Alex Manns

Video game companion mobile applications and their future

Bachelor’s Thesis
Bachelor of Engineering, Information Technology

2019
The objective of the thesis was to examine the companion application concept like where it came from and where it is now. Additionally, to explore this subject more, making a game and companion application prototypes was a part of the objective to study the concept’s practicality in the design and development of games.

This thesis details the general concept of companion applications and what it stemmed from to understand how it has evolved and what the current state of it is to be able to implement one in a prototype. To help this, eight companion applications from recent years were compared with each other based on groups of common features found in them. Additionally, this thesis describes the design and implementation processes of the game and companion application prototypes in a comprehensive manner, going over details of planning and issues in development.

Designing and development of a game and its companion mobile application was the predominant purpose of this thesis. The Unity game engine and Xamarin mobile development tools were the main tools of development, with other various tools used to improve the ability to test the prototypes and manage the project. The designing of the prototypes was done further than the prototypes would be developed, to explore ideas relevant to the companion application concept at least during the planning process.

As a result, more insight into the current state and the future of companion applications was gained. The prototypes showed some difficulties with the concept at a small scale, nevertheless the project was a success as the prototypes achieved some of the features present in companion applications. The companion application concept shows great future promise, as the market of people owning both a smartphone and other gaming hardware grows and that there are more of them being made although there is room for experimentation with features. Additionally, the designed game and companion application serve as a good example for the concept.

**Keywords**

companion applications, game design, game development
CONTENTS

ABBREVIATIONS ................................................................................................................. 5

1 INTRODUCTION .................................................................................................................. 6
   1.1 Objective of thesis ........................................................................................................... 7
   1.2 Commissioner ................................................................................................................ 8

2 VIDEO GAME COMPANION APPLICATIONS ..................................................................... 8
   2.1 History and current state ............................................................................................... 10
   2.2 Functionality .................................................................................................................. 13
   2.3 Analyzing examples ....................................................................................................... 15
      2.3.1 Features .................................................................................................................. 17
      2.3.2 Battlefield Companion ............................................................................................ 21
      2.3.3 Call of Duty Companion App ................................................................................ 23
      2.3.4 Final Fantasy XIV Companion ............................................................................... 24
      2.3.5 World of Warcraft Companion App ...................................................................... 26
      2.3.6 Fallout Pip Boy ....................................................................................................... 27
      2.3.7 Red Dead Redemption 2 Companion ..................................................................... 28
      2.3.8 Ghost Recon: Wildlands HQ ................................................................................. 29
      2.3.9 Destiny 2 Companion ............................................................................................. 30
      2.3.10 User experience .................................................................................................... 32
   2.4 Thought Experiment ...................................................................................................... 33

3 TOOLS ................................................................................................................................. 35
   3.1 Unity ............................................................................................................................... 36
   3.2 Visual Studio ................................................................................................................ 38
      3.2.1 Xamarin ................................................................................................................ 40
      3.2.2 Json.NET .............................................................................................................. 40
   3.3 SourceTree .................................................................................................................... 41
   3.4 Wireshark ...................................................................................................................... 42

4 PLANNING ............................................................................................................................ 43
# Table of Contents

4.1 Scope ............................................................................................................. 44
4.2 Game ............................................................................................................. 46
  4.2.1 Overall design ......................................................................................... 47
  4.2.2 Development plan .................................................................................... 49
4.3 Mobile app ...................................................................................................... 50
  4.3.1 Overall design ......................................................................................... 51
  4.3.2 Development plan .................................................................................... 53
4.4 Shared networking library ............................................................................. 54
5 DEVELOPMENT ................................................................................................. 56
  5.1 Unity ............................................................................................................. 57
    5.1.1 Data structure ......................................................................................... 59
    5.1.2 Character ................................................................................................. 60
    5.1.3 Game World ............................................................................................ 62
  5.2 Xamarin ......................................................................................................... 63
    5.2.1 Learning Xamarin .................................................................................... 64
    5.2.2 User Interface ........................................................................................... 65
    5.2.3 Game data view ....................................................................................... 66
  5.3 Data transfer connection .............................................................................. 69
    5.3.1 Game Server ............................................................................................. 70
    5.3.2 Mobile Client ............................................................................................ 71
6 CONCLUSIONS .................................................................................................... 72
  6.1 Prototypes ..................................................................................................... 74
  6.2 Future development ....................................................................................... 75
SOURCES .................................................................................................................. 76
FIGURES ................................................................................................................... 80
APPENDICES

Appendix 1. The Quest of Companionship Game Design Document
ABBREVIATIONS

AI  Artificial Intelligence, computer controlled element

GDD  Game Design Document, a document used to give a detailed description of each component in a game.

GUI  Graphical user interface, visual controls that the user of software uses to operate the software.

HUD  Heads up display, commonly used in video games where visual elements showing game statistics over game and that game elements can be hidden behind.

JSON  JavaScript Object Notation, a programming language-independent human readable file format meant for data object transmission.

NPC  Non-player character, a entity in the game that looks similar to the players character, but is AI controlled.

OS  Operating system, a software that manages a computers resources and programs running on it.

UI  User interface, like GUI but a more general term for controls the user can use to operate the software with.

IP  Internet Protocol, system or method for designating receiving and sending elements of a internet connection.

TCP  Transmission Control Protocol, system for sending bits of data packets over the internet, with TCP emphasising on data being sent and received properly.

UDP  User Datagram Protocol, system for sending bits of data packets over the internet, with UDP emphasising on speed, where 100% correct data is not required.
1 INTRODUCTION

Smartphones are an increasingly used for a multitude of purposes through their ever-increasing mobile application or app market. Many of these directly support a specific hobby or activity, such as consuming different types of media, board games, sports or even specific video games. They are meant to provide a new level of interactivity to these activities, enhance the experience, or otherwise support by providing information on the activity for the user. These kinds of applications are also sometimes called second screen applications. The second screen term is more often used as a word for the experience or trend itself, i.e. using a companion application is a second screen experience.

The increasing use of video game companion applications is a form of utilizing this growing use of smartphones for computer and console games is an indication of smartphones becoming more and more common as supportive devices. These companion applications are a rather recent phenomenon for video games, starting to truly become a fully realized concept during the late 2000’s and early 2010’s. One of the first considerably major uses of the concept was when some of the eighth generation of home video game consoles released with accompanying companion applications. This is what could be said to have caused interest towards the companion application concept in the gaming industry. The features incorporated in game companion applications have evolved over the years since their theoretical inception since the early 2000’s.

From then onwards some major video game publishers have started to release their own companion applications for some of the games that they have developed. Typically, these applications support the game experience by providing synchronous and asynchronous features. These synchronous features would be synchronized with the game itself, so they would update each other. Commonly this is seen as a map on the companion app, which would update based on where the player of the game is, or other trackers for progress in various aspects of the game. Asynchronous features would be elements, that are more detached of the game experience, such as lore content, or maybe guides for the game. Frequently, game companion
applications are game-specific, meaning that they do not work for other games, even in the same franchise. This lack of modularity or flexibility to support more than one game may be one of the reasons why indie or smaller developers have strayed from this concept of creating a smartphone application to support the experience of a game.

1.1 Objective of thesis

The interest of this thesis lies studying in the companion application concept itself and how it has been implemented in the games industry. How has it evolved and how have the players themselves perceived it in games as a supportive tool? The different ways this concept has been realized is of interest as well, because many developers have different objectives for their companion applications.

Of major interest is also the reason for why larger companies are willing to develop these applications instead of smaller ones. Is it because developing mobile applications in addition to a full computer or console game are not profitable enough, the costs or development time needed upfront are too high for a smaller developer? The primary research question is thus what will be the future of companion applications for games? Big part of that is how difficult it is to develop them. The secondary research question is how these companion applications are built into the game, such as how they share their resources and how they let the user interact with the game through the companion application?

This thesis will first cover the viability of this concept for a smaller game project by studying the concept and its examples of it from different developers with different objectives. To understand the companion application concept, a deep analysis to the available resources on this subject will be done such as articles, online blogs, documentation and other scientific documents. Ample emphasis would be put on how these examples have performed according to their users, like their opinion on different features. Following the exploration of the concept, the thesis will shift focus to using the knowledge gathered from studying the subject to design and develop a game and companion application prototypes. These prototypes were developed to
the point where they could connect to each other and transmit data between them and provided a minimal interface for a game. The resulting prototypes and their plans will be compared with the findings gained from examining the examples of real companion applications made for real games.

The planning and development chapters will provide an overall view of the process of making a game that can connect with a mobile companion application and transmit data. There, multiple topics related to planning and development will be explored in detail, like the requirements of the concept for the game and mobile application, networking, data structure and the problems during the whole process. As the author had no preceding knowledge of networking and mobile application development, those aspects of the prototypes required additional learning and experimentation on top of developing a game and mobile application based on the unfamiliar companion application concept.

1.2 Commissioner

The commissioner of this thesis is the South-Eastern Finland University of Applied Sciences, XAMK. The commission is for the wish to explore the video game companion application concept in a practical context, as it is a largely unexplored topic in XAMK. GameLab is the studying environment for game development students at XAMK. GameLab provides students with modern equipment and tools for game development for multiple platforms, like desktop platforms and mobile platforms. These tools are commonly utilized in the industry, to assist students in transitioning to the work life and to develop a higher likelihood of employment in the game industry.

2 VIDEO GAME COMPANION APPLICATIONS

Since the release of smartphones, using them as second screens or as companions to activities has become an increasingly sophisticated experience. Initially, it was focused on complimenting a television watching experience, like providing information for a television show or movie that the user was watching. Television networks and production houses supported this as it got more popular by funding, developing and releasing mobile
applications for the purpose of providing that enhanced experience and interacting with third party products. This second screen trend is what contributed greatly to creating the companion applications for games as we know them now. This growing trend may be why the gaming industry started seeing experiments with companion applications. (Pediredla, K. 2015.)

