure 97- A). To create the men's beard and the fur on Loper's lion head, a mixture of the ClayBuild and DamStandard brushes was used. The ClayBuild was used to add volume and shape creating some 'V' shapes following the direction of the beard strands. Then, the shaped parts were filled with more volume and the DamStandard brush was used to add some depth. Volume and detail kept being added until achieving the desired result (Figure 97 - B). To create realistic hair, Zbrush has a tool named Fibermesh which grows hair on the selected surface after choosing the hair mesh. It is also possible to comb the hair or even create curls using the groom brush set. This tool is very useful to create realistic fur for characters and animals, since it can then be used with Maya's hair tools to render high definition models. In this case, fibermesh was used for both the lion's fur and whiskers on Loper (Figure 97 - C).

Figure 97 – (A) IMM Brush used together with the pinch brush created Lipp's hair; (B) Dam_Standard brush was used to create loper's beard and his lion fur; (C) Evolution of loper's lion head and final fibermesh render.

Since the game's world was ancient and some parts were damaged, cracks, cuts and erosion was added to the world assets. To create cracks, a set of TerraCracks brushes were used [144]. These used alpha maps like the technique used to engrave detailed images. With the base model sculpted, cracks were then added as seen fit, an example can be found in the game's towers (Figure 98 - ATo create erosion on rocks, like the ones on the hand monument, the noisemaker technique mentioned before was used. However, this time its plugin (noisePlug) was also used, using the granite option with a small noise scale (Figure 98 - B). Furthermore, to add deformation to the game's small rocks, the polish brush was used. This brush can be used in rounded edges to polish them, creating a feeling of old age and creating hard edges.

Some weapons also needed to look used and worn out. Ghora's shield is a good example since, it has many scratches and cuts on and around it. The scratches were made with the same technique and the same brushes used as to make cracks. On the other hand, the cuts were made using the Clip Curve brush. After choosing where to start cutting a 'cut out' shape is created and then it is possible to cut the mesh (Figure 98 - C).

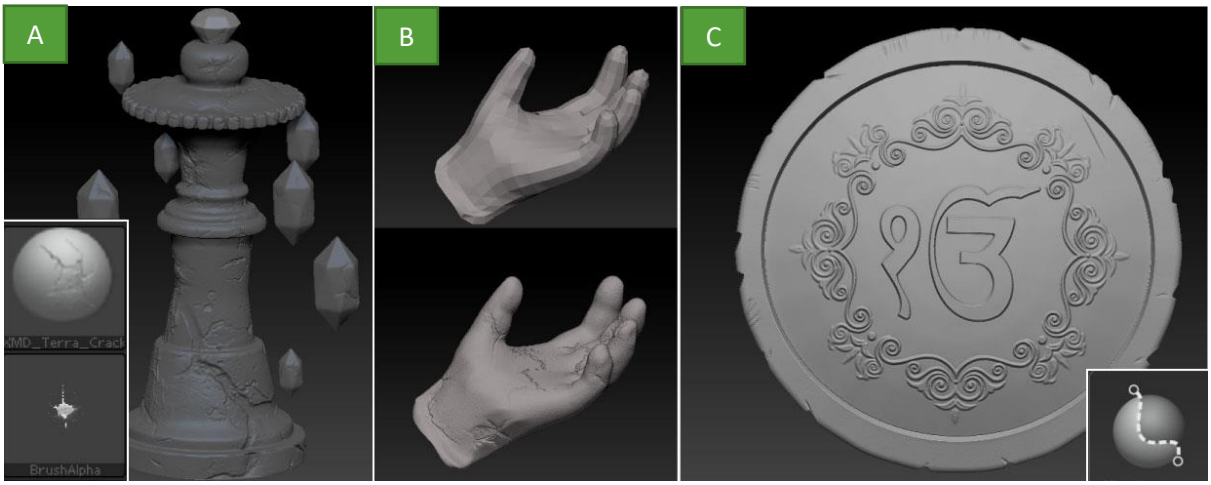


Figure 98 – (A) Tower model with cracks using specific Crack brushes; (B) Low and high poly models of the stone hand; (C) Ghora's shield with cuts created by using the Clip Curve brush.

Although there are still many tools unexplored in Zbrush, all the techniques explained above are sufficient to replicate all the characters, weapons and world assets created in the DAP.

6.3 RETOPOLOGY - FROM THE HIGH TO THE LOW POLYGON MODEL

Now that the high polygon model was created and finalized, it was time to make it a viable game asset. To achieve this, first every model's sub-tool needed to be decimated in Zbrush to reduce the number of polygons without losing detail. A 20% decimation reduction parameter was used as long as the sub-tool in question did not have any noise on it. In that case, the parameter had to be higher, around 50% so it would not lose the finer details. After this, all the sub-tools had to be exported as .obj files. When importing them back to 3DsMax, where the re-topology process would take place, it was important to optimize each object using the Optimize Modifier. These two steps are performed to make sure that the high detailed mesh will not crash 3DsMax when imported since, this program does not handle millions of polygons well.

Now, depending on the type of model being created, it is possible to use the previously created base meshes as a low polygon mesh and save time on the re-topology process or to create a new mesh from scratch. Examples of models that can use the first option are objects that had independent meshes both as a base mesh and as a high definition one but also shared a similar silhouette such as: rocks, small buildings and statues. In these cases, there is no need for additional steps and the re-topology process is concluded.

However, if the model had more than one mesh then there was a last step before starting the re-topology process, to 'explode' the high polygon model. This technique consists in creating a keyframe with all the objects selected on frame 0. Then, on frame 1, objects that make sense to be grouped are positioned around the screen at different X positions (Figure 99 – A). Everything that will become a single mesh with welded vertices should be positioned together for example: pants and belt, a chunk of rocks or a character's face and mask. If the model being created is going to be animated, then it is also important to group objects that need to have their vertices welded when doing the re-topology. This will make it so that when bending the model, the two objects will become one and bend together instead of individually with their vertices in different positions. Once the 'explosion' process was

complete, a new mesh surface was created based on the high polygon model. This process is known as re-topology.

The edge modeling approach was used to create all the re-topologized models. With an offset between 0.05 and 0.1 and the high polygon model as the edge modeling's surface it was possible to start the process.

Two important things to consider when doing re-topology: make sure that the low polygon model has the same silhouette as the high one and add more polygons if additional detail is required. It is not necessary to create detail for every fold and crack but, it is important to create the support for extra polygons when there will be a big difference in volume between the low and high ploy detailed models. For example, Malik's cloak has many folds but, only the most striking ones have some support from the low polygon model (Figure 99 – B). However, the game's towers were filled with cracks and small details but, the low polygon model did not support any of these. Yet, the towers still had a very good look in the end (Figure 99 – C).

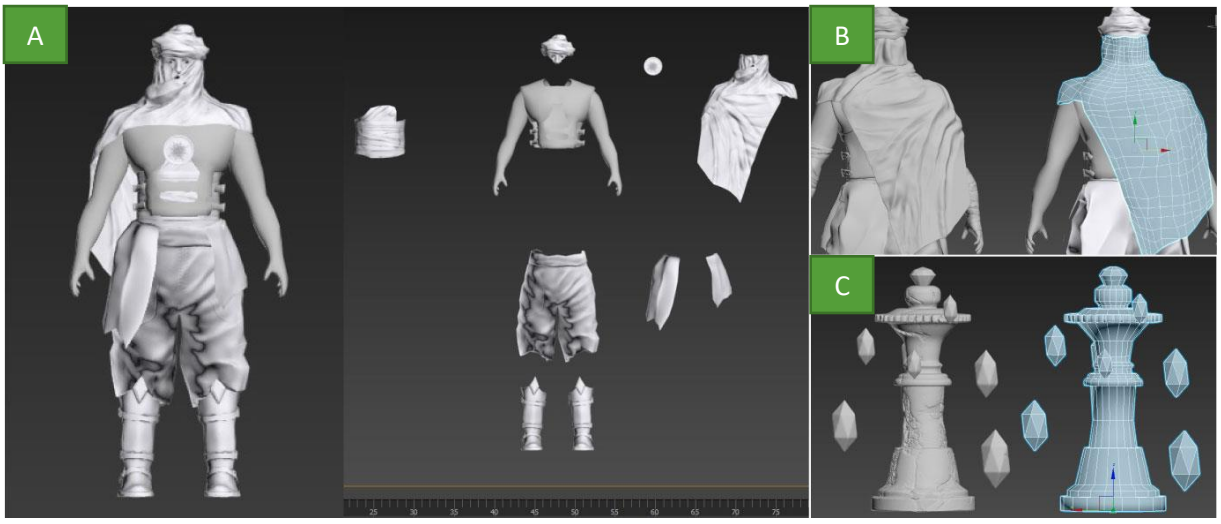


Figure 99 – (A) Malik's model exploded; (B) Malik's cloak low poly model supporting high polygon folds; (C) Tower's low poly model not supporting high polygon model's detail.

An additional concern was if the model was going to be rigged afterwards. If the model would only have its position or rotation animated the topology was less critical since, on this kind of assets the base mesh is usually used as the low-resolution model. The most important aspect is for them to have the same silhouette.

However, if the model was going to be rigged, such as a character, extra considerations while creating its topology were required. The face should have enough polygons to support the high poly model bone structure as shown in Figure 100 - A. The face was modeled by starting at the eyes, creating a loop around them and trying to keep the same number of polygons on the top and bottom (1). After approximately three loops, the eyes were complete. Then, a bigger loop was created starting on top of the eyebrows and going around the eyes and passing through the nose, similar to a carnival mask (2). This was followed by drawing another loop from the top of the nose around the cheekbones to the bottom of the chin, adding small loops inside until reaching the mouth (3). Body joints like elbows, the back of the knee, fingers, wrists and arm pits should be done with extra loops. A loop was added for each side on joints that would be bending and a single loop on the inside of the bent zone as shown

in Figure 100 - B. A better reference of how the base topology of a character was once complete can be seen in Figure 100 - C. Once the final model had achieved a satisfactory level of quality, smoothing groups were set up on all the hard edges, providing additional support for details during the next step.

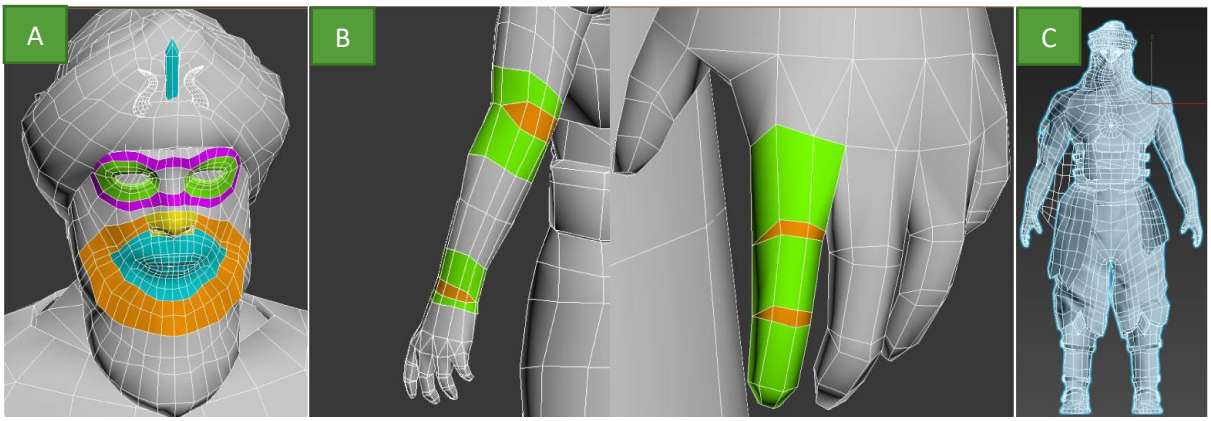


Figure 100 – (A) Ghora's face topology; (B) Extra loops for bending arms and fingers; (C) Malik's body topology.

6.4 UNRAPPING PROCESS – MAKING A PUZZLE

When satisfied with the low-resolution model, the first step was to reset the XForm in 3DsMax, where all the process was performed. This is vital in order to reset any odd scales or rotations in the objects to prevent errors on the unwrapped maps.

After collapsing the stack, the unwrapping process could begin taking into account two vital rules; firstly a seam was always added to a hard edge to avoid having strange black lines in the normal maps when baking textures and secondly, seams should not be added to parts of the model where they will be visible at all times.

Unfortunately, seams are sometimes visible, even if barely, when the viewer is close enough. This was counteracted by adding seams bellow the armpits, between the fingers and when there was a change in material or color.

After dividing the model into several seams, it was time to create the unwrapped 'puzzle'. First a Pelt Map was created for each polygon selection and relaxed by its polygon's angles. As this process was being performed the selections outside the map were spread. Once complete, all the elements were re-scaled and the 'puzzle' began to be assembled (Figure 101 - A). At this point all elements had the same scale but, this was not intended. The goal was for the most important parts of the meshes to have more detail, like character's face, torso or tower's cracks. Therefore, those parts were rescaled to become bigger (Figure 101 - B). By doing this, it was important to keep in mind that as much space as possible should be saved by rotating and filing every single space with parts of the mesh.

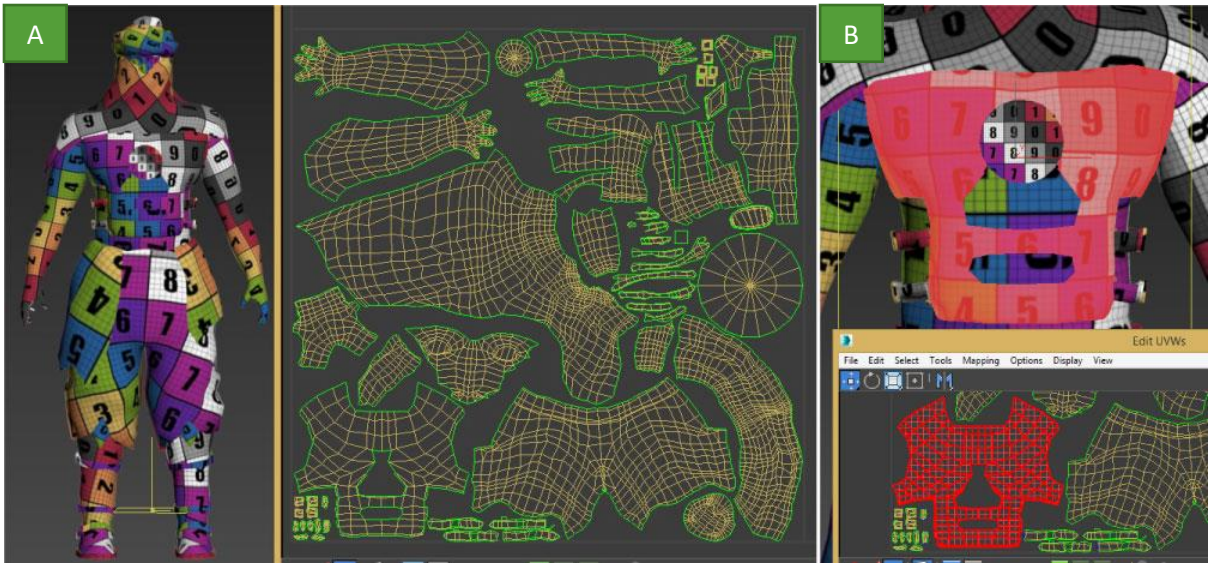


Figure 101 – (A) Malik's unwrap map; (B) Malik's torso armor identified in the unwrap map.

This was a lengthy and arduous process. However, since these maps were going to be used in the next step, to bake the high model details into this model, any mistakes made during the unwrapping process would be costly when texturing the model in the end.

6.5 BAKING TEXTURES - MAKING THE LOW POLYGON MODEL LOOK GREAT

After having the model unwrapped, it was time to create the illusion of detail on the low polygon model. This process was done using 3DsMax. These files were created considering that if additional detail was added to the high polygon model, all that had to be done was to bake new texture maps and the additional detail would added to the low polygon model.

Before starting this process, it was important to paint every high detailed mesh that had different materials with different colors to be able to easily distinguish between them when baking the Diffuse Map. Using the 'Render to Texture' feature in 3DsMax and by enabling the projection mapping feature a Projection Modifier was added to the model and made it possible to choose which objects the detail was being projected from. The Projection cage was reset and pushed accordingly until all of the high polygon model was inside it. Before baking the texture's details some parameters were considered, a padding of 12 was required for baking high resolution textures (2048x2048) and the existing mapping channels were used. Diffuse, Normal and Lighting maps were chosen as output maps.

In order to render the lightning map correctly, a Skylight was added to the scene (its position was not important) and some of the render settings were changed (for more information check Figure 131) Finally, the detailed maps could be rendered. This was done to all the mesh groups that were exploded which, provided all the important details to the low polygon model. Figure 102 shows the differences between the high and low poly meshes.

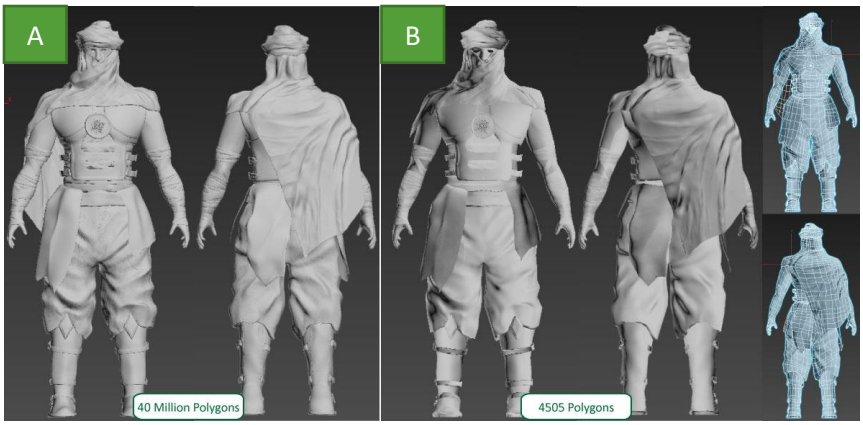


Figure 102 – (A) Malik's high polygon model (40 Million polygons); (B) Malik's low polygon model after adding both baked textures (4505 polygons).

6.6 TEXTURING THE MODEL FOR UNITY

Unity3D has built-in shaders that can be used to render any game asset. One of the best is the Standard Shader which, allows adding different texture slots and parameters to a single material. Furthermore, the Standard Shader has an advanced lightning model called Physically Based Shadding (PBS) which essentially simulates interactions between materials and lights, mimicking reality [145]. As a good practice, only one material should used for each mesh inside Unity, so it was decided that all the textures would be done accordingly to how the Standard Shader works in Unity3D.

Depending on the game asset, several texture maps were created: Diffuse, Specular, Emission and the two maps already baked in the previous section, Normal and Occlusion maps. As for the process used to create the actual textures, it will be briefly explained above for each map.

The Normal map required additional attention, to invert its green channel in Photoshop. The reason for this was that 3DsMax and Unity differ on their axis orientation and when projecting textures in 3DsMax the green channel in the Normal Map (Y axis) was pointing on the opposite direction creating artifacts when adding it to Unity3D's Normal Map slot (Figure 103).