Currently, in more ways than one the gaming industry is connected to the mobile market, as many as 36 percent of United States or US households play on their mobile phones, which is contrasted by the game console and computer users holding 36 and 41 percent respectively. The other platforms with less of an userbase use other wireless devices, dedicated handheld systems and virtual reality devices to play games. These values are shown in a graph in Figure 1. This all totals to 60 percent of the US population playing games daily, which amounts to around 200 million people in 2018. These statistics do however not mean that computers, smartphones and game consoles compete for the same user bases. Most mobile games are much simpler than games from the two other main platforms. This means there is room for coexisting and as such companion applications stand to potentially benefit all these systems and platforms. While this is specifically for US households, it can be used as a rough estimate for other first world nations such as European Union nations. (ESA. 2018.)

Figure 1. Distribution of gaming platforms in USA, 2018 among households where games are played.
One of the main purposes of a video game companion application is to benefit the gameplay experience of its user for a game. Frequently, this is seen in companion applications as mirroring important or useful information from the game itself on to the smartphone device. This is done to reduce the need to shift between the user interface menus of the game and the game itself by having that same data available in a simpler to access format. This works especially well for games that are complex and where information hidden unintentionally behind menus because of how much of it is in the game. Companion applications also incorporate ways to access features from the game itself on the smartphone, like accessing one’s player character and customizing it or in general seeing one’s statistics or inventory in a game. This can be a very useful feature for people who ponder about a game occasionally during the day, and do not necessarily have access to the game at the time. The companion applications usually attempt to incorporate themselves into the theme of the game as well, trying to imitate it visually and implementing the same audio effects. This further builds on the foundation of the companion application to be more immersive to use with a game.

So, while companion applications have been created to improve the experience of a game, their fundamental purpose is to work as a marketing tool for the game they were made for. The gaming industry is still refining the companion application concept, still not having fully identified a model that works consistently. Desirable features have been conceived and suitable ways have been found to manipulate companion applications to fit the smartphone platform.

2.1 History and current state

While the second screen trend of applications made to support television watching may have been what encouraged the gaming industry to make companion applications for smartphones, the true inspirations for them could come from as early as 2003. At that time, a certain game console, Nintendo GameCube, could interface with a portable gaming system called the Nintendo Gameboy Advance to play minigames or otherwise interact with the main game. The gaming industry was smaller at that time and did not match
the scale and number of computers, consoles and smartphones that are active today. (O’Harrow, K. 2015.)

Official companion applications were at their most popular during the 2012-2016 years, started by two of the eighth generation of game consoles, Sony’s Playstation 4 and Microsoft’s Xbox One, when they received their own companion applications before the debuts of the consoles, PlayStation App (Sony) and Xbox SmartGlass, now known as Xbox App (Microsoft 2018) respectively. To connect to their respective consoles, both the console and smartphone had to be on the same local network. Both had good social features, where one could communicate with and manage one’s list of friends like on the console. Both had ways of directly interacting with the console akin to using a controller, the default way of interacting with the console. (Wehner, M. 2013.)

By 2015, the level of interaction possible in companion applications with the game was quite high. In applications such as Watch Dogs ctOS Mobile, one can connect to some other players live game and try to block their progression using a variety of abilities. Battlefield 4 Commander was an extension to one of the special player roles in the game, the commander. It allows someone to support other players in the game. What the application did was to allow a person to use and manage all of those additional responsibilities and abilities the role brought upon a player on a detached map on a mobile device. As for a slightly more common example of companion application features, Assassin’s Creed IV Black Flag companion application has a map that the users can interact with and keep track of their waypoints and mission objectives (Figure 2). It also allows one to play a minigame, where if there are ships available, they can be sent on missions and that way gain resources for the main game. These features found in the Assassin’s Creed IV Black Flag companion application are some of the more common found in companion applications in general and can be found in many different companion apps for games. (Pediredla, K. 2015.)
The popularity of official game companion applications has been fluctuating during the past few years. 2014 and 2015 saw many releases of companion applications for games, but many of these applications have not survived to 2018, like the game Watch Dogs’ ctOS (Ubisoft, 2018), Assassin’s Creed 4: Black Flag’s companion app (Ubisoft Forum, 2017), Metal Gear Solid 5: The Phantom Pain and Ground Zeroes companion applications (Konami, 2018), and Deus Ex Universe application (Eidos Montreal, 2018). This does not mean that there are no new larger profile companion applications being released. New, official companion applications for bigger games are released very sporadically, especially when being released by the same developer or publisher. Larger games take a long time to make, on average 3 years, likely longer (Quora, 2018), and these large games are the most common target for a companion application, whether it came from an official or unofficial source.

Some of them, like the Assassin’s Creed 4: Black Flag, do not receive a graceful ending to their support by the way of an announcement, instead they were left without support from the developers, and slowly became unusable. As applications receive no announcement for their discontinued support, it is hard to say why this occurs, but one such reason may be the cost of
maintenance of applications. Smartphone operating systems get updated often, and so maintenance and reworking the application for long-term support is required to keep the application compatible with the newest software requirements. (ThinkMobiles. 2018.)

### 2.2 Functionality

Companion apps for games differ from their original use of being a part of a television experience because games need more focus than consuming other media such as a movie or a television show. Playing a game, at least on the computer or the console, commonly also occupy both players’ hands. This means they must have a different purpose of their companion app and it varies from game to game. Some companion applications employ features to stay useful outside of playing the game. Examples of such features could be the ability to read the games lore or guides for it, social features, or minigames where rewards give the user some benefit in the game such as items, equipment, resources, experience for a level up or money. (Holly, R. 2015.)

This subject so far has focused on companion applications made or commissioned by the game developers or publishers themselves, who usually can modify the game itself to support a complimentary application. Therefore, a third party or unofficial companion applications, i.e. by an entity not associated with the original publisher or developer of a game, are usually a less interactive experience, as they have limited ways of interaction with games. Because of this limitation, some of them employ very creative ways to make themselves useful for a game. Third party companion applications are much more common, because their development does not usually require approval from the original developers, as their functions do not need to work with the game itself. An exception to this is when games have API’s or Application programming interfaces integrated into them, through which these unofficial developers and begin to interact with a game. (Holly, R. 2015.)

The features found in companion applications vary quite a large amount depending on the style, objective, and genre of the game. Examples of these include things like social features, where the smartphone is connected to a games social system like the friend list, the comprehensive databases of
information for the game including guides and news about the game, or a marketplace for browsing, buying and selling the items and customizations of games for the players character found in the game. Further examples of features in companion applications would contain maps for the game world, both static and real-time maps for the game world such as seen in Figure 2, where the player controls a boat, minigames or other ways to gain rewards for the player in the game such as seen in Figure 3, where the player can play a card game to get in-game currency, or having the smartphone itself act as a controller to the game by it having the ability to directly interact with the game in some way. However, companion applications for the games of a similar genre frequently incorporate features of the same kind. Official and unofficial companion applications are different in their scopes as well. Most unofficial companion applications are unable to directly interact with games, and instead provide other kinds of features for their user. (Tan, B. 2015.)

Figure 3. View of the tower defence minigame in Star Wars Battlefront (Electronic Arts, 2019a)

For example, there are many smartphone applications for simply just calculating times or containing maps, lists and guides for ways to accomplish something for roleplaying games. Games such as RuneScape, Diablo 3 and Guild Wars 2 are great examples of this because of their MMO-nature, meaning massively multiplayer online. They are very large online roleplaying games, and they contain countless activities the player can do and very many
aspects of their character that they may want to know about more. Having that on a smartphone application is useful for a player of these games, because that means one can at any time learn about specific mechanics or information about the game.

While one of the main focus of a companion application might be to be useful to their user and their experience of a game, their other purpose is to market that game itself. The biggest reasons for this are because they are released in advanced of the games they are meant to be companions to and because they are a supportive application for games. It is also very common for them to be free to use, without advertisements integrated into the application or other monetization methods. An exception to this is games which themselves have special monetization models such as microtransactions for in-game currency, as these are sometimes supported by the companion application as well. (Eccentric Engine, 2014.)

Companion applications usually connect with the game experience with a direct connection through the local network, through an account connected to the player for the game or even both at the same time. With an account system, a companion application can more easily access synced statistics that the player may want to see but this also usually means the game itself requires a multiplayer focus or flat out need to be online to function. Minigames that reward the player also usually work through an account system, possibly that it can be validated in some way, which means they need an internet connection to function. Therefore, in general, companion applications have been trying to become more useful outside of the game and been successful at it. They have however become more and more dependent on an internet connection to achieve higher usefulness, especially for using the application outside the game although there are exceptions to this, when they purely use a direct connection to the game to achieve the interactivity.

2.3 Analyzing examples

To more accurately analyze the examples of companion applications, this subchapter focuses on some examples of such for different games. These examples will consist of games with different genres, but they will be of more
recently released companion applications. Examples chosen also must function correctly and that they have not been taken off their respective mobile application marketplaces. As most companion applications require their respective game or an account related to do it, this analysis will rely on information presented about the application, such as on these mobile application marketplaces or user reviews and showcases of said application.

The criteria for grouping functionality used here took inspiration from Tan B. 2015 article about the second screen experience in gaming (Tan, B. 2015). In it, he talks about five categories of functionalities for these applications; Social, information, marketplace, minigame and remote device or control more specifically. These were expanded upon as many overlapping functionalities is found in companion applications, and some are developed only partly. Commonly seen features that were separated were things such as viewing character statistics and modifying them, maps appearing in companion applications in general and otherwise data being synchronized. As this thesis focuses on the official companion applications, unofficial companion applications were not taken into consideration during comparison. The general overview and distribution of these categories for each game chosen as an example can be seen in Table 1. Games which fulfill the requirements of a category get a check, depicted as an X-symbol. There were some important details for certain features for some games which are explained on the table itself.

The following 8 examples of games and their companion applications were chosen for this analysis; Battlefield Companion, formerly Battlefield Battlelog (2012), first made for Battlefield 3 (2011), and now supports more recent Battlefield first person shooter or FPS games, Battlefield 4 (2013) and Battlefield 1 (2016); Call of Duty Companion (2018), made for the most recent Call Of Duty FPS game Black Ops 4 (2018); Final Fantasy XIV Companion (2018), made for a massively multiplayer online roleplaying game or MMORPG of the same name, Final Fantasy XIV (2013); World of Warcraft Companion (2016), made for a MMORPG of the same name, World of Warcraft or WoW (2004); Fallout 4 Pip-Boy, made for a roleplaying game or RPG, Fallout 4 (2015); Red Dead Redemption 2 Companion (2018), made for an action-adventure game of the same name, Red Dead Redemption 2 or

2.3.1 Features

Companion applications often have similar social features that are found in their respective games, but such features are absent in companion applications for games that do not contain these features. Social features such as friend lists, chatting and forming, managing and exploring groups of players are very rarely seen outside of online multiplayer focused games like MMORPG’s or competitive games like World of Warcraft, Final Fantasy XIV, Call of Duty-series or Ghost Recon Wildlands. In those games communication with other is usually encouraged with in-game mechanics, or teamwork is a part of the game. In singleplayer games, the player will always be alone in the game world himself, so these features are not needed and there would not be any useful reason for a player to use them.
The most common feature in companion applications is that they mirror data from the game, such as progress in missions, character information like the status of that character, the statistics of a character or its actions, and items found in inventory. In more account-based games such as MMORPG’s or competitive games, statistics and information for a player’s profile are also commonly found in companion applications for those games. This information varies between games, for example in FPS or other shooter games the companion application could contain how high his rank is in the games ranking or leveling system that is commonly found in place to differentiate between different players progress in the game or amounts for all the player's actions on the game such as defeated enemies, the player's own defeats and accuracy of their shooting. While it is important to some players to monitor their progress in a game, seeing the status and details of missions or quests for games is more useful for the average player, as especially RPG’s contain many missions or quests the player can do at the same time. Keeping a track

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial Release</th>
<th>Genre</th>
<th>Connection</th>
<th>Social</th>
<th>Market</th>
<th>Player</th>
<th>Info</th>
<th>Map</th>
<th>Sync</th>
<th>Mini-games</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battlefield Companion</td>
<td>2013</td>
<td>FPS</td>
<td>Account system</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Call of Duty Companion</td>
<td>2018</td>
<td>FPS</td>
<td>Account system</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Final Fantasy XIV Companion</td>
<td>2018</td>
<td>MMORPG</td>
<td>Account system</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X^1</td>
<td>-</td>
</tr>
<tr>
<td>World of Warcraft Companion</td>
<td>2016</td>
<td>MMORPG</td>
<td>Account system</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fallout 4 Pip-Boy</td>
<td>2015</td>
<td>RPG</td>
<td>Direct connection</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Red Dead Redemption 2 Companion</td>
<td>2018</td>
<td>Action-adventure</td>
<td>Direct connection</td>
<td>*2</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ghost Recon Wildlands HQ</td>
<td>2017</td>
<td>Tactical Shooter</td>
<td>Direct connection</td>
<td>*3</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Destiny Companion</td>
<td>2014</td>
<td>FPS RPG</td>
<td>Account system</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

^1 Only synchronises visual appearance of character, item marketplace and character inventory

^2 Additional features when using account

^3 Mobile game does not require any connection, but connecting that to the desktop game requires account
of all of them at the same time is no small task, so it is very handy if the same information that is usually hidden behind menus can be accessed quickly by using a smartphone.

While many examples of marketplace features in companion applications were not observed in the aforementioned list of games for this chapter, the games themselves did not contain in-game ways of buying customizations or other items for the game using a currency indirectly related to the core mechanics of the game. Exceptions to this are the MMORPG’s, especially ones that are free for a person to acquire and play, which commonly incorporate such a marketplace with an in-game currency that can be bought with real money or achieved through other means as a method of monetization of the game. The use of this marketplace or the player’s use of microtransactions is a growing trend in games, which can be further promoted as an option by enabling the companion applications of games to contain ways of buying from this marketplace.

Games that focus on a rich story and a large world for the player to play in commonly feature a map for that player to navigate the game world. Quests and other objects and structures may be found on the map. Because of this, companion applications for such games, like RPG’s or open world games usually have the maps of the world for the user to view and explore. This can be seen in Figures 4 and 5, where the player discovers locations in a large game world that then are recorded on a map the player can view by going into a menu. Examples of such games are Fallout 4, Red Dead Redemption 2, World of Warcraft and Ghost Recon Wildlands. As navigation in these games is sometimes complicated because of the scale of the game world, having the map available on the smartphone can greatly simplify this. This is especially true if the game contains navigation aids or hints like showing the path to a selected location and the maps are connected such that these aids can be seen on the companion application.
Companion applications have a hard time being useful outside of using it with the game it accompanies. Companion applications usually do not focus on that aspect of the application itself, and instead focus on the features directly related to their respective games. This usually means the benefit someone might gain from using the application outside of playing the game with the application is purely related to the game and the players progress in it.

Companion applications sometimes contain a lot of information about the game, like controls, descriptions of game mechanics, locations and characters in the world and information on the enemies found in the game.

So especially RPG’s benefit from this, as those games are usually complex. With that information being available on a smartphone, something that these days people carry frequently, you can always explore ideas you get for a game. Some companion applications also incorporate news into the
application, related to the game or game series. News could be things like updates or patches, the new releases of new content for the game that can be bought or other events. Another way of being useful outside of directly playing the game is minigames that some companion applications implement. A good example of this is the Ghost Recon: Wildlands HQ companion application, which itself is really smartphone game, but the world and story is connected to the game it is connected to, Ghost Recon: Wildlands, as can be seen from Figure 6. This smartphone game is not the only feature in the application, as it also can connect to that main game and work as a map, or you can access some social features. Such minigames usually bring some benefit to the main game the user is playing, meaning that using the companion applications provides tangible benefits.

Figure 6. Management game in Ghost Recon Wildlands HQ (Ubisoft Entertainment, 2019b)

2.3.2 Battlefield Companion

Battlefield games are first person shooter games with varying themes such as modern, historical or science fiction. They have an emphasis on the use of
vehicles, unlockable equipment, the large player versus player or PvP battles and teamwork through various means like grouping players into squads, having a commander for a team who issues orders for squads and objectives requiring team effort. Through the companion application, you can access the same social system present in the game and its launcher, where you can see friends and their statistics, and what they are doing. The Battlefield statistics for one’s profile and equipment loadouts for the player can be accessed and modified through the application (Figure 7). In the game, there is a feature to make emblems. These emblems can be seen while playing the game. You can browse, make and edit emblems through the application. (Electronic Arts, 2019c.)

The companion application receives up-to-date news and other content on Battlefield games. The companion application requires a Battlefield profile to function, which means the ownership of a Battlefield game and an Electronic Arts account. The theme of the application is based on a game, Battlefield 1. The companion application supports only a few of the recent Battlefield games and not all features are supported for all of them. For example, the loadout functionality is only supported on Battlefield 1 while the other game supported by the app, Battlefield 4, does not get the same support.
2.3.3 Call of Duty Companion App

Call of Duty games are first person shooter games similar to Battlefield games such as modern, historical or science fiction. They have an emphasis on more competitive PvP combat, where players often have objectives completable on their own. The companion application has access to various social features, like the friend list and the ability to compare their game statistics and achievements with yours and create or join groups of players to have special objectives appointed to said. When completed, all members of the group gain special rewards based on the objective. The companion application receives up-to-date news on Call of Duty games, events and game updates. (Activision, 2019b.)

The user of the application also receives personalized tips and recommendations on loadouts and game modes based on this players playstyle, game statistics and equipment use in matches (Figure 8). Based on these statistics are also generated for different maps in the game for better
visualization. These statistics can also be viewed on their own for the player to analyze them. The companion application only supports the newest Call of Duty game, Black Ops 4, but support is planned for the next games in the series. The theme of the application is like the game. A Call of Duty account or other related account like a console or computer platform account is required to use the companion application.

![Figure 8. View of a player's statistics in Call of Duty Companion App (Activision, 2019a)](image)

### 2.3.4 Final Fantasy XIV Companion

Final Fantasy XIV or Final Fantasy 14 is a massively multiplayer online roleplaying game, set in a fantasy world. It follows the MMORPG-formula pioneered by World of Warcraft quite closely, meaning its focus is the online only massive open world, that is filled with various quests, events, activities and non-player characters. You can access the friend list of the account and chat with friends although only people with the companion application can chat with each other. The same restriction applies to the ability to plan or view events made for you, your clan or otherwise are a planned participant of the event. While you cannot view the statistics of characters, you can see what they look like and their items in the inventory and storage locations. These
items can be sorted for better viewing, moved between the inventory and storages, and completely discarded. The companion application also has access to the item marketplace of the game, where you can buy, sell and view all items being sold. (Square Enix, 2019a.)

The companion application for Final Fantasy XIV is unique in it that it uses a system more common in mobile games, where actions cost in-app currency, which can be bought with real money or gained daily by using the companion application. Like the game requires an account with an active subscription to play, the companion application requires a Square Enix account with an active subscription to be used. The application is loosely visually based on the game, except for items and characters, which directly come from the game. Figure 9 shows the home view of the application.

Figure 9. Home page in Final Fantasy XIV Companion (Square Enix, 2019b)
2.3.5 World of Warcraft Companion App

World of Warcraft is a massively multiplayer online roleplaying game, set in a fantasy world. It is one of the games that pioneered the MMORPG-genre like we see it today, which is an online only massive open world, with various quests, events, activities and non-player characters populating it. As it has evolved, players have become a part of the game experience for other players, and as such, the communities of players in the game can affect the game world itself. Most players also join one of the two factions who then in the game get missions and other events to fight the other faction. The focus is on the expansive world the player has the freedom to explore, and the roleplaying elements of the game. (Blizzard Entertainment, 2019a.)

Currently, there are no social features in the companion application, however such features are planned. Using the application, you can view your player character, its statistics, equipment, skills and progress in the game world in quests and other activities. You can also view active quests in detail, like their rewards and objectives. The companion application also contains the maps of the game world with markers that update in real time for things like quests. (Figure 10.)

One can complete or monitor the progress of faction missions through the companion application. To complete these missions the player needs champions, which are characters that can be recruited and upgraded using in-game resources. You can also track and research using the warfront technology tree, which is related to the faction system. These faction missions or the warfront research features are not exclusive to the application. As MMORPG games are almost exclusively online only, as it is a core part of their gameplay, the companion application requires a World of Warcraft account with an active subscription to be used, like the game needs one to played. The theme for application borrows many elements from the game itself, and otherwise visually is very similar.
Fallout games are role playing games, set in an apocalyptic world. The focus of the games of the series has changed over the releases, but the one of the recent examples, Fallout 4, focuses on a single player experience with the shooting aspect of the game being a big part of the game after that. There is also emphasis on exploration, and roleplaying features such as the player having the customization and freedom to play the character however desired. The companion application for Fallout 4 is unique, in it that it completely mirrors the in-game interface for the player's inventory, character statistics, mission progress and real time map in the game (Figure 11). (Bethesda Softworks, 2019a.)