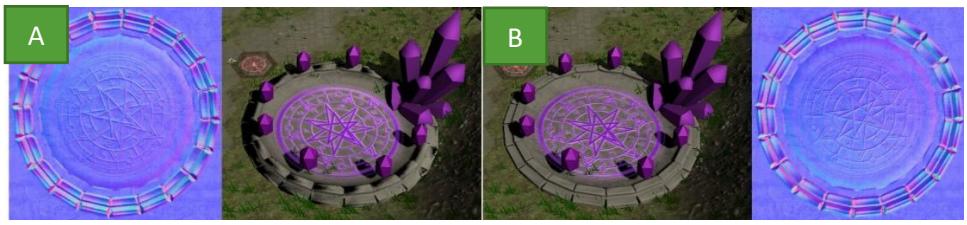


Figure 103 – (A) 3DsMax baked Normal Map creating artifacts in the model; (B) Inverted Normal Map fixed artifacts.

By previously baking the Diffuse Map, it was possible to have the different parts of the mesh organized by different colors. Then, in Photoshop the Select by color range feature was used to create masks for the different parts. To add additional details such as shadows, the occlusion map was used as a multiplied layer on top of all the other layers. Some parts still did not have sufficient detail, such as the character's face texture so, further detail was hand painted. Also, to provide additional details that were sometimes missing from Zbrush's model, such as leather patterns and metal scratches, these were added as an additional overlay layer below the occlusion map layer (Figure 104).

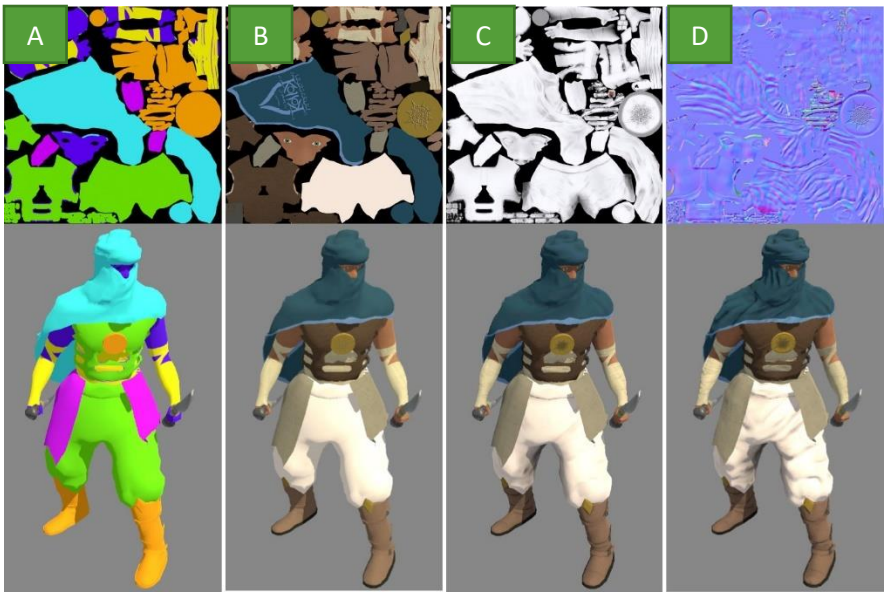


Figure 104 – Model texture evolution - (A) Baked Diffuse Map; (B) Painted Diffuse Map; (C) Occlusion map; (D) Normal Map.

The specular map, was always created after having the Diffuse map since, it was created on top of it. All the parts that would have reflections were selected and a mask was created increasing its curve values along with an alpha channel of it. For all the other parts, a mask was created and the Hue was changed to -180 (Figure 105 - A). In unity3D, to control which parts have reflections and which do not, a smoothness parameter is used which is controlled by the alpha channel of the texture. Since this channel had already been created, all that remained was to define from white to dark grey which parts would shine more (white) and which would not (dark grey) (Figure 105 - B). Then, inside Unity3D the parameters were adjusted accordingly (Figure 105 - D).

The process to create the Emissive map is very similar to the Specular map. The alpha channel was used to determine which places would emit light and which would not (white equals full emission). Then, with a copy of the diffuse map but with more saturated colors, the color map was defined. In the game, this maps is used on stones around each tower, defining its defense zone (Figure 105 - C).

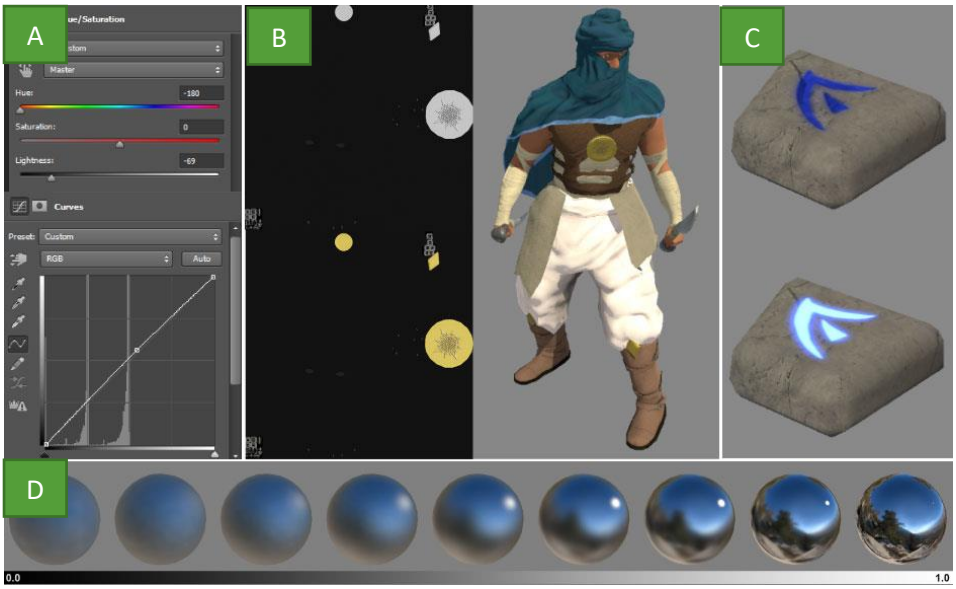


Figure 105 – (A) Options when creating Specular Map on Photoshop; (B) Specular map and texture applied to model; (C) Tower stones without and with the Emissive map applied; (D) Smoothness levels of the Unity standard shader [146].

There many other ways to detail and paint the model's textures as previously mentioned. Any other way could have been used to achieve the same or even better textures to then integrate with Unity3D's Standard Shader.

6.7 RIGGING AND SKINNING PROCESS

This was the lengthiest process in the entire 3D pipeline. Previous projects, provided key insight on what should not be done but, the experience in rigging and skinning a character was still reduced. 3DsMax was the tool of choice for this part of the process. This process was only performed for characters since any other animations in the other game assets were based on physics and did not require any bones. When choosing between biped and CAT for the character's skeleton, the biped was chosen for two reasons: firstly, there was previous experience with it and secondly, while researching about which was better, there were many complains of 3DsMax users about CAT related crashes.

Unity3D has very specific rules about the skeleton [93] that should be used. It should only contain two spine bones, one neck bone, a minimum of fifteen bones [147] and it only supports rotation animations (bones are not allowed to change their local position). On top of this, Unity3D does not support bone twists which, are used to prevent skin deformations on wrists and knees when they are in a forced twist setup.

Unity only supports manually specifying muscle twists. This prevents many animations from working properly since, in real-life humans can twist their wrists. This leads to the loss of information since motion capture files often have this type of information. A last thing to consider when creating the skeleton to work with Unity was the way that the biped's hierarchy had to be structured. The hips have to be the root element of the skeleton's hierarchy, the biped should have a triangulated neck, its pelvis should not be triangulated and as mentioned previously, the second spine join has to be the parent of the neck bone (Figure 106 - A).

To start the rigging process, a biped was created with the previously mentioned settings, five fingers with three joints each and additional bones to serve as weapon placeholders. After positioning the pelvis in the middle of the character's pelvis, the biped's bones were positioned and scaled as close to the character's mesh as possible (Figure 106 - B). Since parts of the character's clothes were to move with physics and, since Unity3D allows choosing which vertices are fixed and which can move, it was decided it would be best to divide the meshes into smaller parts to be able to paint/select the correct vertices. Malik's character, was divided into three different parts: his cloak, the pieces of cloth hanging from his pants and the rest of his body. To all three meshes a Skin modifier was added but, only the bones that made sense were added to the bone list. This modifier automates the skinning process initially and, if the bones were placed properly during the rigging process, each bone will have the weights equally and properly distributed. It was only required to skin half of the model since, the skinning modifier allows mirroring the weights to the other side of the body if the characters are symmetrical.

For a more efficient workflow, a simple animation was created to evaluate which parts of the skinning needed more work and which were already complete. This animation consisted in doing stress tests to the rig, like bending the wrists, knees, shoulders, rotating the spine and opening the legs (Figure 106 - C) in order to find any anomalies and fix them.

For the first character created, a full rigging setup was done using the 3DsMax Skin modifier along with the biped. In an attempt to decrease the time of this lengthy process, Mixamo's auto-rigger combined with its 3Dsmax script was used (Figure 106 - D) [148]. These did not create a perfectly animated model but, they did save time at the start of the process where each bone had to be skinned to the model's mesh. Once the skinning process was complete, it was time to add animations to the biped and incrementally edit and refine the model's motion.

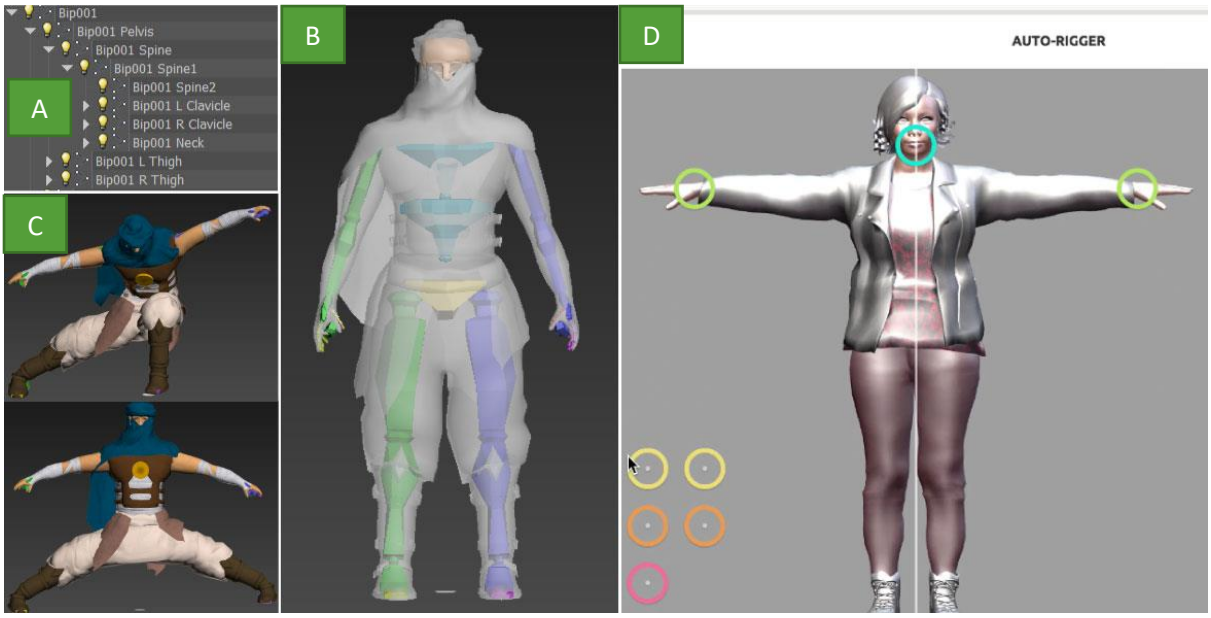


Figure 106 – (A) 3DsMax biped hierarchy; (B) Final biped position before applying the Skin Modifier; (C) Animation stress test to the rigged character; (D) Mixamo's auto-rigger screen [148].

6.8 ANIMATION PIPELINE – FROM 3DS MAX TO UNITY

Both characters and other assets were animated in 3DsMax. Motion Capture animations were used from Mixamo's free library [148] to animate each character.

6.8.1 CHARACTER'S PIPELINE

Mixamo makes it possible to export all the chosen animations with Mecanim rig support, as a .bvh file. All the animations were then processed by a basic biped and converted to .bip files so they could be used on the character's bipeds without losing their size or position. Even though the skinning process was theoretically already completed, sometimes some vertices still needed to be refined. Thus, all the animations belonging to the character were imported one by one and refined.

Any animations that made the character change its position were locked in place (using 3DsMax 'inPlace' biped feature). This would be used later in the game to control the movement speed. Since the characters had accessories on their hands and, some needed to have their hands closed, biped layers were used to add those additional keyframes. This consists in adding another layer of animations on top of the animation. For example, having base animations with their hands opened, an additional layer is added where the hands are closed on the first keyframe, creating an animation that combines the whole base animation with closed hands (Figure 107 - A). Layers were also used to

fix anomalies in the motion capture animation, since the skeleton being used did not have the same number of bones as the one from motion capture.

To improve the workflow, a few of rules were set. The base character's mesh was exported without any animations as an .fbx file named 'characterName' and saved as .max file as well. Then, for each animation, another mesh was exported and a .max file created. This time, the options export animations and animation baked were both checked and the files were named 'characterName@AnimationName' as Unity3D advises [147]. If for some reason, additional animations were requited to fix any skinning problems, they could always be fixed in that file. Then, the skinning envelopes could be exported to the base character's mesh file which, was always the one being exported and used in Unity3D as the animation's target. In Unity3D, all the files named 'characterName@AnimationName' used the base character's model avatar (Figure 107 - B). Thus, any changes on that model would affect all animations. It was also the only file had to be edited to make changes to the whole character. All animations inside Unity had to have their root transforms locked unless they were jump animations where the Y axis had to be unlocked (Figure 107 - C).

Since Unity3D has a great physics engine, all cloth simulations were done using the Cloth Component there. Malik's cloak and Ghora's chainmail among others, are simulated in real time and every time the character moves they have a new animation. This cloth component is divided into two important parts: vertices' distance and penetration and cloth colliders. The first, takes into account of how much a vertice can distance itself from the initial mesh position and how much it can penetrate the other parts of the mesh. The second, defines which colliders the cloth will have. For example, Malik's cloak has a collider on both arms so it does not clip into them while Malik is moving (Figure 107 - D).

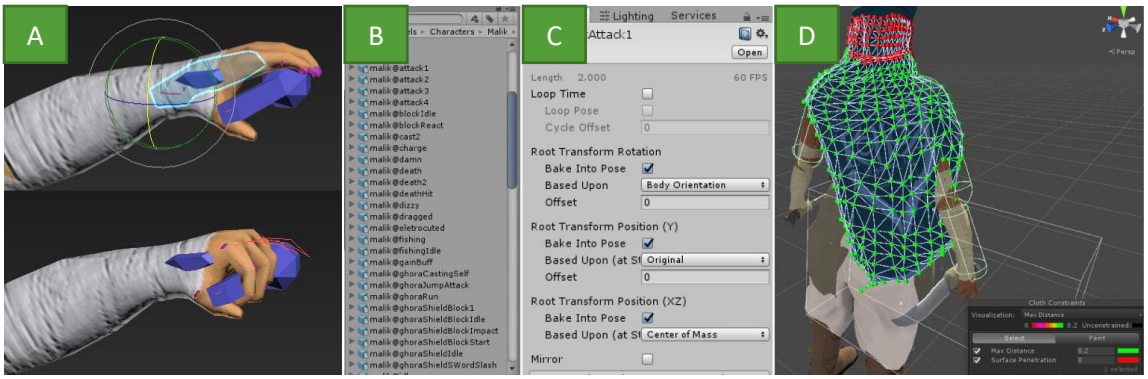


Figure 107 – (A) Biped layers applied to hands (to hold weapons); (B) Malik's animation list in Unity; (C) Unity animation options; (D) Malik's cloth component showing vertices distance and arm colliders.

This set of rules substantially improved the workflow since, any changes to an animation could be done in that specific file without changing anything in the base character model making this process much faster and less error prone.

Due to the time it took to create each character and integrate it with Unity, of the six characters developed, only three were completely implemented. The remaining characters (Lipp, Loper and Wang) were only developed up to the digital sculpting step of the 3D pipeline (Figure 88). Since the game was only a prove of concept, three characters were sufficient to play and learn what steps had to be followed to fully integrate them in Unity3D. The fourth character in the game, Lipp, was

downloaded from Mixamo's free library [148] and integrated into Unity3D so it could support the currently implemented game mechanics.

The progress in the creation of Malik is shown in Figure 108 below. The development of all the other characters can be seen in annex 9.5.2.

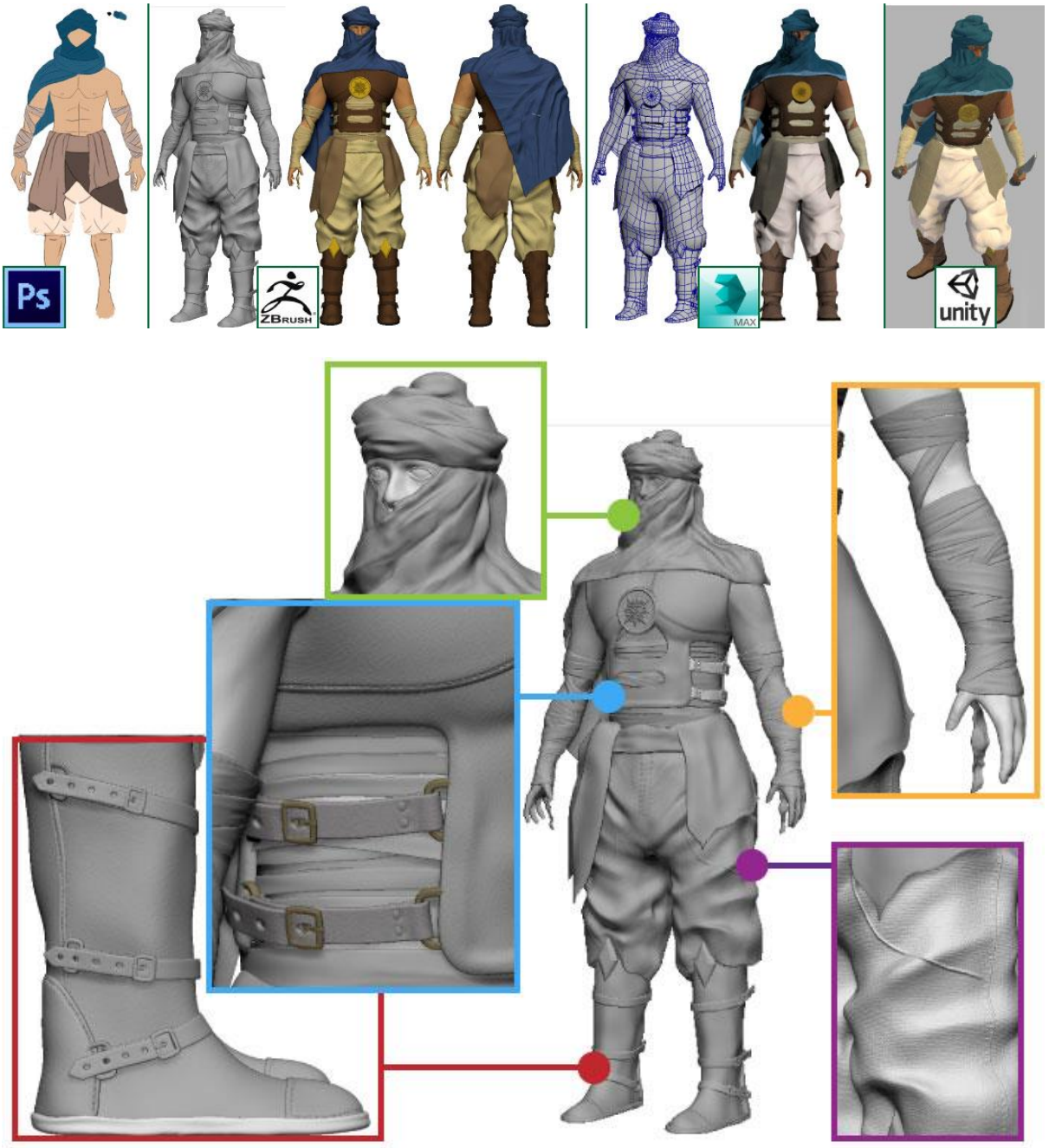


Figure 108 – Malik model creation evolution.

6.8.2 OTHER ASSETS

In the game, one of the animated assets besides the characters are the game's towers. They have a destruction animation to provide additional feedback to the player that the tower has been killed.

This animation consists in breaking the tower into twelve different pieces and making it look like it was destroyed. To achieve this, a 3DsMax plugin was used, Fracture Voronoi [149], [150] which allows choosing in how many parts a mesh will be broken into and then breaks it into smaller meshes (Figure 109 - A). By breaking the tower in twelve parts and adding a MassFx Modifier to all of them it was possible to apply physics to them (Figure 109 - B). The last step was to bake the animation and export the model as an .fbx file to be used later in Unity.

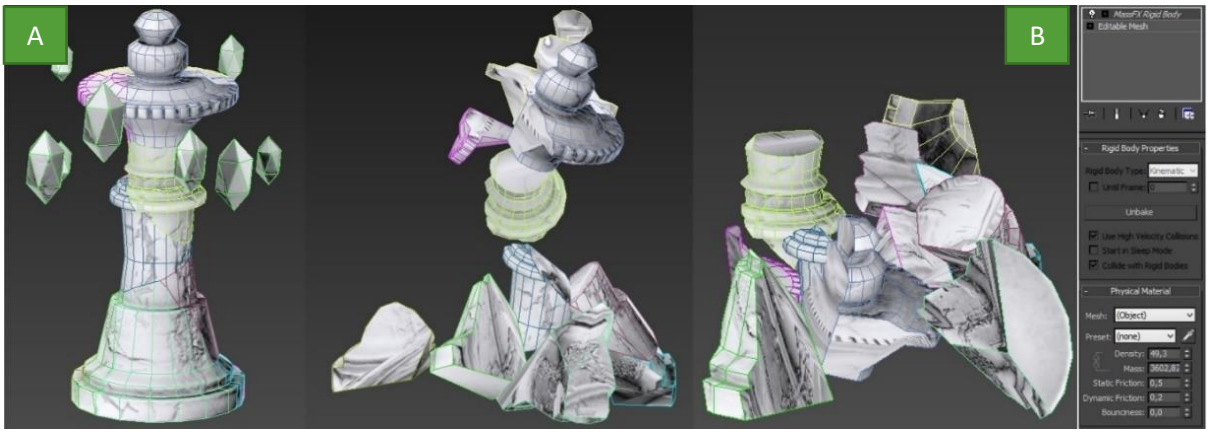


Figure 109 – (A) Tower mesh broken into 12 parts and its animation states; (B) 3DsMax MassFx modifier settings.

6.8.3 MECHANICS INTEGRATION WITH ANIMATIONS

All the ability's mechanics were created in the MTP [2]. One of the requirements for the DAP, was to integrate each ability, elemental power and game mechanic with their corresponding animations and visual feedback. For example, every time a hero is healed, healing particles appear, giving the player visual feedback. On the other hand, every time a hero takes damage, a hit animation is called. However, this needs to be synchronized with the visual feedback from the GUI (floating text) and the different hero's abilities. Attacking with a sword is different from attacking with a spell. With melee attack animations, all that has to be done is to synchronize the attacker's attack animation with the target's take damage animation. On the other hand, with spells, there is no way of knowing how much time it takes for a particle to collide with its target since, it depends on the distance between the target and the attacker. Considering this, a small animation system was implemented, already allowing for its future expansion but also making it independent of the hero currently performing the animation.

Each character in the game has a different 3D prefab. When loading the game, a 3D prefab for each hero is instantiated and referenced in its MonoHero.cs class. Each character's 3D prefab has an animator, a character animation controller script and a particle controller script. The Animator is a standard component from Unity which, allows a user to create a state machine with all the animations that a character can perform using simple parameters to control the transitions between states. Malik's full animator state machine can be seen in Figure 110 which is equal to all the other character's animators. All character animators were created keeping in mind future animation expansions. Taking this into account, every animation state has an index and the current default is zero. If a new animation pack is added, a new index can be added and depending on the index, different animations will be played. The animator is also prepared to support the addition of new abilities to any character since, each ability animation state has the ability number as a transition parameter.

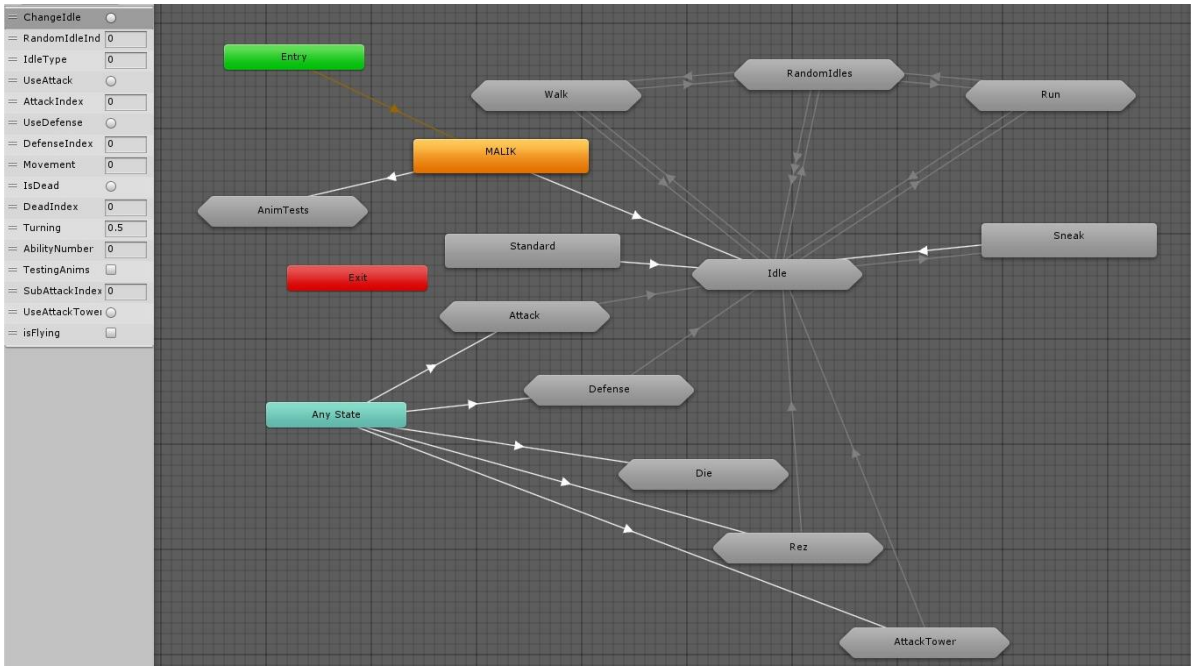


Figure 110 – Animator state machine for the character Malik.

Since Unity3D allows the use of Interfaces, an interface was created to reference a generic hero's animation controller. All animators are equal, varying only on the animations that each one loads. Each hero's game actions such as, walking, running, defending, dyeing, idling and all eight ability animations, are then managed by this controller. For all the necessary game mechanics, particles were also added which are controlled by this script as well. Each ability has its own particle systems which are then called by events on the animation or by the character's animation controller.

For example, the player decides to use a simple ability such as Ghora's jump ability (Ghora3). This ability involves choosing a hex and jumping to it. When Ghora arrives at his destination, he damages all the heroes around him. There is no way of knowing if the hero has already arrived at his destination without using a coroutine and checking every frame if the hero's position is the same as the target hex (Figure 111).

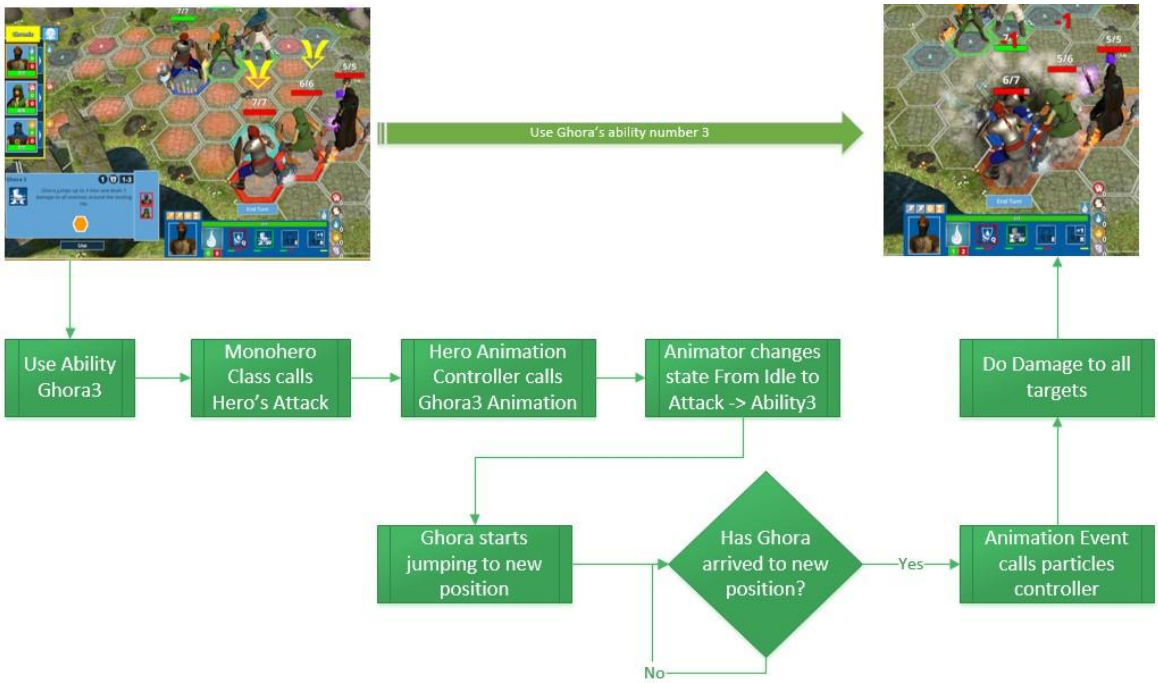


Figure 111 – Pipeline of how an ability is performed behind the game.

The same approach that was used in the integration of hero's mechanics with animations, was used with the game's towers, elemental pets and elemental powers animations.

6.8.4 INDEPENDENT TESTING ENVIRONMENT

To improve the implementation workflow and, to create game components that were independent from the game logic implementation, a helper panel was created. This panel relied on assigning game functionalities to a single button. A few examples of these include: modifying a hero's or tower's health, moving a hero to any tile on the map and unlocking more abilities. This would then be used to test each of the game's GUI functionalities regardless of having the game logic behind it which, was later implemented on the MTP [2]. As the game was developed and the networking components were added, many of these buttons were then changed in the MTP [2] to work on the network as well.

An additional sub-state machine was added to all character animation controllers to be able to test all animations in an independent scene to improve the workflow and particle creation. This scene (3DCharacters scene in Unity project) has a button for each character's animation and in the future, can be used to implement game sound effects and additional particle systems (Figure 112).

Figure 112 – Unity3d game scene environment for testing animations.

6.9 UNITY WORLD CREATION

Unity allows the user to create world terrains which can be painted like in a 3D painting software. Additionally, it allows the user to paint terrain assets (add detailed meshes) such as grass, rocks and Unity trees.

Starting from a terrain plane in Unity, height was added and subtracted to simulate mountains and a small lake. Using seamless textures together with their normal maps and with the hex grid (created in the MTP [2]) as a guideline, the game's world terrain was painted and textured (Figure 113 – A). Then, the world assets were assembled and added to the game's map. Although Unity's terrain detailed meshes feature was compelling, it used an extremely high amount of processor and GPU resources. To overcome this, every single grass, flower, mushroom, rock and tree were added by hand but, all the instances count as a single object. Regardless if there are 10 or 100 clones of the same grass model, Unity only uses the resources to render one model instead of those 10 or 100. Besides having to add each mesh manually, the only other downside of not using the detailed mesh feature is that the terrain's wind system cannot be controlled. To overcome this, a shader [151] was added to the grass models so the wind and bouncing effects could be controlled.

Unity also allows the creation trees. Instead of having to create the 3D models in another 3D program, the user can decide how many branches, sub branches and leaves are in each tree and then give it a log and leave texture (Figure 113 - B). All the trees in the game were created using this method (Figure 113 - B) and all the textures were from the work of a Unity asset publisher named Shapes [152]. For the world creation, some models were taken from the Unity store. All bush models, terrain paint textures and small grass models were obtained from the work of Yughues [153].

Figure 113 – (A) Game's terrain painting starting point; (B) Unity tree settings; (C) Game's terrain and its assets during the development process.

Visual effects are an important part of the game because they define part of the aesthetics. This was done in this game by using Unity's particle system. This system is very overwhelming in the beginning but after some experimentation, the user can create any type of spell or effect. It is important to note that if this system is being used for spells and attacks, it is vital to turn off both shadow casting and shadow receiving features so that it does not consume a lot of computer resources when playing the game. For the DAP, a pack of particles created by A. Salvati [154] was bought from the Unity asset store in order to both learn how particles worked and to use some of the existing particle system's and textures.

As mentioned previously, for each elemental power used/gained (water, fire, earth, lightning and wind) required some sort of feedback in the GUI. A particle system was created that was then called as a character or tower was directly affected by that power. For example, every time a hero is hit by a fire or water elemental power, a geyser/stream of fire or water hits that hero. Every time a hero is hit by an earth or lightning elemental power, a rock falls on their head or the hero is hit by a cloud of lightning respectively. Finally, if a hero or tower is hit by a wind elemental power, a tornado hits or takes it into the air (depending if it is healing or giving movement).

To be able to run the game in computers that do not have the most powerful hardware in the market, additional attention was paid to Unity's lighting system. The game map has one light, the sunlight. Every single 3D object that has a mesh renderer activated needs to have its casting shadows option on. To minimize the use of resources required to run the game all the static objects such as rocks, flowers, trees and statues have their light maps baked (Figure 114 – A and B). This allows Unity to previously create lightmaps of the terrain and the objects on it. This way, there is no need to render these shadows in real time, making the game a lot less demanding, resource wise. The only downside

to this is that the developer must be very careful when creating the game's terrain since a bigger resolution terrain will need more RAM to pre-bake the textures. Assets such as characters and towers still need to be dynamic objects, since their shadows need to move and disappear as they move or die. An additional advantage is the fact that, any player that has a low-quality computer, can still see most of the game's shadows since they were pre-computed in Unity before creating the game's build.

Figure 114 – C shows the final look of the game's terrain and the position of all the assets, together with all the baked light maps.

Figure 114 – (A) Game's terrain with lighting map baked; (B) Game's terrain with all the assets; (C) Game's final terrain and its assets.

7 CONCLUSION

This report describes the successful implementation of a proof of concept for a video game based on a board game. However, it was extremely ambitious and ideally would require at least two other students to fully develop it.

Throughout the DAP, research was conducted to find how information is displayed in other games and to know how to represent the requirements provided by the board game. Knowing and understanding the standard game industry pipeline to produce game assets was a priority to be able to achieve a high-quality standard. This pipeline is mostly the same between game studios but varies considerably in the software tools used which, were analyzed and compared considering the requirements and constraints of the GGP.

The DAP was also responsible for creating and implementing the game's interface. This interface includes both elements before the play session (menus) and the play session itself (in-game). Its implementation was prepared to support multiple languages which, would ease the localization process. User tests were performed for the menus interface while, monitoring the performance of users. This information was used to guide the improvements of the interface.

The narrative of the game was then established and the aesthetics of the world and its characters were defined. Using these aesthetics as a foundation, the asset creation pipeline was applied to produce all the game's 3D content which, was then integrated into the game engine, Unity3D.

To complete the game's implementation, all the content produced by the GIP was periodically combined with the MTP.

One of the main hurdles of the GIP was how the development pipeline was planned and executed on both the MTP and the DAP. Now that the GIP has been concluded, it is clear that the implementation of the interface should have been carried out at a later stage of the development. It should have happened once all the data structures and the data base implementation had been concluded, instead of while they were still being developed. This slowed down the overall progress of the MTP and the DAP because part of the development had to be redone or reviewed since it was not prepared for the full integration of the MTP and the DAP.

Despite all of this, the GIP was successful and, served as a great learning opportunity on both personal and professional levels. The fact that most of the references were directly from the game industry served to provide a unique perspective but also, firsthand experience that is applicable when applying for a job in the video game industry.

The next steps in the development of this game will be to implement a higher fidelity version of the interface, to implement the sound effects, ambient sounds and music of the game and to conclude the integration of the characters into the game engine.

Through this project it is possible to get a glimpse into the development of a videogame. This is a complex process that requires many fields of knowledge. It also showed how much had been learnt in the years spent studying Computer Engineering at the University of Madeira and how some courses provided the knowledge of where to start the development of each part of the game.

8 REFERENCES

[1] R. Lauer, “The Elemental Tetrad of Games | Cleveland Institute of Art College of Art | 800.223.4700.” .

[2] Y. Almeida, “Thesis of Keepers of Intheris: Mechanics and Technology,” 2017.

[3] Unity, “Unity - Game Engine.” [Online]. Available: https://unity3d.com/. [Accessed: 22-Sep-2016].

[4] Microsoft, “Agile, Git, CI with TFS | Team Foundation Server.” [Online]. Available: https://www.visualstudio.com/tfs/. [Accessed: 18-Dec-2016].

[5] Eli Epstein, “The evolution of gaming, from the console to the cloud.” [Online]. Available: http://mashable.com/2015/01/08/gaming-tech-ces/#sIGm8bRtakq9. [Accessed: 12-Nov-2016].

[6] R. Chikhani, “The History Of Gaming: An Evolving Community | TechCrunch.” [Online]. Available: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/. [Accessed: 13-Nov-2016].

[7] T. Stein, “The Evolution of Story & Characters in Games | Big Fish Blog.” [Online]. Available: http://www.bigfishgames.com/blog/video-game-stories-evolution-of-story-character-development/. [Accessed: 26-Nov-2016].

[8] R. Leadbetter, “Q*Bert • Eurogamer.net.” [Online]. Available: http://www.eurogamer.net/articles/q-bert-review. [Accessed: 26-Nov-2016].

[9] W. Westwater, “The Evolution Of Storytelling In Video Games.” [Online]. Available: http://parade.com/301954/camayak/the-evolution-of-storytelling-in-video-games/. [Accessed: 26-Nov-2016].

[10] J. Cork, “Game Informer’s Top 100 Games Of All Time (Circa Issue 100).” [Online]. Available: http://www.gameinformer.com/b/features/archive/2009/11/16/game-informer-s-top-100-games-of-all-time-circa-issue-100.aspx?PostPageIndex=2. [Accessed: 26-Nov-2016].