The companion application can do this, because the interface for viewing and affecting this data is one system, so converting it to a mobile interface requires very little modification. The companion application contains no features to connect to the internet, for things like news about Fallout games. All this data
is synchronized with the game when the application has a direct connection to the game through being in the same network. This data is also saved to the mobile device, so that they can be viewed without a direct connection to the game. In Fallout 4 the player can find and play minigames inside the game itself. When they are found, and the companion application is used and synchronized with the game, you can play these minigames on the companion application. Apart from the direct connection to the game requirement, there is no other requirement to use the application in conjunction with the game. Because the companion application completely mirrors an interface from the game, it is themed after Fallout 4 and it only supports that game.

![Figure 11. Fallout Pip-Boy mimicking Fallout 4's menu interface (Bethesda Softworks, 2019b)](image)

### 2.3.7 Red Dead Redemption 2 Companion

Red Dead Redemption games are action-adventure games, set in in a western theme. The focus of the game is on the single player linear story, with rich characters and large world for this story to work in. There is a large emphasis on exploration, and some freedom for the player to do with the player character what they want. The player can unlock new equipment by spending currency or by performing tasks in the game. (Rockstar Games, 2019a.)

The companion application for Red Dead Redemption 2 is focused on accompanying the single player experience of the game, and so contains no social features. The companion application has a real-time map of the game world, which can be manipulated through the mobile interface by panning and zooming. The mobile companion application can also be configured to display
many of the user interface elements of the game, which then are disabled on the main screen of the game. One could access the game characters statistics and the player's progression in the game through the application.

As the game progresses, the player can unlock specific journal entries and illustrations written in the style of the protagonist of the game, that then can be viewed on the companion application. Equipment can be bought and received by the game character in the game. The companion application contains a manual for the game, where there are details of controls for the game, player statistics and activities. Through the companion application you can buy the official guide for the game that you can then view through the application. The application requires a connection to the game for most features, but some require the use of a Rockstar Social Club account. The companion application themes itself heavily after Red Dead Redemption 2, and the application only works with that game. Figure 12 shows the home view of the application.

![Figure 12. Main navigation view in RDR2 Companion (Rockstar Games, 2019b)](image)

### 2.3.8 Ghost Recon: Wildlands HQ

Ghost Recon: Wildlands is a tactical shooter game, set in the modern era. The focus of the game is in its open world setting, where missions and action occur in varied locations (Ubisoft Entertainment, 2019a). The companion applications focus is to function as a game itself where you get resources you can then send onto the main game to benefit the player's character. This
companion game’s scale is extensive and works as a story prequel to the game. In the companion game your complete side and story missions indirectly with recruited and upgraded rebels (Figure 6). The companion application does also contain features to connect to the game itself, like using a real time map and being able to access the user's friend list and Task Force, which is a group the player is a part of (Figure 13).

One can also chat with these friends and other Task Force members through the companion application. The application also receives news about the game, events and you can view information on the main game’s story elements like factions. The companion application can be used independently without the need for a Ubisoft or console account or a direct connection to the game, but to use the interactive features or to gain the rewards for the game you need both. The companion application is themed after the game.

![Map view in Ghost Recon Wildlands HQ](Ubisoft Entertainment, 2019b)

2.3.9 Destiny 2 Companion

Destiny 2 is a first-person shooter game, set in a mythical science fiction world. It is an online only game, with some roleplaying elements additional to the focus on the exploration of the game world. The gameplay consists of the
player versus environment and the player versus player elements (Bungie, 2019a). As social features, you can access your friend list and chat with them through the application. You can also create and manage a clan which is a group of players or join and view one’s statistics using the application. This clan interface can also be used to chat to the other members of the clan. There is a feature in the companion application to form fireteams, that are small groups of players. These fireteams are used to quickly get into a game match as a group.

The application receives up-to-date news, updates and updates on active activities and events for Destiny 2. Data, statistics and progress to collections and triumphs can be seen using the companion application. The mobile application also provides an interface to view all the user’s weapons, armor, and other items and to see all their statistics and perks. The view of this can be seen in Figure 14. These items can be moved between characters and the item storage vault using the companion application. While all features of the companion application are not restricted behind a Destiny or the console account requirement, most of them are. You can see news, updates, and community content showcases for the game, and access to a forum for the game without using an account.
2.3.10 User experience

As companion applications do not have a specific model that they can fully follow for designing their functionality, so do the user of these companion applications have very little demands from them. The demands from user of these generally align with normal smartphone application demands; functionality fitting a mobile interface meaning smoothness and simplicity of use. A generally unfavorable aspect of some companion application has been the exclusivity of some features or benefits from using the companion application. For example, Assassins Creed Unity launched with a companion application in 2014, which contained many of the popular features such as a dynamic map, statistics for the player's character, information about the game world and mission monitoring like the objectives and status of them. It also contained some minigames that rewarded the player with resources and sometimes. Some of these items that could be gained were exclusively gained from this companion application, and due to feedback from the players, less than a year later the game itself gained a way to get these previously exclusive items (Phillips, T, 2015). Players think that they do not want for a part of the games experience be locked behind something like a companion application. People do also hold the same opinion although less so for downloadable content or expansion packs for games that need to be bought with real money to be unlocked.

One concern of companion applications more related to their users is their longevity. Games do not need constant support, but if support for a game is discontinued, especially account-based online focused games can quickly become unusable. In these cases, companion applications likely follow the same fate, if they utilize this same account system or rely on data from an account from the game. Sometimes games that do not entirely focus on online gameplay, but the companion application relies on that connection and support for the game is discontinued, playing the game can still be possible while using the companion application is not. This is a common problem with older singleplayer games that had companion applications connected through an account system, because companies that publish or develop games release more games and move their focus of supporting those newer games.
This does not mean that companion applications that connect directly to a game completely avoid this, because the game itself is likely to work longer without support, as consoles and computers have a larger emphasis on backward compatibility and longer support programs made for them than smartphone platforms which are innovating and being developed constantly. This is also why the smartphone companion applications require more maintenance for them to function adequately and are more likely to be taken away from smartphone application marketplaces if this support becomes too time consuming or expensive for its developer.

2.4 Thought Experiment

To make it clearer how this concept could be utilized, an example companion application was thought of based on a recent game. The game chosen for this experiment is PA, or Planetary Annihilation. Planetary Annihilation is science fiction themed real time strategy game where the objective is to destroy your enemy commander, his constructed structures and units using your own constructed structures and units. These units and some structures function on different levels, and that limits or affects their ability to fight enemies on different levels. These levels are land, sea, air and space. The main focus of the game is on the large scale of warfare, so players get a great number of units to use and structures to build, and have to manage their movement, placement and planning. One of the more unique features of the game is that game matches can happen on multiple planets. These planets are multiple stages between which you can move units and on them you can build more structures and units.

A good companion application is meant to consist of functionality that aids the user playing the game. What those features need to contain to be useful depends on the genre if the game. In this case, the example is Planetary Annihilation, a large-scale real-time strategy game. There is a limited amount of companion applications for strategy games, especially official ones, and the unofficial ones that are there usually only contain information for all the units and structures in the game. Companion applications that interact with the game could be very beneficial for strategy games, because of their complexity and especially in larger scale strategy games they could provide information
on events that the player cannot see. In Figure 15, we can see a solar system consisting of a small red planet, a small earth like planet with water, a gas giant and a sun star. The small planets allow units to move on them, thus the planets are available for the player to migrate some units to. However, at the beginning, the player must pick one of them as their starting location.

Figure 15. View of a generated solar system game match in PA.

In the game, combat can happen in so many places and at different levels such as land, sea, air and space. With the addition of these separate levels and different stages or planets in a game match itself, the companion application could work as a sort of control and notification hub for the game to simplify responding to these randomly located events. When enemy players attack your units or structures, it creates a notification on the screen that the player can click on to bring the view to where that is happening. You could move it to the companion application so that the user could press it to bring the view there in the game. The companion application could also work as an enlarged minimap for the game, where you could see what the situation in places is. The game also incorporates the grouping of units, so through the mobile application you could have an interface where you can see the status of all these groups and be able to select them for the game. The game focuses on producing a massive number of units with factories, so to ease the monitoring of the status of those, you could have an interface for checking up on their status as a list of them and what unit they are producing currently. (Figure 16.)
Figure 16. View of the starting planet in PA, where the purple is the player and red elements are an ally of the player.

3 TOOLS

From this chapter onwards begins the process of planning and developing two prototypes, specifically the game and the mobile companion application itself to further explore the practicality of the companion application concept. This chapter describes the different tools used for the development phase. Before any considerations or planning was made towards the game and its companion application, the tools used when developing them had to be chosen. This is to avoid any confusion during the planning or development phases that might result from not knowing for example the limitations or requirements of the tools in the projects for the game and mobile application.

The tools chosen for these two projects were Unity, Visual Studio 2019 and Xamarin. Xamarin is an additional mobile development toolset available for Visual Studio. Additionally, other smaller tools were used, some of which came in use during development. These are Newtonsoft Json.NET, which is a JSON framework for the .NET framework that is used by Visual Studio, SourceTree, which is a desktop client for Git, a popular version control system
and Wireshark, which is a network packet analyzer. More detailed explanations for these tools and their use in the project are given in the following subchapters.

These tools used for the implementation process of these two prototypes have some requirements that were considered as objectives for this thesis, namely the ability to run it on other platforms. This means the cross-platform support of these tools was considered when choosing the tools. Them being free was also a requirement for the ease of adoption of the tool into one’s own project. Consideration was also given to the author's previous experience when choosing the development tools. The number of the used tools was also kept to a minimum, to test the practicality of developing companion application features at a low-level code. The computer system operating system used to develop the prototypes was chosen to be Windows 10, and the mobile operating system to be Android, but the tools are capable of being used to develop for multiple platforms although this was not to be tested as a part of this thesis.