[11] The Well-Red Mage, “The Legend of Zelda (1986) ‘First Quest’ |.” [Online]. Available: https://thewellredmage.wordpress.com/2016/03/19/the-legend-of-zelda-first-quest/. [Accessed: 26-Nov-2016].

[12] W. Henfield, “Gaming Greats: Metroid (1986) | SquabbleBox.co.uk - Entertainment Under Attack.” [Online]. Available: http://www.squabblebox.co.uk/2016/02/gaming-greats-metroid-1986/. [Accessed: 26-Nov-2016].

[13] Bethesda Softworks, “Wolfenstein 3D,” 1992. [Online]. Available: http://3d.wolfenstein.com/. [Accessed: 08-Oct-2015].

[14] B. Softworks, “Doom,” 1993. [Online]. Available: http://bethsoft.com/en-us/games/doom. [Accessed: 08-Oct-2015].

[15] Red List, “1996, Quake, id Software: Graphic Design, 3D | The Red List.” [Online]. Available: http://theredlist.com/wiki-2-343-918-344-1207-view-beginnings-1-profile-1996-bquake-b-id-software.html. [Accessed: 26-Nov-2016].

[16] Extra Credits, “Graphics vs. Aesthetics - Why High Resolution Graphics Aren’t Enough - Extra

Credits - YouTube." [Online]. Available: https://www.youtube.com/watch?v=5oK8UTRgvJU. [Accessed: 26-Nov-2016].

[17] A. Hutchison, "Game Studies - Making the Water Move: Techno-Historic Limits in the Game Aesthetics of Myst and Doom." [Online]. Available: http://gamestudies.org/0801/articles/hutch. [Accessed: 26-Nov-2016].

[18] A. Wall, "Ten Ton Hammer | Arena Gaming & the Birth of the MOBA." [Online]. Available: http://www.tentonhammer.com/articles/arena-gaming-the-birth-of-the-moba. [Accessed: 14-Nov-2016].

[19] Daleske, "Daleske Space - History of Computing - PLATO System - Games - Social Media." [Online]. Available: http://www.daleske.com/. [Accessed: 26-Nov-2016].

[20] "Civilization V | Civilization Wiki." [Online]. Available: http://civilization.wikia.com/wiki/Civilization_V. [Accessed: 26-Nov-2016].

[21] Take-Two Interactive, "XCOM.com | XCOM 2." [Online]. Available: https://xcom.com/pt. [Accessed: 23-Nov-2016].

[22] IndieRetroNews, "Indie Retro News: Dune 2 - History and Remakes." [Online]. Available: http://www.indieretronews.com/2012/12/dune-2-history-and-remakes.html. [Accessed: 26-Nov-2016].

[23] Total War, "Home - Total War." [Online]. Available: https://www.totalwar.com/. [Accessed: 26-Nov-2016].

[24] J. Funk, "MOBA, DOTA, ARTS: A brief introduction to gaming's biggest, most impenetrable genre - Polygon." [Online]. Available: http://www.polygon.com/2013/9/2/4672920/moba-dota-arts-a-brief-introduction-to-gamings-biggest-most. [Accessed: 14-Nov-2016].

[25] Riot Games, "Welcome to League of Legends." [Online]. Available: http://play.na.leagueoflegends.com/en_US. [Accessed: 26-Nov-2016].

[26] Valve, "Dota 2." [Online]. Available: http://blog.dota2.com/. [Accessed: 26-Nov-2016].

[27] Activision Blizzard, "Heroes of the Storm Official Game Site." [Online]. Available: http://us.battle.net/heroes/en/. [Accessed: 23-Nov-2016].

[28] Hi-Rez studios, "SMITEgame.com." [Online]. Available: https://www.smitegame.com/. [Accessed: 26-Nov-2016].

[29] Z. Ye, "Designing User Interfaces for Games," Carnegie Mellon, 2000.

[30] D. Krannich, *Research in Media Informatics on Advanced User Interfaces*. Nordestedt: Books on Demand GmbH, 2012.

[31] C. S. Ang, P. Zaphiris, and S. Mahmood, "A model of cognitive loads in massively multiplayer online role playing games," *Interact. Comput.*, vol. 19, no. 2, pp. 167–179, 2007.

[32] J. Juul and M. Norton, "Easy to Use and Incredibly Difficult: On the Mythical Border between Interface and Gameplay," *Proc. 4th Int. Conf. Found. Digit. Games - FDG '09*, p. 107, 2009.

[33] D. Pinelle, N. Wong, and T. Stach, "Heuristic evaluation for games: usability principles for video game design," *Proc. SIGCHI Conf. …*, pp. 1453–1462, 2008.

[34] A. Stonehouse, “Gamasutra: Anthony Stonehouse’s Blog - User interface design in video games.” [Online]. Available: http://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php. [Accessed: 24-Oct-2016].

[35] E. Fagerholt and M. Lorentzon, “Beyond the HUD. User Interfaces for Increased Player Immersion in FPS Games,” *Test*, p. 124, 2009.

[36] M. Andrews, “Gamasutra - Game UI Discoveries: What Players Want.” [Online]. Available: http://www.gamasutra.com/view/feature/4286/game_ui_discoveries_what_players_.php?print=1. [Accessed: 24-Oct-2016].

[37] Ubisoft, “Assassin’s Creed® II | Official GB Site | Ubisoft.” [Online]. Available: https://assassinscreed.ubi.com/en-GB/games/assassins-creed-2.aspx. [Accessed: 23-Nov-2016].

[38] “Eagle Vision | Assassin’s Creed Wiki | Fandom powered by Wikia.” [Online]. Available: http://assassinscreed.wikia.com/wiki/Eagle_Vision. [Accessed: 23-Nov-2016].

[39] GameSpot, “Call of Duty: Modern Warfare 2 Review - GameSpot.” [Online]. Available: http://www.gamespot.com/reviews/call-of-duty-modern-warfare-2-review/1900-6239806/. [Accessed: 23-Nov-2016].

[40] L. Elliott, A. Golub, G. Ream, and E. Dunlap, “Video game genre as a predictor of problem use.,” *Cyberpsychol. Behav. Soc. Netw.*, vol. 15, no. 3, pp. 155–61, Mar. 2012.

[41] K. A. Johansson, “Next generation interface for MMO-RPG Creating a immersive experience,” 2011.

[42] S. O. Entertainment, “Everquest,” 1999. [Online]. Available: https://www.everquest.com/home. [Accessed: 08-Oct-2015].

[43] Activision Blizzard, “World of Warcraft,” 2004. [Online]. Available: http://eu.battle.net/wow/en/. [Accessed: 08-Oct-2015].

[44] M. Davidovici-nora, “The Dynamics of Co-Creation in the Video Game Industry: The Case of World of Warcraft,” *Commun. Strateg.*, no. 73, pp. 43–66, 2009.

[45] S. Targett, V. Verlysdonk, H. Hamilton, and D. Hepting, “A Study of User Interface Modifications in World of Warcraft,” *Game Stud. - Int. J. Comput. game Res.*, vol. 12, no. 2, pp. 1–24, 2012.

[46] J. Dyck, D. Pinelle, B. Brown, and C. Gutwin, “Learning from Games: HCI Design Innovations in Entertainment Software.,” *Graph. Interface*, vol. 2003, pp. 237–246, 2003.

[47] K. Gkikas, D. Nathanael, and N. Marmaras, “The evolution of FPS games controllers : how use progressively shaped their present design,” no. 2004.

[48] Activision Blizzard, “Warcraft 2,” 1995. [Online]. Available: http://us.blizzard.com/en-us/games/war3/. [Accessed: 08-Oct-2015].

[49] J. Fromme and A. Unger, *Computer Games and New Media Cultures: A Handbook of Digital Games Studies*. London: Springer, 2012.

[50] Ubisoft, “Might & Magic® Heroes 7 | Ubisoft Official | Fantasy RPG.” [Online]. Available: https://mmh7.ubi.com/de/blog. [Accessed: 23-Nov-2016].

[51] Paradox Interactive, “Magicka | Paradox Interactive.” [Online]. Available: http://www.magickagame.com/. [Accessed: 23-Nov-2016].

[52] Activision Blizzard, “Hearthstone: Heroes of Warcraft Official Game Site.” [Online]. Available: http://us.battle.net/hearthstone/en/. [Accessed: 16-Nov-2016].

[53] Polycount, “Texture types - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Texture_types. [Accessed: 03-Oct-2016].

[54] E. Russell, “Understanding the Difference between Texture Maps.” [Online]. Available: http://blog.digitaltutors.com/understanding-difference-texture-maps/. [Accessed: 03-Oct-2016].

[55] Polycount, “Diffuse map - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Diffuse_map. [Accessed: 03-Oct-2016].

[56] Unity, “Unity - Manual: Optimizing graphics performance.” [Online]. Available: https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html. [Accessed: 03-Oct-2016].

[57] E. Russell, “Know the Difference: Bump, Normal and Displacement Maps.” [Online]. Available: http://blog.digitaltutors.com/bump-normal-and-displacement-maps/. [Accessed: 03-Oct-2016].

[58] A. Oshchepkov, “A Practical Guide On Normal Mapping for Games (1).docx - Google Drive.” [Online]. Available: https://drive.google.com/file/d/0B02lElvs8BcvYllmQWpXUGxod3M/view. [Accessed: 03-Oct-2016].

[59] Wayne, “Quick Reference: Baking Normals from a Bump Map | theZEDLAB.” [Online]. Available: http://thezedlab.com/quick-reference-baking-normals-from-a-bump-map/. [Accessed: 26-Nov-2016].

[60] Polycount, “Height map - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Height_map. [Accessed: 03-Oct-2016].

[61] CRYTEK, “Tangent Space Normal Mapping - Technical Documentation - Documentation.” [Online]. Available: http://docs.cryengine.com/display/SDKDOC4/Tangent+Space+Normal+Mapping. [Accessed: 27-Nov-2016].

[62] “Displacement mapping - Wikiwand.” [Online]. Available: http://www.wikiwand.com/it/Displacement_mapping. [Accessed: 26-Nov-2016].

[63] Polycount, “Light map - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Light_map. [Accessed: 03-Oct-2016].

[64] Polycount, “Ambient occlusion map - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Ambient_occlusion_map. [Accessed: 03-Oct-2016].

[65] S. Spencer, *ZBrush Character Creation: Adavanced Digital Sculpting*. 2008.

[66] Unity, “Unity - Manual: Emission.” [Online]. Available: https://docs.unity3d.com/Manual/StandardShaderMaterialParameterEmission.html. [Accessed: 26-Nov-2016].

[67] Adobe, “Adobe Photoshop CC.” [Online]. Available:

https://www.adobe.com/pt/products/photoshop.html?promoid=KLXLS. [Accessed: 29-Sep-2016].

[68] Pixologic, “Pixologic :: Home of ZBrush.” [Online]. Available: http://pixologic.com/. [Accessed: 19-Sep-2016].

[69] L. Ahearn, *3D Game Textures*, Second Edi. 2009.

[70] Leigh Van Der Byl, “PHOTOREALISTIC TEXTURING FOR DUMMIES.” .

[71] M. Masters, “Understanding 3D UV Texturing.” [Online]. Available: http://blog.digitaltutors.com/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know/. [Accessed: 06-Oct-2016].

[72] G. Henrique, “What To Know When Creating Next Gen Assets : CG Masters.” [Online]. Available: http://www.cgmasters.net/free-tutorials/what-to-know-when-creating-next-gen-assets/. [Accessed: 06-Oct-2016].

[73] R. J. S. Sloan, *Virtual Character Design for Games and Interactive Media*. 2015.

[74] M. Masters, “The Rise of Motion Capture and What it Means for You.” [Online]. Available: http://blog.digitaltutors.com/rise-motion-capture-means/. [Accessed: 17-Oct-2016].

[75] Microsoft, “Kinect para Xbox One | Xbox.com.” [Online]. Available: http://www.xbox.com/pt-PT/xbox-one/accessories/kinect-for-xbox-one#fbid=2xh6BxS8lYv. [Accessed: 27-Nov-2016].

[76] A. Beane, *3D Animation Essential*. 2012.

[77] K. Forrester, “A research investigation into asset creation and production pipelines for Unreal Engine 3,” 2009.

[78] G. A., *3ds Max Modeling for Games. Insider’s Guide to Game Character, Vehicle, and Environment Modeling*. .

[79] G. Tironeac, “The Pipeline Behind Modeling and Animating a Game Character in Zelgor | ASSIST Software Romania.” [Online]. Available: https://assist-software.net/blog/pipeline-behind-modeling-and-animating-game-character-zelgor.

[80] P. Ciszek, “3D Production Pipeline in Game Development Title: 3D Production Pipeline in Game Development Työn nimi: 3D-tuotantolinja pelikehityksessä,” 2012.

[81] M. Dublin, “Persian Warrior / Art Bully Productions Workshop — polycount.” [Online]. Available: http://polycount.com/discussion/155586/persian-warrior-art-bully-productions-workshop. [Accessed: 27-Nov-2016].

[82] M. Masters, “Why Are Ngons and Triangles so Bad?” [Online]. Available: http://blog.digitaltutors.com/ngons-triangles-bad/. [Accessed: 11-Oct-2016].

[83] G. Boudon, “How Does a 3D Production Pipeline Work.” [Online]. Available: http://blog.digitaltutors.com/understanding-a-3d-production-pipeline-learning-the-basics/. [Accessed: 11-Oct-2016].

[84] B. Harvey, “Current Trends in 3D Modelling and Game Asset pipeline example | bridgetharveysae.” [Online]. Available: https://bridgetharveysae.wordpress.com/2014/07/02/current-trends-in-3d-modelling-and-game-asset-pipeline-example/. [Accessed: 11-Oct-2016].

[85] J. Amin, “Maya Modeling: Polygonal Modeling Theory.” [Online]. Available: http://www.3dtotal.com/tutorial/1754-maya-modeling-polygonal-modeling-theory-by-jahirul-amin-character-face. [Accessed: 27-Nov-2016].

[86] M. Aerni, “ZBrush Speedsculpt.” [Online]. Available: http://www.3dtotal.com/tutorial/2001-zbrush-speedsculpt-3ds-max-v-ray-by-mathieu-aerni-sculpt. [Accessed: 27-Nov-2016].

[87] Polycount, “Topology - polycount.” [Online]. Available: http://wiki.polycount.com/wiki/Topology. [Accessed: 11-Oct-2016].

[88] J. Kinney, “Tips for Creating Perfect Normal Maps Every Time.” [Online]. Available: http://blog.digitaltutors.com/tips-creating-perfect-normal-maps-every-time/. [Accessed: 13-Oct-2016].

[89] Xormal, “xNormal web page.” [Online]. Available: http://www.xnormal.net/. [Accessed: 27-Nov-2016].

[90] Quixel, “Quixel - Scan based PBR texturing.” [Online]. Available: http://quixel.se/suite/. [Accessed: 27-Nov-2016].

[91] Allegorithmic, “Substance Painter - 3D Painting Software.” [Online]. Available: https://www.allegorithmic.com/products/substance-painter. [Accessed: 27-Nov-2016].

[92] RFX, “RFX Inc.” [Online]. Available: http://www.rfx.com/products/80. [Accessed: 27-Nov-2016].

[93] Unity, “Mecanim Humanoids – Unity Blog.” [Online]. Available: https://blogs.unity3d.com/pt/2014/05/26/mecanim-humanoids/. [Accessed: 28-Dec-2016].

[94] S. Kumari, “Complete Human Character Rig In 3D Studio Max, Part 5 Skinning.” [Online]. Available: https://cgi.tutsplus.com/tutorials/complete-human-character-rig-in-3d-studio-max-part-5-skinning--cg-18775. [Accessed: 27-Nov-2016].

[95] Autodesk, “3ds Max | 3D Modeling, Animation | Rendering Software | Autodesk.” [Online]. Available: http://www.autodesk.com/products/3ds-max/overview. [Accessed: 19-Sep-2016].

[96] Autodesk, “Maya | Computer Animation & Modeling Software | Autodesk.” [Online]. Available: http://www.autodesk.com/products/maya/overview. [Accessed: 27-Nov-2016].

[97] Blender Foundation, “blender.org - Home of the Blender project - Free and Open 3D Creation Software.” [Online]. Available: https://www.blender.org/. [Accessed: 19-Sep-2016].

[98] Autodesk, “Game Development Software | Maya LT | Autodesk.” [Online]. Available: http://www.autodesk.com/products/maya-lt/overview.

[99] “Maya vs Maya LT: Why I Decided to Switch to Maya LT for Game Environment Modeling and UVing.” [Online]. Available: http://www.worldofleveldesign.com/categories/3d-game-modeling/maya-vs-mayalt-why-i-switched-to-mayalt.php. [Accessed: 19-Sep-2016].

[100] Autodesk, “Maya Vs Maya LT | Compare Maya Products | Autodesk.” [Online]. Available: http://www.autodesk.com/products/maya/compare/compare-products. [Accessed: 19-Sep-2016].

[101] M. Masters, “3ds Max vs. Maya: Is One Better than the Other?,” *Digital Tutors*. [Online]. Available: http://blog.digitaltutors.com/3ds-max-vs-maya-is-one-better-than-the-other/. [Accessed: 19-Sep-2016].

[102] M. Masters, “Which 3D Software Should I Choose for Asset Creation?,” *Digital Tutors*. [Online]. Available: http://blog.digitaltutors.com/3ds-max-maya-lt-blender-3d-software-choose-asset-creation/. [Accessed: 19-Sep-2016].

[103] M. Bousquet, *How to Cheat in 3ds Max 2011*. .

[104] F. Freedom, “Biped vs CAT vs Max Bones | An Animator’s Journey.” [Online]. Available: https://faeythfreedom.wordpress.com/2013/10/20/biped-vs-cat-vs-max-bones/. [Accessed: 19-Sep-2016].

[105] M. Masters, “Understanding the Importance of Skinning Your Rigs.” [Online]. Available: http://blog.digitaltutors.com/understanding-skinning-vital-step-rigging-project/. [Accessed: 16-Nov-2016].

[106] Autodesk, “Digital Painting And Sculpting Software | 3D Painting | Mudbox.” [Online]. Available: http://www.autodesk.com/products/mudbox/overview. [Accessed: 19-Sep-2016].

[107] J. Marshall, “ZBrush or Mudbox: Sculpting Showdown,” *Digital Tutors*. [Online]. Available: http://blog.digitaltutors.com/zbrush-mudbox-sculpting-showdown/. [Accessed: 19-Sep-2016].

[108] Wacom, “Wacom | Interactive Pen Displays & Tablet Styluses.” [Online]. Available: http://www.wacom.com/en-us. [Accessed: 16-Nov-2016].

[109] B. Schwulst, “Mudbox vs Zbrush – 3D Sculpture Software Comparison.” [Online]. Available: https://blog.udemy.com/mudbox-vs-zbrush/. [Accessed: 19-Sep-2016].

[110] Pixologic, “ZRemesher » ZBrush Docs.” [Online]. Available: http://docs.pixologic.com/user-guide/3d-modeling/topology/zremesher/. [Accessed: 16-Nov-2016].

[111] Pluralsight Creative, “Mudbox Top Tip: Quickly Paint the Base Skin Textures for Your Game Character - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=Mc_Og5kwECo. [Accessed: 16-Nov-2016].

[112] M. Masters, “3D Texturing Terminology | Learning to 3D Texture.” [Online]. Available: http://blog.digitaltutors.com/cover-bases-common-3d-texturing-terminology/. [Accessed: 29-Sep-2016].

[113] Epic Games, “About Unreal Engine 4.” [Online]. Available: https://www.unrealengine.com/unreal-engine-4. [Accessed: 22-Sep-2016].

[114] Crytek, “CryENGINE | Crytek.” [Online]. Available: http://www.crytek.com/cryengine. [Accessed: 22-Sep-2016].

[115] M. Masters, “Choosing the Right Game Engine | Unity, Source 2, Unreal Engine 4 or CryENGINE.” [Online]. Available: http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/. [Accessed: 22-Sep-2016].

[116] “Unity 5 vs Unreal Engine 4 – Create 3D Games.” [Online]. Available: https://create3dgames.wordpress.com/2015/09/07/unity-5-vs-unreal-engine-4/. [Accessed: 05-Oct-2016].

[117] R. Fortin, “Gamasutra: Raphael Fortin’s Blog - 5 reasons why Unity is better for learning game development than Unreal Engine.” [Online]. Available: http://www.gamasutra.com/blogs/RaphaelFortin/20150302/237665/5_reasons_why_Unity_i

s_better_for_learning_game_development_than_Unreal_Engine.php. [Accessed: 22-Sep-2016].

[118] Unity Demo Team, “Unity - Adam.” [Online]. Available: https://unity3d.com/pages/adam. [Accessed: 05-Oct-2016].

[119] A. Vintsevych, “Gamasutra: Andrii Vintsevych’s Blog - Unity 5 vs Unreal 4: From a single developer perspective.” [Online]. Available: http://www.gamasutra.com/blogs/AndriiVintsevych/20150717/248808/Unity_5_vs_Unreal_4_From_a_single_developer_perspective.php. [Accessed: 05-Oct-2016].

[120] A. Haining, “Gamasutra: Andrew Haining’s Blog - Unity, Unreal Engine, Source or Roll your own engine: Pros and Cons.” [Online]. Available: http://www.gamasutra.com/blogs/AndrewHaining/20150306/238146/Unity_Unreal_Engine_Source_or_Roll_your_own_engine_Pros_and_Cons.php. [Accessed: 22-Sep-2016].

[121] CryTek3, “Crysis 3 | Pierre-Yves Donzallaz | Page 2.” [Online]. Available: https://pydonzallaz.wordpress.com/category/crysis-3/page/2/. [Accessed: 16-Nov-2016].

[122] N. Scoutter, “Doom 4 News - Latest DOOM Screens Show Powerful idTech 6 Graphics Engine In Action.” [Online]. Available: http://www.game-debate.com/news/19056/latest-doom-screens-show-powerful-idtech-6-graphics-engine-in-action. [Accessed: 16-Nov-2016].

[123] A. Mayden, “Unreal Engine 4 vs. Unity: Which Game Engine Is Best for You?” [Online]. Available: http://blog.digitaltutors.com/unreal-engine-4-vs-unity-game-engine-best/. [Accessed: 05-Oct-2016].

[124] S. Prescott, “CryEngine V releases today on a pay-what-you-want basis | PC Gamer.” [Online]. Available: http://www.pcgamer.com/cryengine-v-releases-today-on-a-pay-what-you-want-basis/. [Accessed: 05-Oct-2016].

[125] T. Vieira, Y. Almeida, and J. Serina, “Game of Pawns - The rule book.” [Online]. Available: https://docs.google.com/document/d/1-cNKUIjTcKfzBaJzae0YbmlYBmNRT4Jm2BBlSG7bwbY/edit#heading=h.18p07vhlrwr7.

[126] The Writter’s Journey, “The Hero’s Journey Outline.” [Online]. Available: http://www.thewritersjourney.com/hero’s_journey.htm. [Accessed: 19-Dec-2016].

[127] Crystal Dynamics, “Tomb Raider: underworld.” [Online]. Available: https://www.tombraider.com/en-us. [Accessed: 19-Dec-2016].

[128] 2kGames, “BioShock.” [Online]. Available: https://www.2kgames.com/bioshock/. [Accessed: 19-Dec-2016].

[129] Bethesda, “Dishonored 2.” [Online]. Available: https://dishonored.bethesda.net/en. [Accessed: 19-Dec-2016].

[130] J. Garrett, “Vikings of Bjornstad - English to Old Norse Dictionary.” [Online]. Available: http://www.vikingsofbjornstad.com/Old_Norse_Dictionary_E2N.shtm. [Accessed: 19-Nov-2016].

[131] Wikipedia, “List of average human height worldwide.” [Online]. Available: https://en.wikipedia.org/wiki/List_of_average_human_height_worldwide. [Accessed: 19-Nov-2016].

[132] Game Icons, “Game-icons Library.” [Online]. Available: http://game-icons.net/. [Accessed: 01-Dec-2016].

[133] Digital Extremes, “Warframe.” [Online]. Available: https://warframe.com/landing. [Accessed: 01-Dec-2016].

[134] Wizards of the Coast, “Magic Online | MAGIC: THE GATHERING.” [Online]. Available: http://magic.wizards.com/en/content/magic-online-products-game-info. [Accessed: 26-Dec-2016].

[135] Software AG, “How and Why Are Companies Using XML?: Benefits of XML (in general).” [Online]. Available: http://www.mulberrytech.com/papers/HowAndWhyXML/slide009.html. [Accessed: 03-Dec-2016].

[136] W. Reich, “Gamasutra: Wendelin Reich’s Blog - C# Memory Management for Unity Developers (part 1 of 3).” [Online]. Available: http://www.gamasutra.com/blogs/WendelinReich/20131109/203841/C_Memory_Management_for_Unity_Developers_part_1_of_3.php. [Accessed: 14-Dec-2016].

[137] A. Ernest and B. Ip, “Gamasutra - From Casual to Core: A Statistical Mechanism for Studying Gamer Dedication.” [Online]. Available: http://www.gamasutra.com/view/feature/131397/from_casual_to_core_a_statistical_.php?page=1. [Accessed: 21-Nov-2016].

[138] J. P. Chin, V. a Diehl, and L. K. Norman, “Development of an instrument measuring user satisfaction of the human-computer interface,” *CHI ’88- Proc. SIGCHI Conf. Hum. factors Comput. Syst.*, pp. 213–218, 1988.

[139] O. Vandecasteele, “non Quad Selector | ScriptSpot.” [Online]. Available: http://www.scriptspot.com/3ds-max/scripts/non-quad-selector. [Accessed: 27-Dec-2016].

[140] W. Bradberry, “Zbrush 4r4 tutorial - Making of fine details.” [Online]. Available: https://www.youtube.com/watch?v=INOtmFnruyk. [Accessed: 27-Dec-2016].

[141] Pixologic, “Insert Multi Mesh Repository.” [Online]. Available: http://www.zbrushcentral.com/showthread.php?170167-Insert-Multi-Mesh-Repository. [Accessed: 27-Dec-2016].

[142] eFerrari, “Mummy Wraps/Bandages Technique with SliceCurve & Panel Loops.” [Online]. Available: http://www.zbrushcentral.com/showthread.php?178522-Mummy-Wraps-Bandages-Technique-with-SliceCurve-amp-Panel-Loops. [Accessed: 27-Dec-2016].

[143] vilson martins, “ZBRUSH ENGRAVING BRUSH.” [Online]. Available: https://pt.pinterest.com/pin/494551602805068676/. [Accessed: 27-Dec-2016].

[144] M. Dunnam, “Zbrush Brushes repository.” [Online]. Available: https://gumroad.com/mdunnam#. [Accessed: 27-Dec-2016].

[145] E. Körner, “Working with Physically-Based Shading: a Practical Approach – Unity Blog.” [Online]. Available: https://blogs.unity3d.com/2015/02/18/working-with-physically-based-shading-a-practical-approach/. [Accessed: 26-Nov-2016].

[146] Unity, “Unity - Manual: Smoothness.” [Online]. Available: https://docs.unity3d.com/Manual/StandardShaderMaterialParameterSmoothness.html. [Accessed: 28-Dec-2016].

[147] Unity3D, “Unity - Manual: Preparing your own character.” [Online]. Available: https://docs.unity3d.com/Manual/Preparingacharacterfromscratch.html. [Accessed: 28-Dec-2016].

[148] Adobe, “Rig Your 3D Characters Automatically, Auto-Rigger.” [Online]. Available: https://www.mixamo.com/auto-rigger. [Accessed: 28-Dec-2016].

[149] Garp, “Fracture Voronoi | ScriptSpot.” [Online]. Available: http://www.scriptspot.com/3ds-max/scripts/fracture-voronoi. [Accessed: 28-Dec-2016].

[150] Lee Jackson, “3D Studio MAX - Massfx - Wall break - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=Ta3hJ789D4k. [Accessed: 28-Dec-2016].

[151] piablood, “Toon Forest Free Set - Asset Store.” [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/66124. [Accessed: 28-Dec-2016].

[152] Shapes, “Nature Starter Kit 2 - Asset Store.” [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/52977. [Accessed: 28-Dec-2016].

[153] Nobiax, “Nobiax / Yughues - Asset Store.” [Online]. Available: https://www.assetstore.unity3d.com/en/#!/search/page=1/sortby=popularity/query=publisher:4986. [Accessed: 28-Dec-2016].

[154] A. Salvati, “MEGA 503 Magic Spells FX - Asset Store.” [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/23294. [Accessed: 28-Dec-2016].

9 APPENDIXES

9.1 GANTT

Table 23 – GANTT table

Task Name	Duration	Start	Finish	Predecessors	Resource Names
Navegação de Menus com cliente ligado ao servidor	**165 days**	**Mon 05/10/15**	**Fri 20/05/16**		
Estrutura de Dados	**130 days**	**Mon 05/10/15**	**Fri 01/04/16**		
Criação do UML conceptual de classes	**130 days**	**Mon 05/10/15**	**Fri 01/04/16**		
ED GUI	100 days	Mon 05/10/15	Fri 19/02/16		Tatiana & Yuri
ED InGame	15 days	Mon 29/02/16	Fri 18/03/16		Tatiana;Yuri
Habilidades	30 days	Mon 15/02/16	Fri 25/03/16		Yuri
Elements ED	5 days	Mon 28/03/16	Fri 01/04/16		Yuri
Base de dados	**110 days**	**Mon 19/10/15**	**Fri 18/03/16**		
Criação diagram ER	15 days	Mon 19/10/15	Fri 06/11/15		Tatiana;Yuri
Ligação da base de dados ao Unity	5 days	Mon 23/11/15	Fri 27/11/15		Yuri
Estruturação das conexões as bases de dados (abstrações)	45 days	Mon 30/11/15	Fri 29/01/16	10;9	Tatiana;Yuri
Implementação base de dados (Selects)	**30 days**	**Mon 04/01/16**	**Fri 12/02/16**		
Copia local da base de dados	30 days	Mon 04/01/16	Fri 12/02/16		Yuri
Player	30 days	Mon 04/01/16	Fri 12/02/16		Tatiana;Yuri
Implementação base de dados (Inserts e Updates)	10 days	Mon 07/03/16	Fri 18/03/16		Yuri
GUI	**165 days**	**Mon 05/10/15**	**Fri 20/05/16**		
Navegação	**130 days**	**Mon 05/10/15**	**Fri 01/04/16**		
Árvore de menus	25 days	Mon 05/10/15	Fri 06/11/15		Tatiana
Conceito GUI	40 days	Mon 09/11/15	Fri 01/01/16	18	Tatiana
Log de Tarefas	15 days	Mon 02/11/15	Fri 20/11/15		Tatiana
Guião	5 days	Mon 14/03/16	Fri 18/03/16		Tatiana
Testes Utilizadores	9 days	Tue 22/03/16	Fri 01/04/16	26;20;21	Tatiana
Implementação	**45 days**	**Mon 08/02/16**	**Fri 08/04/16**		
XML linguagens + tooltip	10 days	Mon 08/02/16	Fri 19/02/16		Tatiana
Login	5 days	Mon 07/03/16	Fri 11/03/16		Tatiana
Menus	20 days	Mon 22/02/16	Fri 18/03/16	4;18;24	Tatiana

InGame	**25 days**	**Mon 07/03/16**	**Fri 08/04/16**		
Criar hierarquia Unity	5 days	Mon 07/03/16	Fri 11/03/16		Tatiana
GUI Logica de Jogo	15 days	Mon 21/03/16	Fri 08/04/16	5	Tatiana
Loading	15 days	Mon 14/03/16	Fri 01/04/16		Tatiana & Yuri
Networking	**145 days**	**Mon 02/11/15**	**Fri 20/05/16**		
Networking - criação da estrutura para utilizar frameworks de networking (abstrações)	25 days	Mon 02/11/15	Fri 04/12/15		Yuri
Networking - InGame / Logica de Jogo	25 days	Mon 18/04/16	Fri 20/05/16	32	Yuri
Networking - Lobby	5 days	Mon 02/05/16	Fri 06/05/16	32	Yuri
Networking - Chat	5 days	Mon 02/05/16	Fri 06/05/16	32	Yuri
Arte do GUI	**50 days**	**Mon 09/05/16**	**Fri 15/07/16**		
Desenhar icones	10 days	Mon 04/07/16	Fri 15/07/16		Tatiana
Conceito e arte final do GUI	10 days	Mon 04/07/16	Fri 15/07/16	17;23	Tatiana
Protótipo jogável v1 Milestone	0 days	Mon 09/05/16	Mon 09/05/16		
Mapa do jogo com peças	**50 days**	**Mon 04/04/16**	**Mon 13/06/16**		
Mapa do jogo com peças	**50 days**	**Mon 04/04/16**	**Fri 10/06/16**		
Construção do mapa igual ao boardgame	10 days	Mon 04/04/16	Fri 15/04/16		Yuri
Contrução Mapa com editor (inclui visualização e objetivos do mapa)	10 days	Mon 04/04/16	Fri 15/04/16		Yuri
Navegação no mapa	15 days	Mon 23/05/16	Fri 10/06/16		
*A**	15 days	Mon 23/05/16	Fri 10/06/16	42	Yuri
Protótipo jogável - networking + Gui + DB + mapa	0 days	Mon 13/06/16	Mon 13/06/16		
Lógica de jogo	**30 days**	**Mon 25/04/16**	**Fri 03/06/16**		
Combinações de Elementos	5 days	Mon 25/04/16	Fri 29/04/16		Yuri
Executar elementos (efeitos)	5 days	Mon 02/05/16	Fri 06/05/16	7;48	Yuri
Executar Abilidades	5 days	Mon 09/05/16	Fri 13/05/16	6	Yuri
Ciclo de Jogo	5 days	Mon 16/05/16	Fri 20/05/16		Yuri
Interação entre peças e mapa	**10 days**	**Mon 23/05/16**	**Fri 03/06/16**	**42**	
Implementação	10 days	Mon 23/05/16	Fri 03/06/16		Yuri
GUI	10 days	Mon 23/05/16	Fri 03/06/16		Tatiana
Art	**6 days**	**Mon 11/04/16**	**Mon 18/04/16**		
Art - Mundo	3 days	Mon 11/04/16	Wed 13/04/16		Tatiana
Art - Herois	3 days	Thu 14/04/16	Mon 18/04/16		Tatiana

Projecto completo - falta 3D e IA	0 days	Mon 06/06/16	Mon 06/06/16		
3D	**59 days**	**Tue 19/04/16**	**Fri 08/07/16**		
Modelação	29 days	Tue 19/04/16	Fri 27/05/16	57	Tatiana
Texturização	5 days	Mon 30/05/16	Fri 03/06/16	60	Tatiana
Rigging	10 days	Mon 06/06/16	Fri 17/06/16	61	Tatiana
Animação	15 days	Mon 20/06/16	Fri 08/07/16	62	Tatiana
Particulas de tudo	10 days	Mon 11/07/16	Fri 22/07/16	27;59	Tatiana
Playtesting - 1	5 days	Mon 13/06/16	Fri 17/06/16	58	Yuri;Tatiana
Inteligencia artificial	30 days	Mon 13/06/16	Fri 22/07/16		Yuri
Play					
Basic Strategy					
Outras coisas					
Projeto completo Milestone	0 days	Fri 22/07/16	Fri 22/07/16		
Escrita da Tese - Tati	15 days	Mon 25/07/16	Fri 12/08/16	64;59;36	Tatiana
Escrita da Tese - Yuri	20 days	Mon 25/07/16	Fri 19/08/16		Yuri
Tese concluída					

9.2 GAME REQUIREMENTS

9.2.1 FUNCTIONAL REQUIREMENTS

9.2.1.1 MENUS

1. The player shall be able to create an account.
2. The player shall be able to check its profile information.
3. The player shall be able to buy new heroes in the game.
4. The player shall be able to see each hero's abilities.
5. The player shall be able to see each hero's information
6. The player shall be able to create new hero presets.
7. The player shall be able to see each hero's presets.
8. The player shall be able to activate a preset.
9. The player shall be able to edit a preset.
10. The player shall be able to choose the team they want to play with.
11. The player shall be able to check its current account currency.
12. The player shall be able to see the game's shop.
13. The player shall be able to check its account statistics.
14. The player shall be able to change the game's video settings at any time.
15. The player shall be able to change the game's sound settings at any time.
16. The player shall be able to exit the game at any time.

9.2.1.2 IN-GAME

17. The player shall be able to see both teams' resources in the GUI at any time.
18. The player shall be able to see both teams' towers health information in the GUI at any time.
19. The player shall be able to see both teams' scores in the GUI at any time.
20. The player shall be able to know which team is playing at any time.
21. The player shall be able to know which team belongs to them in the GUI at any time.
22. The player shall be able to see the last five actions in the game at any time.
23. The player shall be able to see at the game's duration in the GUI any time.
24. The player shall be able to see the status of both teams' heroes in the GUI at any time.
25. The player shall be able to see the hero's ability bar for both teams in the GUI at any time.
26. The player shall be able to see the element combinations of both teams' heroes in the GUI at any time.
27. The player shall be able to see the health of both teams' heroes in the GUI at any time.
28. The player shall be able to see the activated element powers for both teams' element zone in the GUI at any time.
29. The player shall be able to see in the GUI both team's number of element powers stored at any time.
30. The player shall be able to know which hero is currently playing at any time.
31. The player shall be able to see each owned element power stored.
32. The player should be able to see its hero's turn actions in the GUI at any time.
33. The player shall be able to see more information about their hero's ability.

34. The player shall be able to see more information about their owned element powers.
35. The player shall be able to use a hero's ability.
36. The player shall be able to use an owned element power.
37. The player shall be able to attack a tower.
38. The player shall be able to unlock new abilities.
39. The player shall be able to fill ability's hero targets.
40. The player shall be able to fill ability's hexagon targets.
41. The player shall be able to fill element power's hero targets.
42. The player shall be able to fill element power's tower targets.
43. The player shall be able to fill element power's hexagon targets.
44. The player shall be able to see a timer to use a counter ability.
45. The player shall be able to see a timer to activate an element.
46. The player shall be able to see a turn timer.
47. The player shall be able to resurrect a hero by choosing an available hexagon on the map.
48. The player shall be able to spawn a hero by choosing an available hexagon on the map.
49. The player shall be able to move a hero by choosing an available hexagon on the map.
50. The player shall be able to see the available movements on the map.

9.2.2 NON-FUNCTIONAL REQUIREMENTS

1. The game shall have a scalable interface.
2. The game shall have menus that are easy to navigate.
3. The game shall have multi-language support incorporated.
4. The game shall be able to run on computers with older hardware.
5. The game shall be able to give visual feedback to both teams of any action performed by a player.
6. The GUI shall be able to fully integrate the game mechanics.