3.1 Unity

Unity is a cross-platform game engine. For developing games with it, Unity comes with a customizable editor such as seen in 3D-modelling or image editing software, as seen depicted Figure 17. This editor is very flexible, allowing the switch locations and sizes of the elements of it, allowing a developer to easily create, assemble and build the code of the game and assets into a complete game. This code is written in the C#-language by the developer. The Unity game engine itself has been in development since 2005 and has millions of users around the world. It has added support for a multitude of platforms aside from normal desktop and mobile platforms. It is also free to use for non-commercial purposes, with there being licensing options to use it commercially depending on revenue. Unity enables this by using a subscription service and an account system, but these different versions differ very slightly in terms of available functionality to the developer using the engine. For this thesis, the free version was used, with the specific version chosen for the thesis being 2018.3.5. During the development process updates to this unity version were made up to 2018.3.14. Unity was chosen
because of the author’s previous experience, its good cross-platform support and that Unity scales well for the varying sizes of game projects. (Unity, 2019d.)

Figure 17. View of a project in Unity Editor 2018.3

When Unity was used to develop the prototypes, it was set up in the default way of development for windows as seen in Figure 18. Because mobile development was being handled a different tool, the Android build support for Unity was not needed. The reason why Unity was not chosen for mobile development is that it is more suitable used to develop fully fledged games, and a simple mobile application that just needs a data connection would contain many features from the game engine that would remain unused in the application. This would make the Unity version of the companion application take excess space on a mobile device and likely run slower as it processes the redundant Unity features in the application.
3.2 Visual Studio

Visual Studio is an integrated development environment (IDE) made by Microsoft for Windows and macOS. Specific version used in this thesis is the 2019 version. Its use in this thesis is to enable the developer to write code for the prototypes. It also allows the developer to deeply debug and test written code. Visual Studio offers multiple versions called editions, community, professional and enterprise. Of these, the community is the free version, so it was used for this thesis. The other, paid versions have some features over the free versions, like more in-depth debugging tools and collaboration features. Like Unity, Visual Studio provides a flexible editor for developers to write code with as seen in Figure 19. It implements features like code completion, which attempts to predict what the developer is going to write to speed up the process. (Microsoft, 2019g.)
To fully support Unity development, Visual Studio offers an addon, or toolset in Visual Studio as displayed in Figure 20 when installing or modifying a Visual Studio installation. This enables features like support for Unity specific elements for the code completion feature, and debugging the game through the Unity editor, as the editor can launch the game for debugging and code execution while the Unity game project code is viewed in Visual Studio.
3.2.1 Xamarin

The Xamarin name encompasses many different tools that a developer can use to create applications for different platforms such as mobile device OS' like Android and iOS, and desktop OS' like Windows and macOS by writing code in the C#-language. Developers have access to many different aspects of the device on the respective platform, for example, an Android device would have different a different way of accessing the camera than an iOS device. These applications can share their code between these different platforms, making cross-platform development simpler as unique code for each platform is reduced. Applications developed using these tools allow them to run natively on their respective platforms, meaning that when they are deployed on that platform, they do not do any emulation of conversion of commands between the application and the system, enabling them to be executed in a performance efficient manner and the interface for the application using OS specific controls. (Microsoft, 2019h.)

As Xamarin tools are used globally, and especially after recently the company responsible for them, Xamarin, was acquired by Microsoft in 2016, those tools offered have themselves been developed to be very robust and contain particularly modern and useful features for developers. Since Xamarin was acquired by Microsoft, Xamarin tools are available by adding an addon to Visual Studio using its installer, as seen selected in Figure 20. This comes with many different items that the developer can use. This includes such things as software for emulating Android devices and connecting a physical device to the computer for testing code specific for the OS of the device although there are some specific device requirements to develop for iOS.

3.2.2 Json.NET

Json.NET is a software framework for .NET. It is taken in as a dependency into a C# application project to give that project the functions provided by Json.NET. Its main feature is a flexible JSON serializer, that is used to convert .NET objects or other data containing items into JSON text to be used in the application or files for external use and then back to the .NET object. It also provides other features related to converting the JSON to other formats like
XML or customizing the serializer to application project specific needs. (Newtonsoft, 2019.)

The use of this framework came in during development of the data structure in the game. The game engine Unity comes with an integrated JSON serializer for the same function as Json.NET, but it is much less customizable, and in practice, lacked support for deep class structures that needed to be serialized. This created a necessity for a better JSON serializer library. Json.NET was chosen for this thesis because it is one of the most used frameworks for JSON related tasks and because it is endorsed by Microsoft for use with the C# language, which the prototypes were going to be programmed in. Specifically, a slightly modified version of this framework was used, namely the Json.NET for Unity, which includes support for that game engine and some features often used for game development like turning a JSON file into a binary file, meaning it cannot be read by a human directly.

3.3 SourceTree

To manage the files in this development project, SourceTree was chosen. SourceTree functions as a GUI for the Git version control system that tracks changes in files during software development. The provided interface significantly simplifies the use of Git which originally is a command line program with commands inputted as text. This can be seen in Figure 21, where the history view is seen. Git works in terms of commits to a repository, which is the main folder of a project. Commits are compilations of changes to the files and file structure of a repository with an included description of the changes. SourceTree allows one to select which changes to are set to be included in a commit, and then add a message to accompany that commit to describe the changes. There are a multitude of options for Git GUI applications, however SourceTree was chosen for the author's good experience with its usage. (Atlassian, 2019.)
Wireshark is a network packet analyzer. It is used to monitor the activity going through a selected network interface on a device, traffic going into and from the device where the software is used. It can be used to individually check every single activity on that interface, or one can enable the software to filter all the activity using custom rules, like what IP address is in the packet, or what protocol was used. This was used in the thesis to debug the networking components of the prototypes developed. Its use was limited to determining whether all devices in the companion application loop could detect the packets sent by the other device and seeing if the packets were valid in terms of their content (Figure 22). (Wireshark, 2019.)
Figure 22. View of local networks and their activities as seen by WireShark

4 PLANNING

This chapter goes over the designing and planning aspect for the game and the mobile companion application prototypes with their purpose was to help experiment with applying the companion application concept. Planning and designing of these prototypes went much further than they were developed. Details of what aspects were considered during the design of the game and the companion application were discussed. A GDD was made to assist the development of the game, which contains more detailed descriptions of many of the features a sufficiently complicated game may have if it has an accompanying companion application. This chapter focuses more on the companion application concept’s limitations, requirements, and how those affect a games design and how it determines the functionality of the mobile application. An overview of the design of the game and mobile applications is shown using diagrams and concept imagery, with the external GDD being referenced for further details on the game as Appendix 1. To make the diagrams and the concept imagery, a free online diagram and flowchart maker called draw.io was used, and paint.net, a free image and photo editing program.
During the planning process a need for a shared code library was desired that would share the declarations of datatypes as C# interfaces or implemented datatypes, so that both the game and companion application were required to have knowledge of and be able to use, as managing a connection between the two without being able to interpret the response from the other device was going to be problematic. This library also contained code for the networking component of both the game and the mobile application, which would then use that functionality through it. The game received the majority of the work during the planning phase. During the planning phase, these projects proceeded in a linear and sequential manner, meaning that after the planning for the game project was finished, then came the mobile project to be worked and after that the shared networking library. Some adjustments occurred to all of the projects throughout the whole process, but largely close to the end of the planning phase, as new findings created a need for revision of the other designs.

4.1 Scope

The first step in the planning process was determining the scope and general plan of the game and mobile companion application. Most games with assistant companion applications are complex in terms of the game world scale, multiple statistics related to the player character and gameplay elements utilizing these. In many cases, this is a result of the particular genres of games, RPG’s. As a base in an RPG, commonly the player character various statistics like the characters different attributes, health, magic and stamina points which determine the characters alive state and what spells or skills it can use, or how much damage the character receives or deals to other characters. Most of these are also connected to each other or determined by some of them, like a specific attribute can increase the player’s maximum health points when increased by the player. This is one particular area which creates a lot of complexity to the gameplay, as the player is recommended to follow all of this data closely in order to succeed in the game. This means having a game be a traditional RPG results in it by default containing a lot of complexity. So, for the purposes of the thesis’ prototypes, this was chosen as the genre for the video game portion of it, as it would be reasonably realistic to simulate a real game that would benefit from a companion application.
As a result, examples for the companion applications also came from RPG’s that have had companion applications made for them. Specific examples used during the planning process were games like Fallout 4 and its companion application, The Elder Scrolls V: Skyrim and Metal Gear Solid 5’s companion application. Fallout 4 is a very traditional RPG, where many of the player’s characters attributes and skills determine what and how well you can do things in the game world. Its companion application is also very simple in its concept, as it replicates the menu interface from the game onto the mobile application, allowing the player to use it to see everything they could see inside the game (Figure 23). Skyrim served as another reference point for a traditional RPG in a fantasy theme instead of Fallout 4’s sci-fi theme. Metal Gear Solid 5 is a stealth shooter game, which contained an open game world. The player could use a map in the game to see all the enemies and objects of interest they had scouted. The companion application mirrored this map, so you could see all the same important points of interest on your smartphone. There was also a degree of interactivity on this map, where you could initiate some gameplay functions to happen, as a resupply or support (Figure 24). These were used as the examples of when planning the prototypes for this thesis. Of interest was some interactivity with through the companion application, like in Metal Gear Solid 5.
4.2 Game

The idea for the game started as a simple RPG game, with the companion system built on top of that. To simplify development, the visual elements like the UI, game graphics and other extra things like audio were not considered very deeply, instead gameplay elements received the majority of the planning focus. As such, most visual elements would be simple 2D or nonexistent. Albeit the planning incorporating many features and elements into the game, it was decided early on that the bare minimum of functionality in the game itself was to be developed for the game prototype or ignored if even the bare minimum would take too long to develop. The data structure or framework would be the focus of the game prototype, apart from the companion application functionality.