9.2.3 GAME ABILITIES

Table 24 – All game's ability cards

CodeName	**Name**	**Description**	**Range**	**Cost**	**Cycle**
Ghora1 1	Ghora 1	Ghora deals 1 damage and an additional 1 damage for every 2 health he has lost.	1	1RC	
Ghora2 2	Ghora 2	Ghora deals 1 damage and an additional 1 damage for every 3 health he currently has.	1	1RC	
Ghora3 3	Ghora 3	Ghora jumps up to 3 tiles and deals 1 damage to all enemies around the landing tile.	-	1RC	
Ghora4 4	Ghora 4	Redirects to Ghora an ability that would affect a friendly character next to him.	1	1RC	x
Ghora5 5	Ghora 5	[COUNTER] Ghora reflects the damage of an ability targeting him. Every other effect is canceled.	-	1RC	
Ghora6 6	Ghora 6	Ghora deals 2 damage to all enemies and 2 healing to all allies within 1 tile.	-	1RC	

GhoraU1 7	Ghora U1	Ghora becomes immortal. Damage from abilities or ultras that would kill him, drains resources instead of health.	-	2RC	X
GhoraU2 8	Ghora U2	[COUNTER] Ghora takes the damage that would hit an ally and heals them by half the damage that Ghora takes.	1 to 4	2RC	
KULLA					
Kulla1 9	Kulla 1	Kulla deals 2 damage and if the target dies, all characters of the same team within 1 tile will take 1 damage	2 to 3	1RC	
Kulla2 10	Kulla 2	[COUNTER] Kulla cancels the next damaging ability that would hit her	-	1RC	
Kulla3 11	Kulla 3	Choose one: Kulla shoots an explosive trap that immediately deals 1 damage to all characters around the target tile. Kulla arms 2 traps on the map that deal 2 damage once triggered. These traps can not be placed on tiles next to or with a character on them.	2 to 3	1RC	X
Kulla4 12	Kulla 4	Kulla deals 2 damage and moves the target 1 tile closer to her.	4	1RC	
Kulla5 13	Kulla 5	Kulla rolls over 2 tiles and applies her element to all characters on the tiles around her landing.	-	1RC	
Kulla6 14	Kulla 6	Kulla deals 1 damage to the target. If there is another character of the same team next to the target, Kulla can incapacitate it for one turn	2 to 3	1RC	
KullaU1 15	Kulla U1	Kulla deals 3 damage. Spending 1 additional resource will increase the range of this ability by 1 tile, up to a maximum of 3	2 to 3	2RC	
KullaU2 16	Kulla U2	Kulla deals 3 damage to the target closest to her and 2 damage to the target furthest from her.	2 to 3	2RC	
LIPP					
Lipp1 17	Lipp 1	[COUNTER] Lipp cancels an ability targeting her or an ally. If Lipp is the target of the ability, she deals 1 damage to the attacker.	1 | -	1RC	
Lipp2 18	Lipp 2	Choose one: Lipp switches places with her target and then pushes every character within a 2 tile radius by 2 tiles Lipp switches places with her target and then pulls every character within a 2 tile radius by 2 tiles	1 to 2	1RC	
Lipp3 19	Lipp 3	Lipp deals 3 damage to the target. If the target dies, Lipp deals an additional 2 damage to another target of the same team within 2 tiles.	1 to 2	1RC	
Lipp4 20	Lipp 4	Lipp deals 1 damage to the target. If the target dies, Lipp deals an additional 4 damage to another target of the same team within 2 tiles.	1 to 2	1RC	
Lipp5 21	Lipp 5	Lipp deals 2 damage to the target and an additional 2 damage to another target of the same team within 2 tiles.	1 to 2	1RC	
Lipp6 22	Lipp 6	Once activated, every time Lipp takes damage, the attacker takes 1 damage in addition to the damage dealt to Lipp. Spends 1 resource for each damage returned.	-	1RC	X
LippU1 29	Lipp U1	Lipp reverses the state of all her abilities. Used abilities become unused and unused abilities become used.	-	2RC	

LippU2 23	Lipp U2	Copies an Ultra from the target. Lipp may copy another ultra to change the currently active ultra Ghora: GhoraU1 - 24 {7} Kulla: KullaU2 - 25 {16} Loper: LoperU2 - 26 {37} Malik: MalikU2 - 27 {45} Wang: WangU1 - 28 {52}	1 to 5	2RC	
LOPER					
Loper1 30	Loper 1	Loper exchanges his health with the targets' current health. Neither Loper nor the target can exceed their own maximum health.	1 to 2	1RC	
Loper2 31	Loper 2	[COUNTER] Loper can sacrifice himself and die instead of an ally within range. The ally stays alive with Loper's current health. Cannot exceed the targets' maximum health.	1 to 5	1RC	
Loper3 32	Loper 3	Loper deals 2 damage and is healed for 2 health.	1 to 2	1RC	
Loper4 33	Loper 4	Loper deals 2 damage and if the target dies, Loper can use one of its abilities.	1 to 2	1RC	
Loper5 34	Loper 5	Loper transfers 1 health from all heroes 2 hexes away from the target to the target.	1 to 2	1RC	
Loper6 35	Loper 6	The damage taken by an ally character in a tile next to Loper is equality distributed between the two. Damage is rounded down.	0	1RC	X
LoperU1 36	Loper U1	Loper resurrects an ally and places them on a tile next to him.; Loper resurrects an ally and places them on a tile next to him. Loper sacrifices 4 health instead of paying resources.	-	2RC	
LoperU2 37	Loper U2	Lopper resurrects an ally under his control for 1 turn. The ally will resurrect on a tile next to Lopper and for each resource they would spend, Lopper takes 1 damage instead. The ability will wear off if Lopper would be killed by this damage.	-	2RC	
MALIK					
Malik1 38	Malik 1	Malik hides in the shadows and can't use Damage Abilities while hidden. Malik can only be targeted by element powers and Utility abilities cost 1 more resource. When Malik leaves the shadows, he deals 2 unavoidable damage to his target.	- / 1	1RC	X
Malik2 39	Malik 2	Malik deals 2 Damage. If he doesn't have Malik 1, he deals 1 additional damage.	1	1RC	
Malik3 40	Malik 3	Malik can move 1 tile to reach the target and deals 1 damage. Malik deals 2 damage if he doesn't need to move	1 to 2	1RC	
Malik4 41	Malik 4	Choose one: Malik picks the target's pockets and places one of their damage abilities on cooldown.; Malik protects himself from an ability during the enemy turn.	1	1RC	
Malik5 42	Malik 5	Choose one: Malik removes a trap within range.; Restore 1 health point to Malik.	1	1RC	
Malik6 43	Malik 6	Malik creates a smoke cloud around him that prevents all characters from using or being targeted by abilities.	-	1RC	X
MalikU1 44	Malik U1	Malik deals 3 damage and an additional 1 damage for each character within 2 tiles of him.	1	2RC	
MalikU2 45	Malik U2	Malik prepares to instantly kill an opponent, immediately ending his turn. During the next turn, Malik can instantly kill any target within range. This ability will reset automatically even if no target is killed.	1	1RC / 2RC	-

WANG					
Wang1 *46*	Wang 1	Wang deals 2 damage on 3 tiles around to him.	1	1RC	
Wang2 *47*	Wang 2	Wang uses his chains to deal 1 damage and moves the target up to 2 tiles.	2 to 3	1RC	
Wang3 *48*	Wang 3	Wang deals 2 damage and if the targets dies, Wang is healed by 2.	1	1RC	
Wang4 *49*	Wang 4	Wang moves to the closest tile next to the target and deals 2 damage. If there is another target of the same team within 2 tiles of the first, Wang will move to the closest tile next them and deal 1 damage.	2	1RC	
Wang5 *52*	Wang 5	[COUNTER] Wang redirects the damage from an ability that would hit him to another target. All secondary effects are canceled.	1 to 2	1RC	
Wang6 *51*	Wang 6	Wang changes the position of two characters. Both characters must be within Wang's range.	1 to 2	1RC	
WangU1 *52*	Wang U1	Wang deals 2 damage and an additional 2 damage if the target has less than half of its maximum health	1	2RC	
WangU2 *53*	Wang U2	Wang creates a clone of himself. Wang and the clone move separately but share abilities and their cooldowns. As soon as the clone takes damage it disappears.	1 to 3	2RC	X

9.2.4 GAME ELEMENT POWERS

Table 25 – All game's element power cards

Nome	**Description**	**Time to Use**
F1	Deal 1 damage to the last character of the opposing team that played and 1 damage to another one.	Instant
F2	Deal 1 damage to the last character of the opposing team that played and 2 damage to another one.	Instant
F3	Deal 1 damage to the last character of the opposing team that played and 3 damage to another one.	Instant
F4	Deal 1 damage for each utility ability the character currently playing has. Deals at least 1 damage.	Instant
F5	Deal 3 damage plus, 2 for each character of the same team in an adjacent tile to the target.	Instant
F6	Choose one: Draw a Water elemental power Draw a Lightning elemental power Deal 1 damage to a character.	Instant
F7	Choose one: Draw a Wind elemental power. Draw an Earth elemental power. Deal 1 damage to a character.	Instant
F8	Deal 1 damage to a character for each character controlling Fire in the game. Deals at least 1 damage.	Instant
F9	Choose one: Restore 1 resource for each damage ability your team has used. Deal 1 damage to a character.	Instant

F10	Deal 1 damage plus, 1 for each additional ability the opposing team has unlocked.	Instant
F11	Deal 1 damage to all characters of the opposing team. Deals 1 more if one of your bases has been destroyed.	Instant
	WATER	
A1	Heal 1 health to the last character of your team that played and 1 health to another one.	Instant
A2	Heal 1 health to the last character of your team that played and 2 health to another one.	Instant
A3	Heal 1 health to the last character of your team that played and 3 health to another one.	Instant
A4	Heal 1 health for each damage ability the character currently playing has. Heals for at least 1 health.	Instant
A5	Heal 3 health plus, 1 health for each character of the same team in an adjacent tile to the target.	Instant
A6	Choose one: Draw a Fire elemental power. Draw a Wind elemental power. Heal 1 Health to a character.	Instant
A7	Choose one: Draw a Lightning elemental power Draw an Earth elemental power Heal 1 health to a character.	Instant
A8	Heal 1 health to a character for each character controlling Water in the game. Heals for at least 1 health.	Instant
A9	Heal 1 health to all characters of your team for each ability of the opponent team has and 2 health for each unlocked ultra	Instant
A10	Heal 1 health plus, 1 health for each additional ability the opposing team has unlocked	Instant
A11	Heal 1 health to all characters of your team. Heal 1 more health to each character if one of your bases has been destroyed.	Instant
	WIND	
V1	Heroes from the opposing team that are inside the Base area of this team will suffer damage equal to the Base's damage. At the start of the round every hero from the opposing team is moved outside of the Base area and if, they reenter it during this round they will take damage equal to the Base's Damage.	Element Zone
V2	Heroes from the opposing team are moved outside of this team's Base areas and are pushed away by 1 + Base's damage tiles.	Save for later
V3	You can move 2 heroes for up to 3 tiles each.	Instant
V4	After being damaged by an ability, a hero can move as many tiles as the damage taken	Element Zone
V5	Choose one: Draw a Water elemental power. Draw an Earth elemental power. Heal a Base by 1 health. Move 1 of your team's heroes by up to 3 tiles.	Instant
V6	Choose one: Draw a Lightning elemental power. Draw a Fire elemental power. Heal a Base by 1 health. Move 1 of your team's heroes by up to 3 tiles.	Instant

V7	Choose one: Move 1 of your team's heroes up to 3 tiles for each hero that controls Wind in the game. Heal a Base by 1 health for each hero that controls Wind in the game.	Instant
V8	During this round, heroes from the opposing team can't stay inside this team's Base areas while those Bases are alive. However, they can use these areas to move. If any heroes remain inside the area after their movement, they will be moved to the closest tile outside the of the area.	Element Zone
V9	The hero playing can move up to 6 tiles during its first movement phase, instead of 3.	Save for later
V10	Heroes from this team can't be targeted by abilities from the opposing team while standing in their own Base area.	Element Zone
V11	Move one hero up to 12 tiles. If you choose to move a hero from the opposing team, they must be moved to the center of the map. You can move 1 additional hero if one of your Bases has been destroyed.	Save for later
	EARTH	
T1	The heroes from the team that uses this power deal 1 more damage to Bases but, also take 1 more damage.	Element Zone
T2	Deal 2 damage to a Base if there are two heroes from your team inside that Base area.	Save for later
T3	Choose one: After killing a Base, fully heal the hero that killed it. One of your heroes takes no damage when attacking a Base.	Save for later
T4	When attacking a Base, you can deal 2 more damage to it.	Save for later
T5	Choose one: Draw a Lightning elemental power. Draw a Wind elemental power. Deal 1 damage to a Base if your team has at least 1 hero inside its area.	Instant
T6	Choose one: Draw a Water elemental power. Draw a Fire elemental power. Deal 1 damage to a Base if your team has at least 1 hero inside its area.	Instant
T7	If the hero playing is inside an opposing Base area, he can deal 1 more damage for each hero controlling Earth in the game.	Save for later
T8	If the hero currently playing attacks a Base, they will deal 1 more damage and take 1 less damage.	Save for later
T9	During 1 round, the heroes from your team are immune to Base damage.	Element Zone
T10	If your heroes are inside the opposing Base area, they can't be targeted by enemy abilities.	Element Zone
T11	The hero currently playing can teleport to a Base area and deal 1 damage to it.	Save for later
	LIGHTNING	
R1	You can use an ability for free, without paying any cost or activating its cooldown.	Save for later
R2	Inverts the effect of the ability being played. Healing now damages and damage now heals	Save for later
R3	Steals an elemental power from the opposing team.	Save for later
R4	Reduces the cost of one ultra by 2 resources.	Save for later

R5	Forces the opposing team to swap the last unlocked ability of the targeted hero.	Instant
R6	During one round both teams can't use more abilities than the team that has the least abilities.	Element Zone
R7	Choose one: Draw an Earth elemental power Draw a Fire elemental power Deal 1 damage to a Hero Store and use it when inside a Base area to deal 1 extra damage.	Instant
R8	Choose one: Draw a Water elemental power Draw a Wind elemental power Heal 1 health to a Hero Move a Hero up to 3 hexes.	Instant
R9	Choose one: Deal 1 damage to a character and to a Base. Heal 1 health to a character and to a Base.	Instant
R10	Increases the cost of the opposing team's abilities by 1 Resource. If you have unlocked less abilities than the opposing team, the cost is increased by 1 more resource.	Element Zone
R11	Unlocks 1 ability for every 3 abilities the opposing team has.	Instant

9.3 GAME STORY

9.3.1 GAME CLASSES

Figure 115 - Wizard world and weapon inspiration: (A) Crystal caves; (B) Dalaran [43]**; (C) Crystal cave cities; (D) Surammar city** [43]**; (E) Wizard weapon inspirations.**

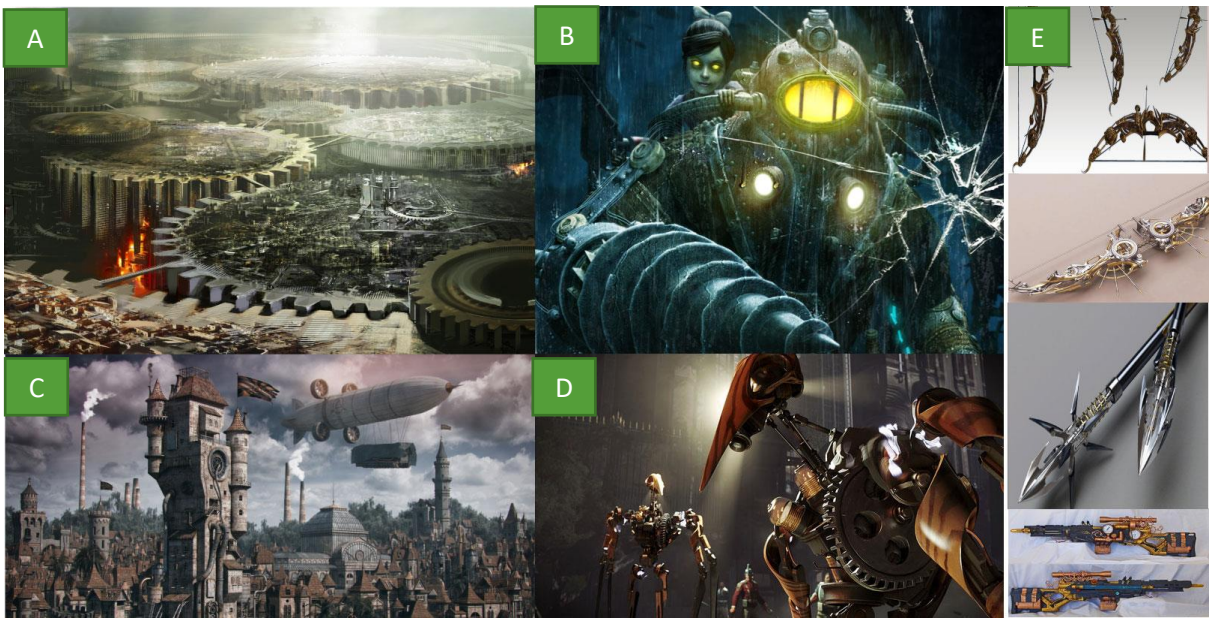


Figure 116 - Pathfinders world and weapon inspiration: (A) Cog Wheel cities; (B) Bioshock [128]**; (C) Steampunk inspired cities; (D) Dishonored 2** [129]**; (E) Pathfinder weapon inspirations.**

Figure 117 - Mender world and weapon inspiration: (A) and (B) Maleficent forest (representing both dead and alive worlds); (C) Forbidden forest from Harry Potter; (D) Mirkwood forest from Lord of the Rings; (E) Mender weapon inspirations.

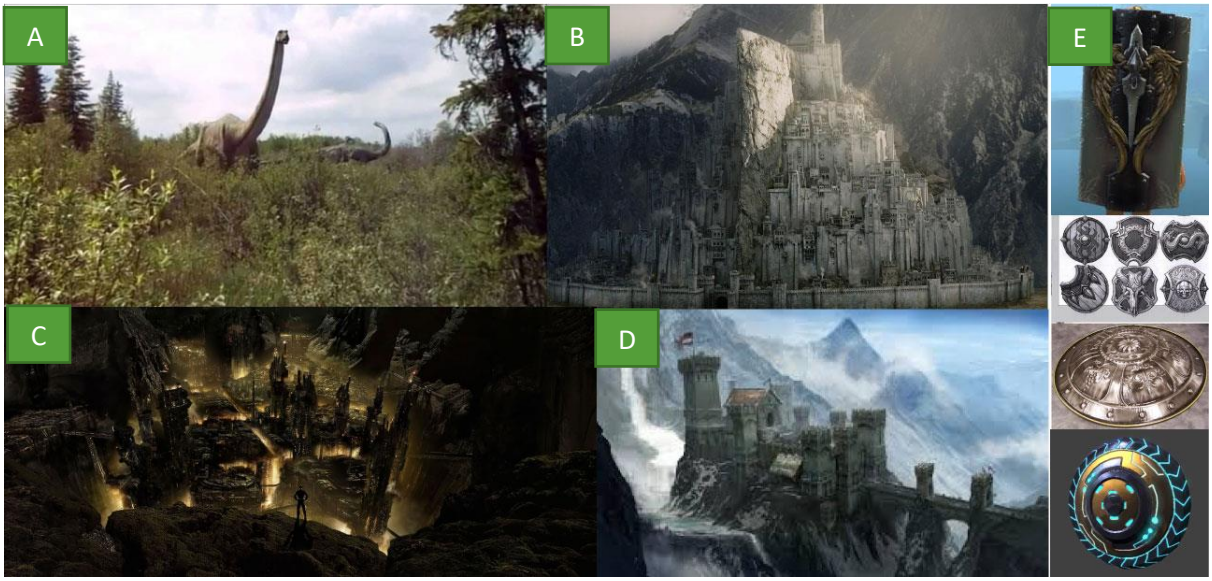


Figure 118 - Guardian world and weapon inspiration: (A) Jurassic park dinosaurs; (B) Minas Tirith from Lord of the Rings; (C) and (D) Cities/castle on top of mountains; (E) Guardian weapon inspirations.

9.3.2 CLANS

Table 26 – Possible names for the other game clans – final choice in bold

Clan's Character	Traits, beliefs and mostly known values	Translated Traits, beliefs and mostly known values
Wang	Discipline	Jìlü
	Many people	Zhòngduö
	Strategists	Zhanhüe Jia
	Technologically advanced	**Mingzhi**
	Virtuous	You Dàodé
Loper	Values life	Waarde lewe
	Ancestors	**Voorváders**
	Animal force	dier krag
	Animal spirit	dier gees
	Community	gemeenskop
Lipp	Brave/courageous	hrauster/bitr/froekn/**Hugrakkur**
	Strong	Kroptugr/Sterkr
	Ancestors	for-eldra
Kulla	Free people	Liri
	Heroic	Heroizem
	Resistant	afatgjatë
	Martyr	**Deshmor**
Ghora	Equality	Bnäbari
	Sacrifice	Kuräbani
	Humility	Nimaratä
	Generous	khul'hä dila
	Honest	**Imanadari**

9.3.3 CLANS AESTHETICS

Figure 119 - The Mingzhi Clan, where the character Wang comes from.

Figure 120 - The Voorváders Clan, where the character Loper comes from.

Figure 121 - The Hungrakkur Clan, where the character Lipp comes from.

Figure 122 - The Deshmor Clan, where the character Kulla comes from.

Figure 123 - The Imandari Clan, where the character Ghora comes from.

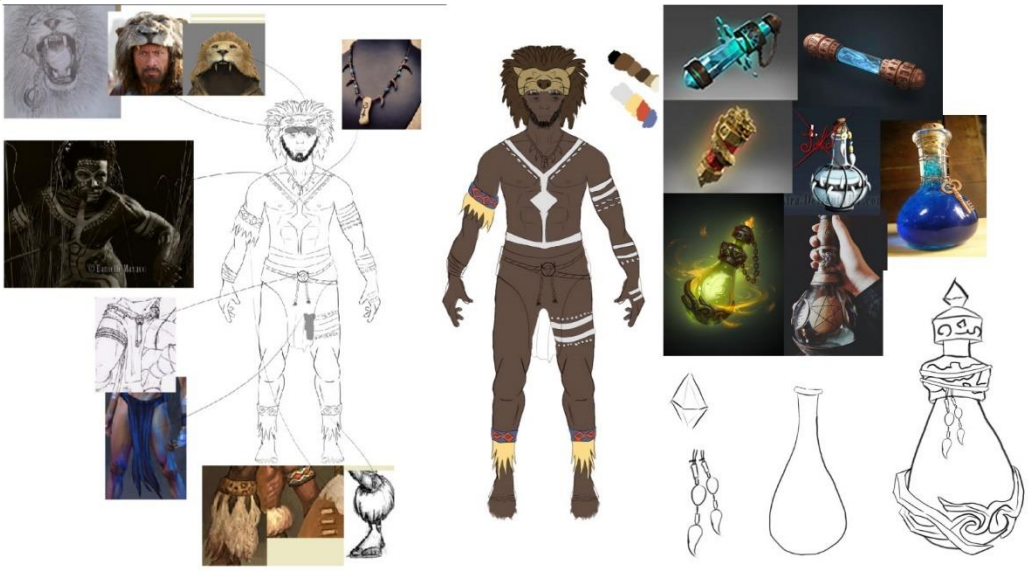
9.3.4 CHARACTERS

Background: As Lipp grew up, she was always drawn to her ancestor's history and knowledge. When she was old enough, she started studying her ancestor's runes. As a consequence, she learned arcane magic. She now manipulates two schools of magic, evocation and conjuration. But, that was not enough for Lipp, she grew thirsty for knowledge and decided to leave her home and explore the world. She learned more about her magic in the Wizard planet and went to Intheris to practice her new discoveries.

Abilities:

1. Lipp1 – Lipp silences the enemies, making them unable to cast spells and interrupting any spells casted. She unleashes a wave of arcane energy, hitting her target and protecting her alies.
2. Lipp2 – Lipp teleports herself to the middle of friends and foes, pushing or pulling everyone around her.
3. Lipp3 – Lipp sends a bolt of energy to the target. If it dies, the same bolt of energy hits another target.
4. Lipp4 - Lipp sends a bolt of energy to the target. If he dies, the same bolt of energy divides in four smaller balls and hit the other target.
5. Lipp5 - Lipp sends a bolt of energy to each target.
6. Lipp6 – Lipp creates a shield of arcane magic around her.
7. LippU1 – Lipp uses all her strength to draw a wave of energy to her
8. LippU2 – Lipp uses her staff to create a laser beam, copying the target's ability.

Figure 124 - Lipp concept inspiration and story



Background: Lopper was born in the Zulu tribe. There was always someone behind him leading the right path and never letting him diverge from it. As time went by, Loper's interest fell to the arts of healing, becoming a famous shaman among his tribe. One day, one of his dearest friends was wounded by a mythical creature which, had never been seen before. Loper read every book around him, tried every single bit of magic he knew but, he still could not heal his wounded friend. His friend was dying and he felt powerless, there was nothing he could do to help him. Then, Loper remembered that when he was a child, one of the eldest of his tribe used to tell stories about blood magic, that could manipulate life but, he would also say that this type of magic was banished a long time ago. He searched in his books about this unspeakable magic and found an old scroll describing how to heal any wound. Against all his beliefs but determined to help his friend he tried that spell and crossed a line that could never be uncrossed. Lopper became a necromancer. As a shaman he had nature magic but now, as a necromancer, he had black and blood magic. Since he was not a shaman anymore, he decided to leave his tribe and join others like him. Eventually he reached the Mender planet.

Abilities:

1. Loper1 – Loper sends one of his shadowy magic hands to the target, which still their health and trade it for Loper's health.
2. Loper2 – Loper sends his soul as a bolt of shadow to the target and dies.
3. Loper3 – Loper uses his magic to lift his target and squeezes him. Dark particles fall all around the target and go to Loper.
4. Loper4 – Loper sends his flask to the target, creating a dark cloud. If the target dies, he resurrects near loper as a ghost.
5. Loper5 – Loper creates a bolt of shadow from everyone around the target and sends them to it
6. Loper6 – Loper creates a ring of shadow around Loper and its allies. Any damage taken by his allies creates a bolt of shadow and hits Loper.
7. LoperU1 – Loper throws his flask on the floor next to him. An ally is resurrected on that spot.
8. LoperU2 – Loper resurrects his target and controls him using shadow strings coming from his hand.

Figure 125 – Loper concept inspiration and story.

Background: Wang used to be a monk. But he was different from any other monk, who are humble and honorable. Wang liked to break all the rules, it was like a challenge for him. As he grew up he became cocky and the monks in his monastery finally kicked him out. He was only allowed to carry one weapon. Wang had no choice but to take with him his favorite weapon, his chain whip. Later he decided to add small blades to each side of the chains making them deadlier. Today, Wang is another mercenary, he fights for the highest bidder and that is how he found himself in Intheris. After coming back from a mission, he was approached by a stranger that asked him if he wanted to become a true master of the chain whip. To show the old man he did not need any more training, he entered the assassination trials.

Abilities:

1. Wang1 – Wang spins his chain around him, hitting everyone in front of him
2. Wang2 – Wang uses his chain to pull his target
3. Wang3 – Wang attacks with the dagger on the tip of the chain. When healed, Wang shouts.
4. Wang4 – Wang attacks each target with one of his chain's tips as he runs towards the last target.
5. Wang5 – Wang dodges the attack
6. Wang6 – Wang sends each tip of his chain to one of his targets and changes their places
7. WangU1 – Wang spins his chains and attacks his target with them from the top to the end of its body. If the target is low on health, Wang kicks his chain and attacks the target again.
8. WangU2 – Wang spins his chains creating a cloud of dust. In the end, Wang and his clone appear.

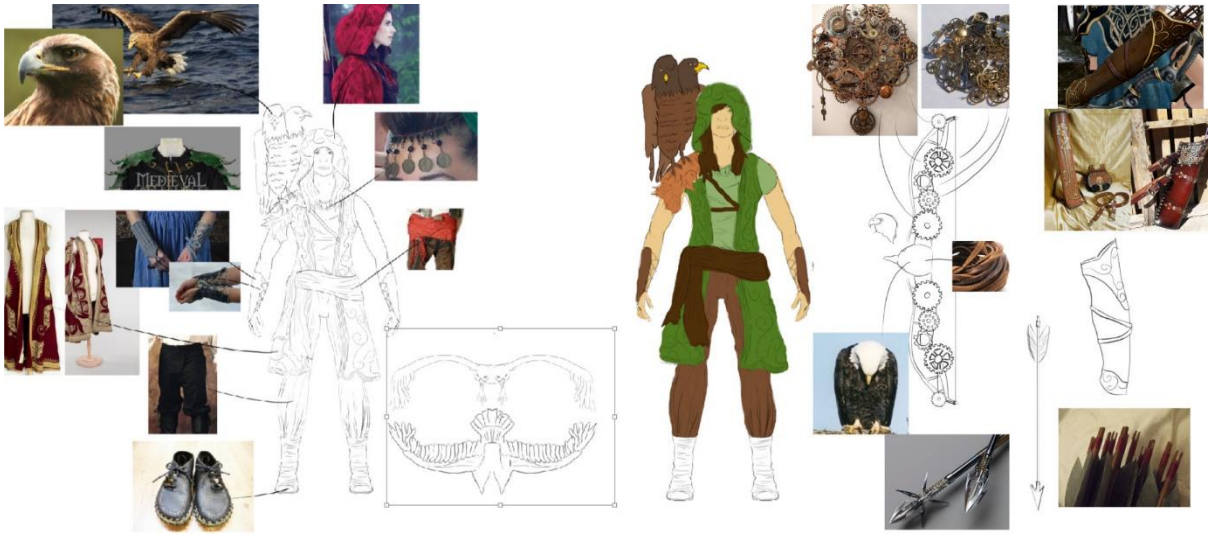
Figure 126 - Wang concept inspiration and story.

Background: Ghora grew as an orphan. Never knowing what had happened to his parents he always felt he had to defend his friends at the orphanage so they would not suffer from the same fate. He always had a sense of honor and loyalty, defending the weak and making justice prevail. He always treasured shields and loved to play with his friends as knights and thieves. As he grew older, he became a great master of the sword and shield, becoming one of the most important Sikh of all times. Just like when he was a child, Ghora remains loyal and honorable. Because of this he went to the Protector planet, to learn more fighting techniques to be able to always defend his friends.

Abilities:

1. Ghora1 – Ghora attacks its target and his hands glow red
2. Ghora2 – Ghora attacks its target and his hands glow blue
3. Ghora3 – Ghora jumps and when he arrives at his destination he creates a wave of dust and cracks in the ground.
4. Ghora4 – Ghora shouts and a defensive shield shows up on his head
5. Ghora5 – Ghora reflects the attack using his shield
6. Ghora6 – Ghora shouts creating a shock wave
7. GhoraU1 - Ghora shouts and creates a three shields around him, protecting him from any damage.
8. GhoraU2 – Ghora charges to his ally, taking the damage and healing them.

Figure 127 - Ghora concept inspiration and story.



Background: Kulla grew up in the vast vales of Albania. She was always very curious and from an early age she always carried her bow and arrow. One day, her village was attacked and she had to run to the forest. To survive, Kulla had to learn how to use her bow to hunt. When the winter came, she had to find shelter and found an abandoned eagle egg. She looked up, looked in the distance and there were no eagle nests around her. She decided to keep the egg warm and after a few days an eagle came out. This tiny eagle had two heads and was scared and lonely. Kulla decided to adopt it and they grew up together, being inseparable now. As a curious explorer, she could not resist the opportunity to visit and explore a new world, the Pathfinder planet.

Abilities:

1. Kulla1 – Kulla shoots an arrow to her target. If the target dies, a wave of small arrows explode around the target.
2. Kulla2 – Kulla puts a barrier around her, protecting her from taking damage
3. Kulla3 - Kulla shoots two arrows and places a trap in each chosen position.
4. Kulla4 – Kulla shots an arrow with a rope, pulling the target closer to her
5. Kulla5 – Kulla dives
6. Kulla6 – Kulla shoots two arrows. The second arrow creates a flash bang stunning the target
7. KullaU1 – Kulla shoots an arrow. This arrow grows as it gets closer to the target
8. KullaU2 – Kulla shoots an arrow to the first target. Then the arrow starts jumping, attacking the second target and then the first target again.

Figure 128 - Kulla concept inspiration and story.

9.4 INTERFACE IMPLEMENTATION

9.4.1 MENUS NAVIGATION TREE

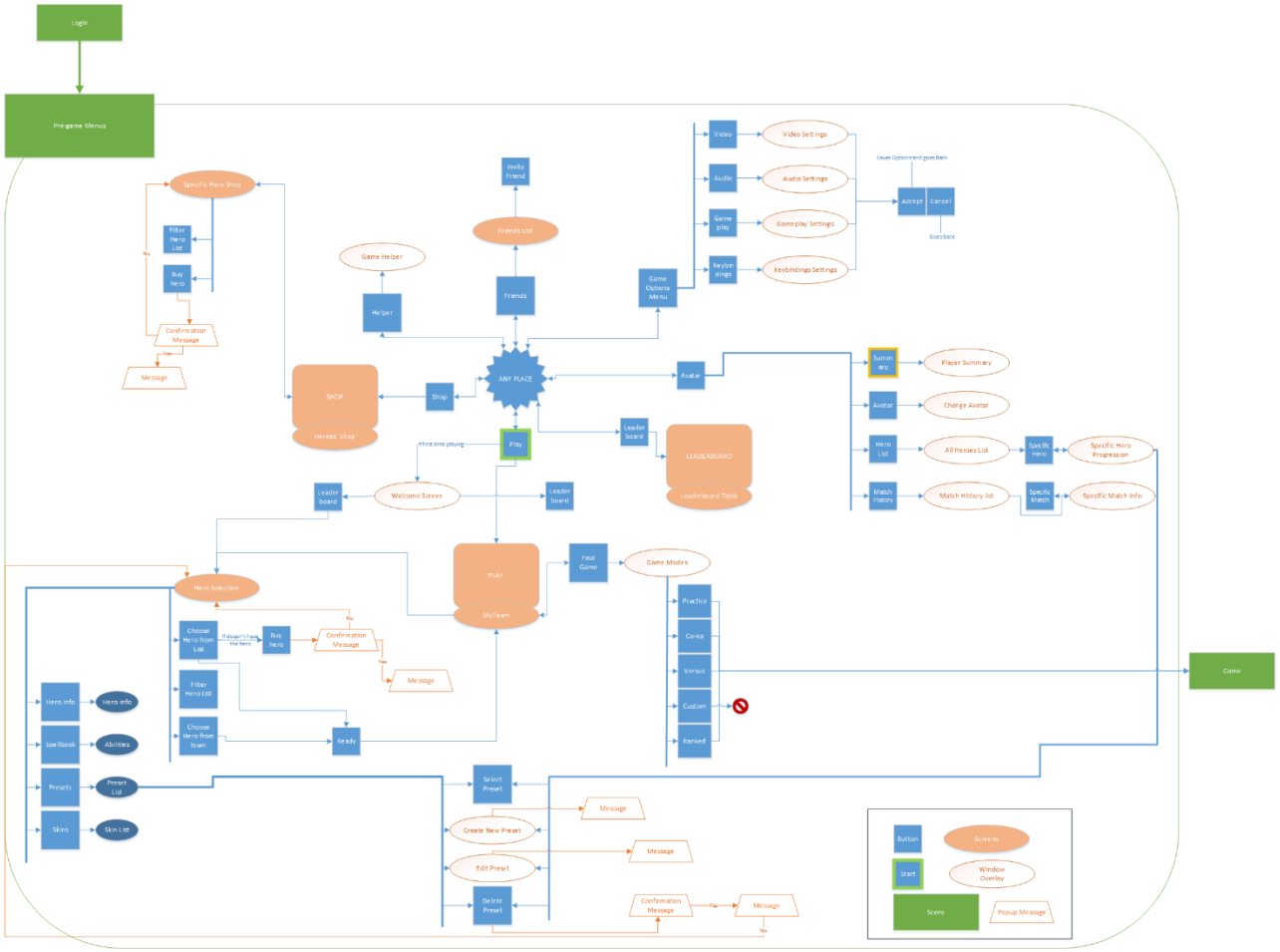


Figure 129 – Full menus navigation tree.

9.4.2 INTERFACE ICONS

Figure 130 – All hero's ability icons created/edited in illustrated using the Game icons library [132].

9.4.3 USER TESTS GUIDE

O jogo está ainda em desenvolvimento, como tal, poderão existir falhas na interface. Por vezes poderá parecer que não é possível completar rápida ou facilmente uma tarefa. Se isto acontecer, é uma falha que estás a ajudar a corrigir e não te deverás sentir mal por tal.

Para ajudar a identificar elementos que causam confusão deverás dizer em voz alta tudo o que estás a pensar e a fazer. Também para me ajudar, todos os teus movimentos na interface e também a tua voz serão gravados. **Concordas?**

Existem ao todo 14 tarefas. Eu vou ler uma tarefa e só depois de eu terminar é que deverás executá-la. Caso precises que eu a repita, é só me pedir. Vamos então começar, **estás pronto?**

1. Acabaste de instalar o jogo e de te registar. Vais agora entrar pela primeira vez no jogo. Como já conheces este género de jogos, vais saltar o tutorial. Nas definições de vídeo ajusta a resolução para 1920x1080 e nas definições de áudio desliga o som.
2. Sempre gostaste do estilo de jogo de assassinos. Como tal, um herói assassino irá ser a tua primeira compra. Filtra a lista de heróis de modo a veres só os assassinos**.**
3. Escolhe o assassino que mais gostares e vê as suas habilidades, na lista de habilidades.
4. Mal podes esperar por ver este assassino em acção, compra-o para o usares no teu primeiro jogo.
5. Como tens de ter uma equipa de três heróis para poder competir com os teus amigos, compra mais dois heróis à tua escolha. Para tal, remove os filtros na lista de heróis para os poderes ver a todos.
6. Para mais informações sobre algum termo dentro do jogo, poderás consultar a ajuda**.** Consulta a ajuda e aprende sobre os elementos.