The specific features chose to be developed will be discussed more in the Development-chapter. As all projects do, this one also requires a name with which to refer to it. With the companion applications being related to this, “The Quest for Companionship” or TQoC for short is used as the name. This name will also be used to refer to the companion application and the shared library as a prefix. The planning process first began with the GDD, which outlines the idea, then its structure as a game and then visual concepts.
4.2.1 Overall design

“The Quest of Companionship or TQoC is an RPG game. The objective is to make enough money to buy your way out of an extensive debt by various means. The player will be able to use many skills to make money, like cutting wood and processing it to furniture, mining rocks for metal and gems to craft into equipment that can be enchanted. But the world is also a hostile place, so the best places with the rarest resources are dangerous, filled with monsters and enemies that the player can fight for loot. As the player starts as a peasant, a nobody, he has very little money to his name, no home and barely any food. But he is not afraid of a fight, so he thinks he can become an adventurer. Towns and random characters in the wild offer quests for the player to do, and these are especially useful in the beginning to gain a starting capital. The player must unlock or discover general skills through quests. These skills can also be mastered through quests after enough experience has been gained for those skills, which grants the character special benefits related to that skill. The game is very mouse driven, and many actions are done using the mouse. The players view is centered around the player character, and the player cannot see outside it”. (Appendix 1/1.)

The game world is 2 dimensional, meaning everything in the game exists on the same depth level for the player. Structures and objects in the world may block movement, and there is no way using normal means that the player or other NPC’s in the game world can move through them. What the player and NPC’s interact with directly are the levels, where they move, interact with each other, points of interests and see details of the world. (Figure 25). This level is a part of the larger concept of the game world, in which the player can move between levels on a map above these separate game levels. Moving on this higher-level map works differently from normal, being much simpler and restricted to single increment movements. This map is a grid and the player can move to any adjacent grid tile. (Appendix 1/3.)
The human characters and to an extent all character like entities in the game would have the same information attached to them. All characters have attributes, that determine their effectiveness and strength at the different tasks and aspects of the character. All characters in some capacity have relations with other characters by the way of a faction system, where characters can be hostile, neutral or allied to each other. Characters can use and equip different items. More detail on inventory, items and equipment can be seen in Appendix 1/4, 1/5 and 1/6. The design of the human characters was to be simple, and so that different equipment was easy to distinguish from other equipment, as seen portrayed in Figure 26, where this is a simple character wearing a leather hat, armor, gloves and a metal sword. Character statistics consist of 3 distinct parts; basic like health points, stamina; attributes like strength and agility which affect how good the character is at melee combat; skills which determine how good the character is at specific non-combat tasks. More information and details on the world and levels in the game can be found in Appendix 1/3 and ¼ which cover those.
The combat and AI are very simplistic, working on a principle of automatic attacking if the character has a valid target which is also within range. Most NPC’s would work as basic AI that will attack characters hostile to themselves. However, the unique feature in TQoC is companion characters, which the player can control in moderate amounts, like giving simple follow, stay or interact with an object kind of command. Most NPC’s would not use their skills, at least not visually, but the player could use these companion characters to use their skills to receive resources for crafting or other purposes, ultimately to make more money more quickly than if the player was alone. Their main purpose however would be to assist in combat. (Appendix 1/6.)

4.2.2 Development plan

The game consists of simple concepts, but they are still difficult to structure properly in terms of a game in Unity. In Unity, one needs to think of large chunks of elements as separate scenes, like the main menu of the game, the gameplay on a level itself or the movement in the higher-level map. Game development in Unity works based on GameObjects, which contain child GameObjects and components like scripts which provide the actual functionality. These are used to structure or group up different elements. The Figure 27 shows how these elements were used to roughly plan a Unity structure for the game idea. On it, the blue objects are GameObjects, with yellow ones being more important, but still GameObjects. The green objects are Unity Scenes.

As the functionality comes mostly from the C#-script components for these, a class diagram for the whole game was made to assist planning the development based on the Figure 27. This and some other diagrams detailing specific elements like characters or items worked as an initial C#-project structure for the game and were not updated to match what was implemented for the game. There was also a much more in-depth class diagram used, but it was not added as an Appendix.
While the game part of TQoC focused on the underlying data structure of it, the mobile application part of it would instead focus on the its use by a user, meaning its UI, and moderately on the visuals. As the main goal of the companion application was to display game information to the user, its design would largely depend on the amount and complexity of the data structure of the game. Additional interactivity was another goal to be in the companion application apart from the viewing of the game character and level data. Like with the game project, many of the features envisioned in the planning process for the companion application were not developed because it was similarly a prototype. The ability to view the player characters statistics and view the level was a prioritized feature, but the ability to control companions remained a possible goal, which depended on the game side of the project receiving the feature first. Visual elements also remained very basic.
4.3.1 Overall design

The planned functions of the companion application can be seen in Figure 28. There, some basic functions like the main menu hub from which the user would navigate to the other elements and use the functions there, application settings and connecting to the game for which this companion application was for. These would be very simplistic, as they would have very few required functions. The main menu would solely be for navigating to the different elements in the application. The settings menu would be a simple menu for application settings like the connection aspect or audio levels and other normal settings. The game connection menu would be very simple, as the connection process would be largely automatic.

The more game specific elements would be the companion character control, a player character view, a quest center, information bank and game map. Companion control and the character view would be the areas which the player could have some control over the game, as in those the user would be able to control and monitor the companions in the game like setup those characters equipment and set orders or view and change the player characters equipment. The quest center would be where the user can track active and inactive quests for their character. The information bank would be a centralized location for information regarding different game elements. The game map would be a 2D visual representation of the active level in the game, with different entities like objects, enemies and the player visible on it. All these elements in Figure 28 would be different views on the mobile device for the user.

Figure 28. Proposed structure of the mobile application
The player character view would be the place where the user of the application can view their game characters information and change some aspects of it. Basic information like health or hitpoints and stamina would be visible, and all the skills and their current levels could be seen, as can be seen on Figure 29. The skills would be displayed below the most important information, hitpoints and stamina, as those dictate the immediate survival in a battle. As there would be more skills than could be displayed in a single view, the list of skills would be scrollable. This data would be up to date, as it would be updated when it changes in the game.

In addition to that basic information about the player character, the inventory and equipment can also be seen, and the equipment of the character can be changed through the application, and it would be updated in the game itself. This can be seen in Figure 30, where there are different slots for items on top of an image of a game character. Below that section would be a grid of all the available item slots of the characters inventory, which could be filled with many different items.
4.3.2 Development plan

As the author was inexperienced with Xamarin and mobile development in general, creating a development plan like a class structure or how to separate different elements was more difficult, as one can do it differently using Xamarin. In Xamarin, there are pages, which on a mobile device are the view a user can see and interact with. What the user navigates between are these pages, for example, there could be a home page where there are buttons for different elements like the settings of a mobile application and one could get from that home page to the settings page, where there are buttons to change different settings. There are multiple ways of navigating between pages, but the simplest way is setting the main page of the application, as it is the one displayed currently on the application. The plan for the functionality of the mobile application was very basic. As seen in Figure 28 which was for the overall mobile application structure, Figure 31 shows that in a more detailed way and with Xamarin concepts considered, like the navigation path and page templates used for some of the pages. The main page would work as the home page, from where navigation would occur and return to from the other pages. Data that was required from other pages was planned to come through
the navigation system, as it essentially created a new instance of a page every time the user navigated.

![Diagram](image.png)

Figure 31. Navigation flow for the companion application prototype

### 4.4 Shared networking library

The planning for the shared networking library occurred during development, as it was a response to a demand for a separate library for the networking side of the whole TQoC project. As the library’s job was small in scope, only needing to handle the networking process and possible shared data types, designing its structure did not take long. Due to author’s inexperience with networking, designing how the game and companion application would connect and through that how the shared library would work to establish that connection and move data was difficult. General structure can be seen in Figure 32, where it is mostly split into two larger elements, UDP and TCP which then split to their respective client and server implementations and the data they share.
Figure 32. Structure of the shared library with networking features and datatypes

The UDP element is the broadcast function using which the companion application and game find each other on the same network and establish a TCP connection to transfer data around. This data would be in specific data classes, the Game packet and the Server information packet, which are known by both the client and the server but originate from the server. In between the UDP broadcasting and TCP data transfer, there is an authentication process that is required by the server. This is simply a randomly generated number appearing in the game application, which is inputted into the companion application. This is to provide a small amount of security but more importantly an identification method in the case there are multiple companion applications and games active in the same network although this feature of multiple clients and servers was not to be tested or implemented. (Figure 33.)
In this chapter, the development of the prototypes and the shared code library will be discussed about in their initial designs using the features of the development tools, finished states and the issues and solutions during their development. Even though the development phase attempts to follow the design for the game, mobile application and shared networking library, the concept itself and many of the components required be used to fulfill the features required by the companion application concept were very new to the author. This means a great deal of learning using documentation, tutorials, and experimentation occurred during this phase and many issues encountered were due to inexperience with the subject. To reiterate from the planning chapter, the objective was to develop two prototypes and the shared library to the point where the game and mobile application could connect to each other and transmit data specific to the game that would then be shown to
the user, which could be used on the mobile application instead of a UI menu in the game.

Like in the planning phase, the development of the Unity game and the Xamarin application initially occurred in a linear and sequential way, from the game to the mobile application and then the shared library. At that stage, connecting the Unity game and Xamarin Android application together using the shared library caused the testing of all three require modification to function properly as issues with implementations surfaced.

These prototypes forego many visual detail or efficiency of the code in the final, to aid the focusing of development during the project, so that features could be tested practically and that the development itself would take less time programming or work on elements not important to the main planned features of the prototypes. During the development process some of these features were not implemented, as development started to take a longer amount of time and effort. These included things like a large amount of the interactivity between the mobile application and game, much of the gameplay elements that would have been present even in a prototype stage and various smaller features in the mobile application, such as settings. These resulted in the game prototype only offering very simple movement and combat with very simple spawning enemies. It could also communicate with the mobile application by sending data. The mobile application could show the player characters statistics and the map the player character was in. In that map, the user could see the player, enemies and the individual types of tiles of the game world represented as colors.

5.1 Unity

The main features of the game prototype were RPG elements, saving and loading that save, simple player movement, simple AI enemies that spawn in, simple combat between the player and the AI, simple tile level and the companion application specific connectivity. In the background, this resulted in a moderately complex data structure for all aspects of the game such as the tile game world and especially the characters. Initially in the development there was more emphasis on implementing the RPG elements into the
prototype, elements like skills, items, using items like equipment and friendly AI through a flexible faction system. All of these proved to be too large for the scope of this project, as the data structure alone would have been difficult to develop for all these elements, as these would have had a viewing counterpart in the companion application.

The game prototype implemented the data transfer related to the companion application concept, but most of it is done through the shared library where the Unity project only has a menu for using it and formatting data. More in-depth descriptions of its functionality will occur in the shared library chapter.

Development for the game started at a framework level to structure the core of the game in Unity. The diagrams outlined in the planning chapter were used as a model for Unity GameObjects, prefabs and the C# classes for this initial structuring. All the class elements in the class diagram were implemented as C# classes. At this point in the project the classes started out as templates, where their names, the possible variables and other references to other classes required by them were contemplated and included in them for future use. This was done to avoid complications with names and that there was a clear purpose to a specific class in the Unity project to begin with. This process also helped understand if classes appeared to be too complex and needed refactoring to new classes as this was less evident in the initial planning phase.

Majority of the development time spent on the game prototype related to the data structure and then the companion application aspect of sending it through a network protocol. The data structure being a large portion of the prototype also lead it to be the most problematic part of the prototype, but it was not the only source of issues. Many of the features or their scale used in this prototype were new to the thesis author, such as tilemaps and interacting with the code, the whole RPG character data structure and saving the game world including this character data structure and the tilemap, which presented issues not anticipated initially.
### 5.1.1 Data structure

The general plan for the data structure as a whole in Unity was for it to use ScriptableObjects and inheritance to have a flexible framework so that adding new variations for elements was simpler. ScriptableObjects in Unity are data containers that are used to save or contain large amounts of data. They are different from GameObjects in it that while those are elements that exists only during when the game is running, ScriptableObjects are references to files on disk, so they persist between sessions. While one is not explicitly allowed to save to ScriptableObjects during runtime on deployed builds, one could circumvent this for example like it was done in the game prototype, which is saving the status of the important data into a JSON-format file. This can then be loaded during runtime and changes be made to ScriptableObjects and GameObjects based on the loaded data. (Unity 2019a.)

The main parts of the data structure were characters, their RPG elements and the game world with separate levels with the content of those levels such as AI characters, AI spawners and level objects. Many of these elements were separated into many different ScriptableObjects, especially if they were going to be reused in different places such as skills where individual ScriptableObjects in the main skill container ScriptableObject. Characters were a notable example for this, as every actor in the game would be some kind of derivative or instance of a character. This made it so that you could make reusable components such as movement values, skill values or basic hp and stamina current and maximum values that you could mix around to make new characters for the game. The ScriptableObject for a character’s skills is shown in Figure 34, which itself contains more references to other ScriptableObjects, in this case a default skill list container, and character skill elements. The object seen is to define the skills for types of characters, such as the player or specific types of NPC’s. The Unity Editor interface seen can be used to set base values for this character type. All of these share the definitions of skill element ScriptableObjects.

The largest problem encountered during the development of this data structure apart from the scale of it was the saving and loading of game data from a JSON file. Initially, saving and loading was attempted using Unity’s
internal JSON serializer, that would allow the game data from GameObjects and C# script components to be extracted into a JSON object. This could then be used to create a file on disk. This did not work, because of how deep most of the data on the game prototype is. This depth equates to the hierarchical level of where the code is. For example, there could be a character class with some data, and this character class is used as an object variable in a level class, that additionally contains some other data. Using Unity’s JSON serializer in the level class would only include the information contained inside itself, which would not include data inside the character class, but would contain a simple object for the character class referenced by the level class. As this was not a bug, but a limitation of Unity’s JSON serializer, an external JSON serializer library JSON.NET was used.

![Figure 34. Editor view of a character’s skill ScriptableObject, that contains its skill types and their current statuses](image)

### 5.1.2 Character

Characters in the game prototype consist of the character class and several components grouping certain features together, such as movement controller, movement statistics, basic statistics, attack controller and skill container. In
Figure 35 the general structure of the character prefab, specifically the player, can be seen in the Unity Editor interface. Additionally, the main class of the character prefab can be seen with its ScriptableObject references, in this case under the Player class. Items, the inventory and equipment were going to be added, but creating a flexible system for them proved to be woefully difficult as flexibility was key, and in RPG games item types and their uses can be very different and complex. Whenever something wanted to move or interact with these characters, they would have references to the components contained within that character, such as the player controller would take the players inputs and call methods from these components such as moving. The game prototype would have very simple directional keyboard movement controls and the ability to move to a target with the mouse, which used raytracing to accomplish this, which is basically the game checking that there is nothing in between the player and the target that would block movement.

The experimental Unity system was used for the player input development. This has since this thesis concluded to become a feature in one of the newer stable Unity versions, 2019.2 (Unity Blog 2019). Apart from its use being completely different from how input was traditionally done, there were no major issues with using it. Largely problems appeared in implementing a character when attempting to properly manage the components of a character. Especially ScriptableObjects were difficult to use, as there was not a simple way of creating ScriptableObject assets, such as different default basic statistics or a certain skill group configuration for a specific type of NPC. To do this, one has to do Unity Editor scripting, which means developing code for custom editor functions and this was done for the prototype for making different skills configurations. Editor development is however a whole other system, and it is very unlike normal Unity game development, so there was not a focus here, but it is very useful for creating shortcuts for creating groups of assets or other development features inside the Unity Editor itself.
The game world was planned to be simple with a world travel feature, where you could move between adjacent levels. Levels themselves would be constructed of a tile map and its tiles, level objects, boundaries, areas on the edges for this world travel function, NPC spawners. This whole world would have been a ScriptableObject to simplify saving and loading it from a file. The world travel features did not turn out to be necessary for the thesis’ scope, so it was dropped due to that and because it was also not a simple feature to implement. In Figure 36 some of the implemented elements contained in the level container class is seen, such as the lists of all the entities and objects contained inside the level. The tilemap system in Unity was used, but with custom scriptable tiles. This tilemap is essentially a 2D array of tiles, which have some visual aspect to them like sprites which is a simple image. Their customized aspect was very minimal, as only a tile type variable was added to them. With the custom tile class being used a tile could be identified to be of a certain type such as ground, water or other variants. (Unity 2019b.)
As the game world was to be very simple, not many problems were encountered during the development. Scriptable tiles were problematic however, as it was difficult to get the custom scriptable tile or specific tile information from the tilemap directly. Getting the custom tile information turned out to be simple to do, as all you had to do was to convert the wanted tile to be of the new tile class type, but Unity tilemap documentation for getting a tile from a specific coordinate and a list of all the tiles in a basic TileBase object type was not simple. This TileBase return value was needed because it is what is the base class for the custom tile so that it can be converted to it. Tilemap’s cellbounds.allPositionsWithin was used to get a list of all the positions in the tilemap, and this position data was then used to create a list of TileBases using Tilemap’s GetTile method. With this, every tile of the level could be accessed, and their type be determined. (Unity 2019c.)

5.2 Xamarin

The mobile prototype consists of basic navigation, basic classes for each separate page, player statistics, the world map and connecting to the game data transfer client. Many of the features cut for the game prototype were also then cut for the mobile prototype, such as showing the additional RPG elements and settings. Development in the mobile prototype began in a similar
way to the game prototype, where it began implementing the whole framework. There was however no specific class diagram to base this framework off, as the lack of experience with Xamarin or even mobile development prevented that. Instead, the general diagram of the components of the mobile application was used as a guide for the initial framework building process. Visual elements and general function as a mobile application was kept very simple for the prototype, and so the only part of the application remaining to be implemented was the utilization of the shared library to receive data from the game client and display it.

Testing in android development turned out to be a huge issue in terms of networking, as emulated android changed slightly how networking between the game and mobile application would happen, and this was not immediately apparent in the documentation. This testing was much easier to do with physical devices, but before a separate wireless local network such as a router was used for testing, it still was difficult to test networking.

5.2.1 Learning Xamarin

Before development occurred for the mobile prototype, the use of Xamarin was studied using Xamarin specific mobile development tutorials and Xamarin documentation. Xamarin is an all-inclusive mobile toolset and well documented, so no other source of guides or documentation such as for Android development was necessarily needed. Specifically, the “Create a Single Page Xamarin.Forms Application” tutorial was used as an initial guide (Microsoft 2019a). This Xamarin.Forms is the cross-platform specific toolkit where code done once works on Android, iOS, and Universal Windows Platform. Generally, elements that may be useful for the actual mobile prototype were researched during this, such as displaying data on the application and how to update it, how Xamarin handled different views or navigation in the application and generally how UI elements were created. The applicable elements from Xamarin were data binding (Microsoft 2019b), which allowed for an easy way of keeping a variable visible to the user updated and XAML pages (Microsoft 2019c), which contained the visual elements and the core code for a specific view or in Xamarin terms a page in the mobile applications.
5.2.2 User Interface

The UI for the mobile prototype was to be very simplistic, where visual elements were either simple numbers, text, shapes like squares or buttons. Because of the RPG data set, there were many elements that needed to be available for the user to see, but they had to be divided into separate XAML pages. The XAML pages could be of different types, which added functionality to the default simple types such as a tabbed page which included sub views on a page. XAML pages are separated into two different components, with the first one being an extended XML document that is largely for constructing the visual face of the page with tags, but it also includes some references to data binding datasets it has access to or other Xamarin concepts.