7. As habilidades dos heróis são divididas em três tipos de polaridades. Todos os heróis têm acesso a todas as polaridades, mas o seu número e ordem variam entre tipos de heróis. Para poderes usar uma habilidade, terás de a colocar numa posição que tenha a mesma polaridade. Para simplificar o teu progresso dentro de jogo, podes criar uma lista de habilidades predefinidas que são desbloqueadas sequencialmente conforme os teus heróis progridem. Usa a aba *preset* de um dos teus heróis para criar uma lista deste tipo. Esta lista inclui ainda itens cosméticos para a tua personagem.
8. Como não estás satisfeito/a com o teu avatar muda-o para um que gostes mais no teu perfil.
9. Enquanto estavas no teu perfil, reparaste que existe uma secção dedicada aos heróis. Vê que informação adicional podes encontrar sobre o herói para o qual criaste uma *preset* anteriormente.
10. Cria um novo *preset* para este herói e ativa-o.
11. Apaga o primeiro *preset* que criaste
12. Vai à loja e vê que heróis ainda se encontram à venda.
13. Agora que estás pronto**,** atribui um elemento a cada um dos teus heróis.
14. Procura um jogo para praticares as tuas habilidades contra inteligência artificial.

9.4.4 GAME DEDICATION QUESTIONAIRE

About You

Genre ☐ Male ☐ Female

Age ☐ 13 - 18 ☐ 19 - 24 ☐ 25 – 34 ☐ 35-49 ☐ 50+

Occupation _____

About the Interface

Overall reaction to the interface

Terrible	1	2	3	4	5	6	7	8	9	10	Wonderful
Difficult	1	2	3	4	5	6	7	8	9	10	Easy
Frustrating	1	2	3	4	5	6	7	8	9	10	Satisfying
Inadequate Power	1	2	3	4	5	6	7	8	9	10	Adequate Power
Dull	1	2	3	4	5	6	7	8	9	10	Stimulating
Rigid	1	2	3	4	5	6	7	8	9	10	Flexible

Reading characters (font) on the screen was..

Difficult	1	2	3	4	5	6	7	8	9	10	Easy

Highlights in the game simplified the tasks

Strongly Disagree	1	2	3	4	5	6	7	8	9	10	Strongly Agree

Organization of the information was

Confusing	1	2	3	4	5	6	7	8	9	10	Clear

The amount of information presented on the screen was:

Inadequate	1	2	3	4	5	6	7	8	9	10	Adequate

The arrangement of the information on the screen was:

Inadequate	1	2	3	4	5	6	7	8	9	10	Adequate

The navigation sequence was:

Confusing	1	2	3	4	5	6	7	8	9	10	Clear

Going back to the last screen was:

Impossible	1	2	3	4	5	6	7	8	9	10	Easy

Gamer Dedication

I play games over many long sessions

Strongly Disagree	1	2	3	4	5	Strongly Agree

I discuss games with friends/bulletin boards

Strongly Disagree	1	2	3	4	5	Strongly Agree

I have comparative knowledge of the industry

Strongly Disagree	1	2	3	4	5	Strongly Agree

I am much more tolerant of frustration

Strongly Disagree	1	2	3	4	5	Strongly Agree

I have indications of early adoption behavior (being the first to adopt latest in trends or fashions)

Strongly Disagree	1	2	3	4	5	Strongly Agree

I have the desire to modify or extend games in a creative way

Strongly Disagree	1	2	3	4	5	Strongly Agree

I am technology savvy

Strongly Disagree	1	2	3	4	5	Strongly Agree

I have the latest high-end computer/consoles

Strongly Disagree	1	2	3	4	5	Strongly Agree

I play for the exhilaration of defeating (or completing) the game

Strongly Disagree	1	2	3	4	5	Strongly Agree

I have a hunger for gaming-related information

Strongly Disagree	1	2	3	4	5	Strongly Agree

I am engaged in competition with myself, the game and the others

Strongly Disagree	1	2	3	4	5	Strongly Agree

I am willing to play

Strongly Disagree	1	2	3	4	5	Strongly Agree

I prefer games that have depth and complexity

Strongly Disagree	1	2	3	4	5	Strongly Agree

I started playing games at a young age

Strongly Disagree	1	2	3	4	5	Strongly Agree

I prefer violent/action games

Strongly Disagree	1	2	3	4	5	Strongly Agree

9.4.5 USER TESTS – FIRST ITERATION RESULTS

Table 27 – Number of wrong clicks on the first user test iteration for each task (it is considered a wrong click, a click that is not meaningfull to complete the task (user should not have pressed it))

Task N	User 1 N Clicks	User 2 N Clicks	User 3 N Clicks	User 4 N Clicks	User 5N Clicks	User 6 N Clicks	User 7 N Clicks	User 8N Clicks
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	2
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	2
8	2	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	2
11	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0
13	0	3	2	32	17	34	0	0
14	0	0	0	0	0	0	0	0

Table 28 – Time it took each user to complete each task in the first user test iteration; The last column shows the minimum time required to perform each task by an experienced user.

Task N	User 1 time	User 2 time	User 3 time	User 4 time	User 5time	User 6 time	User 7 time	User 8 time	Median	Minimum Time
1	00:45	00:04	00:28	00:32	00:23	01:00	00:18	00:06	00:34	00:09
2	00:07	00:05	00:05	00:03	00:17	00:08	00:23	00:05	00:08	00:02
3	00:09	00:05	00:06	00:16	00:23	00:03	00:24	00:08	00:11	00:03
4	00:06	00:01	00:04	00:03	00:03	00:10	00:23	00:21	00:07	00:02
5	00:15	00:05	00:21	00:05	00:15	00:25	00:03	00:39	00:23	00:08
6	00:11	00:03	00:05	00:03	00:04	00:05	00:09	00:14	00:08	00:03
7	00:57	00:08	00:37	00:33	01:27	00:38	00:23	00:25	00:47	00:12
8	00:06	00:02	00:03	00:02	00:04	00:11	00:02	00:02	00:08	00:05
9	00:15	00:00	00:02	00:01	00:06	00:08	00:24	00:02	00:07	00:03
10	00:34	00:00	00:20	00:35	00:14	00:20	00:00	00:28	00:31	00:11
11	00:03	00:00	00:02	00:01	00:02	00:02	00:01	00:01	00:06	00:04
12	00:06	00:10	00:23	00:03	00:09	00:10	00:19	00:05	00:11	00:02
13	00:15	00:03	00:23	01:51	02:28	02:03	00:00	00:19	00:26	00:05
14	00:18	00:00	00:04	00:05	00:01	00:08	00:01	00:03	00:05	00:02

9.4.6 USER TESTS – SECOND ITERATION RESULTS

Table 29– Number of wrong clicks on the second user test iteration for each task (it is considered a wrong click, a click that is not meaningfull to complete the task (user should not have pressed it))

Task N	User 1 N Clicks	User 2 N Clicks	User 3 N Clicks	User 4 N Clicks	User 5N Clicks	User 6 N Clicks	User 7 N Clicks	User 8N Clicks
1	0	0	0	0	0	0	0	0
2	0	3	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	0
8	0	0	2	12	0	0	3	0
9	0	0	0	2	0	0	0	0
10	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0
12	5	0	0	0	0	0	0	0
13	4	0	9	0	2	0	16	0
14	1	1	0	1	0	1	0	0

Table 30 - Time it took each user to complete each task in the second user test iteration

Task N	User 1 time	User 2 time	User 3 time	User 4 time	User 5time	User 6 time	User 7 time	User 8 time	Median	Minimum Time
1	00:31	00:33	00:13	00:20	00:13	00:18	00:10	00:26	00:28	00:09
2	00:33	00:14	00:03	00:06	00:04	00:10	00:23	00:00	00:10	00:02
3	00:34	00:17	00:03	00:06	00:04	00:10	00:01	00:04	00:08	00:03
4	00:23	00:06	00:03	00:09	00:04	00:03	00:05	00:03	00:06	00:02
5	01:26	01:02	00:11	00:49	00:05	00:23	00:11	00:17	00:28	00:08
6	00:02	00:06	00:02	00:04	00:04	00:12	00:00	00:07	00:07	00:03
7	01:20	01:01	00:40	01:04	00:29	00:29	00:42	01:06	01:04	00:12
8	00:08	00:11	00:18	00:53	00:03	00:02	00:11	00:02	00:14	00:05
9	00:02	00:03	00:03	00:07	00:05	00:04	00:19	00:05	00:07	00:03
10	00:36	00:20	00:25	00:28	00:15	00:12	00:20	00:15	00:31	00:11
11	00:05	00:01	00:01	00:02	00:01	00:04	00:00	00:02	00:05	00:04
12	00:17	00:09	00:01	00:00	00:02	00:01	00:04	00:09	00:05	00:02
13	00:26	00:35	01:57	00:23	00:19	00:16	01:38	00:22	00:28	00:05
14	00:05	00:09	00:23	00:00	00:07	00:00	00:22	00:16	00:10	00:02

9.5 3D

9.5.1 BAKE TEXTURES – RENDER SETTINGS

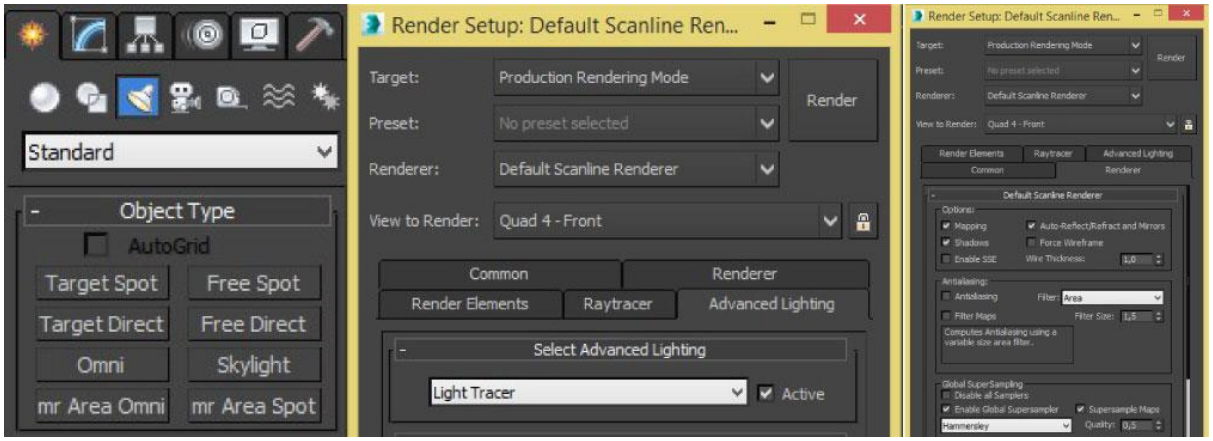


Figure 131 – Render settings to be able to bake the occlusion map.

9.5.2 CHARACTER CREATION EVOLUTION

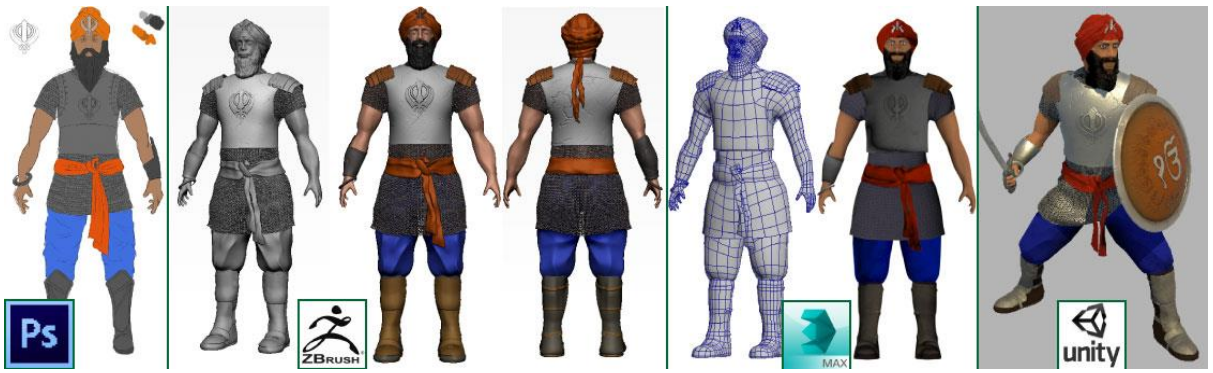


Figure 132 – Ghora model creation evolution.

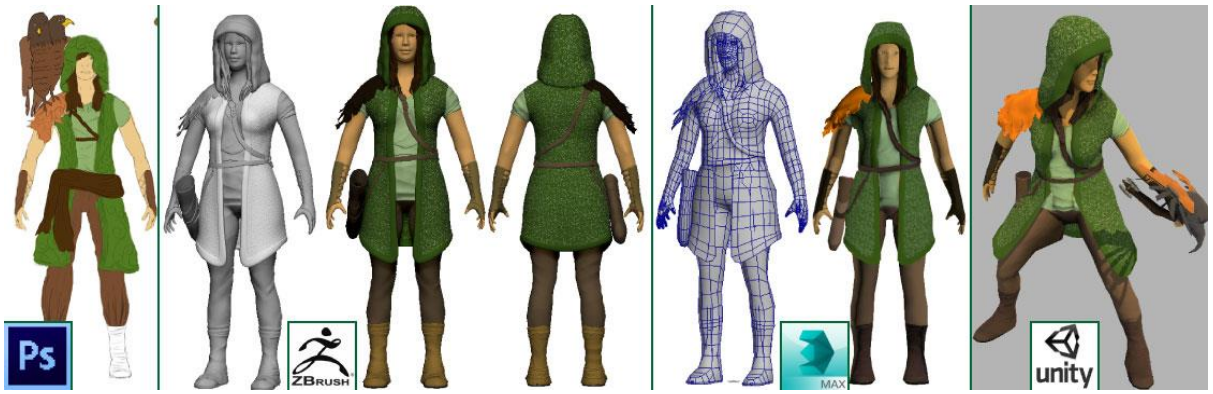


Figure 133 – Kulla model creation evolution.

Figure 134 – Lipp model creation evolution.

Figure 135 – Loper model creation evolution.

Figure 136 – Wang model creation evolution.

9.5.3 WEAPON CREATION EVOLUTION

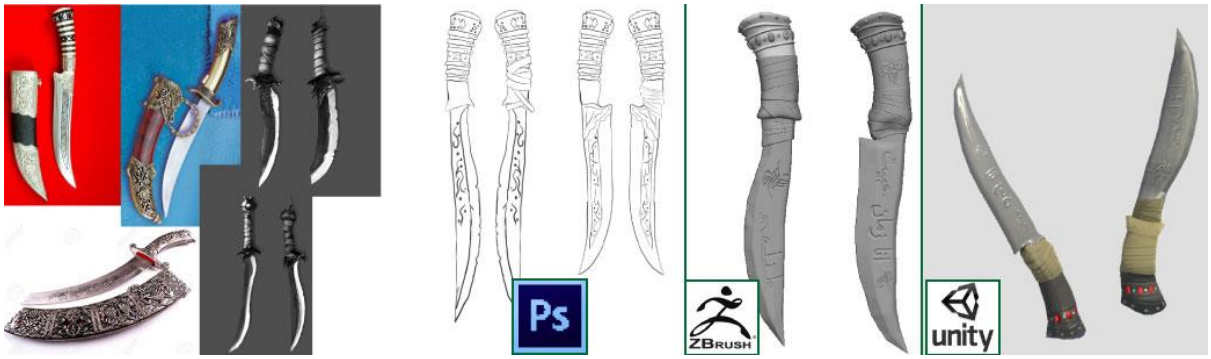


Figure 137 – Malik's daggers model creation evolution.

Figure 138 – Ghora's shield and sword model creation evolution.

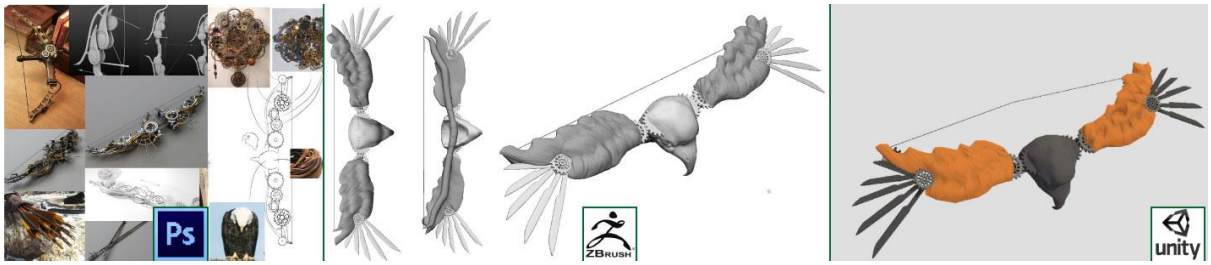


Figure 139 – Kulla's bow model creation evolution.

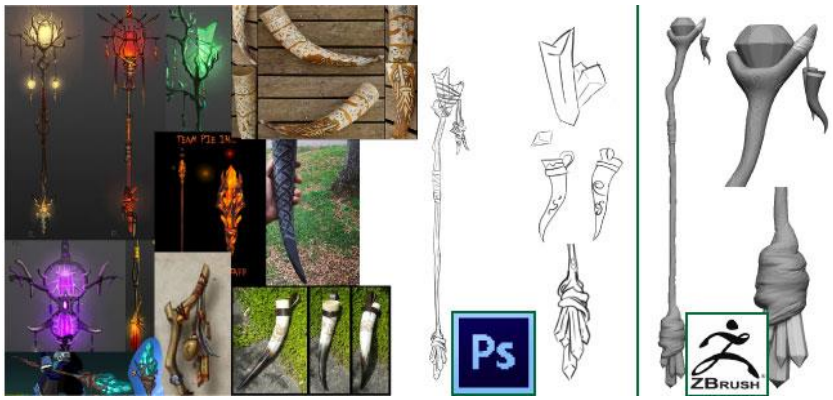


Figure 140 – Lipp's staff model creation evolution.

Figure 141 – Loper's flask model creation evolution.

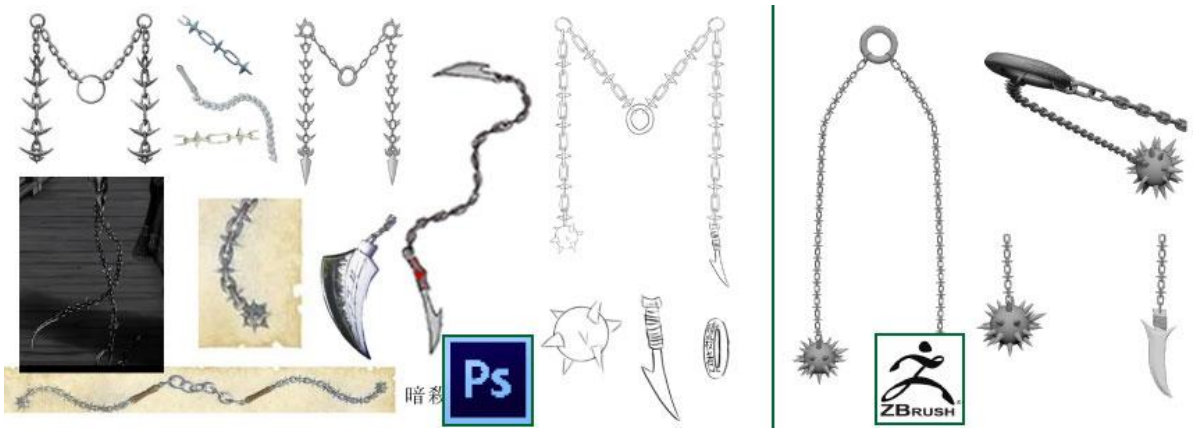


Figure 142 – Wang's chains model creation evolution.