The second part is the actual code behind the XAML page, which is executed when the page is used. The prototype was to be comprised of the home page with navigation paths to the other pages; characters statistics like basic variables, equipment and inventory; a data transfer client menu; and the game world map. This home page would act as the navigation hub, where one would get to all the other pages, and back from them to the same home page. This can be seen in the picture with separate buttons. The other pages were generally simple pages for displaying information. Navigation can be enabled in multiple ways, but in essence, Xamarin works on the basis of that the XAML page visible to the user is the main page of the application, and it can be statically accessed from anywhere in the application (Figure 37). By changing this you also change the view. Xamarin provides more elegant ways that abstract the direct change of this, but it is quick and simple to change it in code. (Microsoft 2019d.)
5.2.3 Game data view

Data binding is one of the main ways of keeping visual information updated with an underlying variable. This is used in the mobile prototype to display character information that is updated with the incoming game data from the server. As outlined before, the mobile prototype needed to display character information, and the game world map with the player, enemies and tile types visible. Data binding works well for this purpose, but it is very specific at its simplest form. What this means is that attempting to keep a large dataset up to date using this creates a lot of data binding specific code. For example, there needs to be a specific property for a certain variable, which is then referenced in the XAML page this data is required in. When there are a lot of separate variables, like seen in Figure 38, needing to enter all of these to that XAML document quickly makes it long and hard to read although adding such elements should be done from code for that reason. (Microsoft 2019e.)
Apart from the actual data transfer client part of the mobile prototype, the most difficult part was to make the game world map visible on it. The view for the map was planned to have been objects at first, and the ability to view specific elements on the map, or in some way get simple information on them. The prototype ended up just having the player, enemies and the tiles of the game level visible. The tiles would be represented by different colors and the player would be a different color than the enemies, as can be seen from Figure 39, where the player is green, and the enemies are red.
As levels were planned to be of varying sizes and random, the map page would receive the data for every single tile to build the map. The first problem was that Unity’s default coordinate arrangement for a tilemap is different from the way UI elements in Xamarin are given a position on the smartphone screen. To convert it the axes had to be flipped and mirrored. Additionally, coordinates in Unity are centered in the case of this game level, but in Xamarin, there are no negative coordinates. For example, all Y-axis and X-axis elements in a positive number were flipped to be negative. All the coordinates were also reduced in size, as the smartphone screen is smaller. One other area that was difficult was to create the UI element squares or Box views to be the tiles (Microsoft 2019f). It turned out not to be easy to have them use coordinates instead of relative measurements. The map is also always created from the ground up when the map XAML page is opened in the application, which leads to performance issues. Due to time constraints, a persistent map that would remain in the background was not implemented.
5.3 Data transfer connection

The shared data transfer library consists of the data transfer features utilized in both the game and mobile prototypes, connection authentication and a shared dataset. Although the game and mobile prototypes had a shared knowledge of the data types and enumerations, this was not utilized fully in the end because of time constraints. The data transfer features consisted mainly of a UDP broadcast discovery and a TCP-connection for the actual data transfer. There were packet C# classes that would contain data for both phases of connections. There were client and server versions for both protocols for both the mobile application and the Unity game, where the mobile application was the client, and the game was the server.

As an overview for how the shared library functioned, the server was started to broadcast its service on the local network using the UDP protocol. This broadcast sent the details for the TCP connection to everyone on the local network. This information would be used by a UDP client on the mobile client to then attempt a TCP client connection to the TCP server present on the game server. For this connection attempt to work, the server generates a number for authentication purposes that the mobile client should send. If this succeeded, if the connection otherwise functioned, data will be sent between them.

Because the networking programming was done at a low level with no higher-level networking library used to abstract the creation of such an interaction, it presented many problems during the development, as the author was very unfamiliar with programming networking at this level. Data is converted from the game to bytes and then back to variables in the mobile application. This proved to be problematic as the byte range where the data was hardcoded, and very needlessly complex to work with. It is also very susceptible to change, as both need to change. This could have been fixed by sending a JSON file over the TCP converted into bytes, but this was not done because of time constraints.
5.3.1 Game Server

The game server is responsible for sending the game data to the mobile client. Before anything is sent through to the client, the game data needs to be converted into a byte array, which is then sent to the client, in multiple parts if needed. The size of the byte array is sent first, with the other data being sent afterwards. This data contains things like detailed player character information, level tile information and AI NPC information in the level. This data as a byte array needs to be sent and received in an exact order by the server and the client respectively. The overall function of the game server communicating with the mobile client follows the diagram outlined in the planning subchapter for the shared library, where the client finds the active server and attempts a connection (Figure 33). The server also works in the background, meaning when the user returns to gameplay it is still running. In the Figure 40, the menu for the server can be seen where the user has a simple start and stop button for the servers. For debugging, there are some indicators visible for the power status of the UDP and TCP servers, and a shared status as to what it is currently doing.

![Figure 40. Server menu in game prototype](image)

There were multiple problems with the planned implementation. As mentioned before, the use of a continuous byte array with a specific order for sending and receiving is very inflexible, and prone to errors when changes are made to it. UDP Broadcasting something on a network is not best practice as it is likely
to flood the network or easily propagate outside of it with devices on those other networks receiving data they do not need to receive. This flooding was apparent when using WireShark on the local network the server was running on. Nevertheless, this was an easy and simple way of achieving a service that a client connects to without specific information, other than the UDP port for the server.

5.3.2 Mobile Client

The mobile client is responsible for finding the game server and receiving the game data from it. Similarly, it operates as depicted in the Figure 33 diagram although the user must enter an authentication code to be used before starting the search process. This can be seen in Figure 41, where the mobile application user has a simple start and stop buttons, and debug status indicators for the current action being done, TCP and UDP clients. Like the game server, this client runs in the background so that the received and converted data can be viewed while it is updating although this only works for the basic information and skills in the prototype. Initially receiving the UDP broadcast was problematic, as the use of it in C# was unknown at the time. The problem was that if the UDP client didn’t have any activity, meaning if it had not sent anything, it could not receive broadcast packets. The reason for this was not entirely clear, but it likely due to some oddity in the used code for configuring the UDP client.
6 CONCLUSIONS

The objective of this thesis was to explore the design of video game companion applications and how this could be taken into account when designing the game itself the companion application is for, the companion application and how it would add functionality for the game. During the research process, it was discovered that there was not much previous public research into companion applications specifically. Scientific data could only really be found related to other mobile applications used for things like media consumption such as television or movies, and statistics on household’s gaming platforms and tastes instead of video games. Older companion applications proved to be difficult to research when their development had been discontinued, as their presence was commonly removed from the most official sources like the company’s own website and the Google Play store or iPhone App store. Account-based companion applications that were discontinued were also commonly unusable, as the servers used to host them would not work through the application. This is something that is also observable in some older mobile applications.

Research in American households, a good approximate for other western countries, show that there is a large portion of households who both own more dedicated gaming platforms like gaming consoles or PC’s and smartphones for playing games. This means there is a sizable market for combining the use of the two and the companion application concept has been in use since the early 2010’s and is still used, with varying amounts of releases for some games. This is however mainly done by larger game developer studios or publishers, with there being a lower amount of indie games which utilize companion applications. As this thesis was not focused on the concept, the reason for this is not explored further. One possible conclusion to this is because the development of a commercial mobile application requires money, so an indie developer is unlikely to create a companion application solely for a game. Especially when most mobile applications and especially companion applications are free, and their use largely requires the ownership of a game. There have been some, where the companion application aspect of the
application is a secondary feature, which work as their own independent games, which is an interesting variant on this concept.

Designing a companion application that is compatible with the core gameplay of the game turned out to be difficult, as players have been shown to dislike platform exclusive elements such as unique unlockable items from the companion application. A good companion application also requires some sort of purpose or benefit, like making an action easier to do in the game, but having an unfair advantage is not good for the player experience either although that is only a real issue in multiplayer games. These are hard to do in something that is supposed to be purely auxiliary activity to the main game and almost always free for someone to use, but also bring tangible benefit for a company to actually want to develop it.

One problem with companion applications is that they are mostly a separate development project from the game itself. This means they have to replicate the changes done for the game, likely from scratch as the game and mobile application are usually incompatible programming wise. This didn’t come apparent until the prototypes were being finished simultaneously, as development regularly shifted between all three projects to make modifications and test those changes. The game designed for this thesis is likely going to remain a prototype, as it ended up being a flawed game. As a game, it was combining too many features, but the companion application features would’ve offset some of the complexity. It however likely would not have been great as an overall experience.

There were many issues with how the planning and development of the prototypes was approached. For one, the initial plan for the game was overall way too complex. The RPG genre appealed initially, as it would be a great way to add complexity to test, but that complexity directly caused the planning development to take much longer, and as a result both development and planning took much longer than expected. Because of this extra time needed, many of the desirable features did not have enough time to be implemented, such as actual interactivity between the two platforms. In retrospect, this would have easily been avoided had the scope of the prototypes be more properly outlined and by having the idea for the game be simpler. Another
significant waste of time during the development was because of the very minimal use of external libraries or additional tools over the game engine and IDE's. Especially using a networking library would have significantly simplified implementing the networking elements for the prototypes.

6.1 Prototypes

The prototype development was a good learning experience personally, but as what someone could learn from it is that the difficulty depends largely on the function of the game, companion application and cross related features such as the transfer of data or commands. In this case, the game and companion application prototypes worked on a local level sharing of data, so they interacted directly with each other. Many companion applications of larger games use an account-based system which is likely simpler to develop as there likely is infrastructure already present for this kind of interactivity over the network or otherwise possible to with small modification repurpose it for the new game and companion application. Developing networking code at the level that was done for the prototypes was also not realistic, as the use of networking libraries would have simplified the development of it greatly.

The resulting prototypes themselves were close to what was required from them although the more advanced features such as the direct control of game elements would have been valuable to test in practice. Development of the game prototype however took much longer than planned, as certain elements received more attention than was necessary to test a feature, as many elements like the data structure for characters became flexible and ready for additions that would never have actually been implemented for the prototype. Much of this is because of the ambitious planning and design, even though that was only supposed to be for contemplation. Overall the prototypes explored how one might approach a companion application like feature set for a game. The RPG genre for the game prototype also introduced many of the problems with a large data structure, but RPG games have usually been the games to receive companion applications because of that complexity and large data sets.
6.2 Future development

More detailed research should be done on how many people use these applications, and how that has changed to be able to properly discern if the companion application concept is viable in the future as that was a very minimal subject in this thesis. There is definitely more to explore in ways of designing companion applications, like the new and different ways of interacting between the game and a mobile application. Especially involving more of the smartphone in the interaction, like using the gyro sensors to turn the camera around for a secondary viewpoint, using its vibration or audio capability as a detector for something, such as a radiation monitor in post-apocalyptic survival games or using the touch screen in new and inventive ways. One interesting way of involving the smartphone in the gameplay is that some games even have the smartphone as the sole controller.

There is also more to investigate how do companion applications statistically benefit the developers or publishers of a game. Such as how many users that download the companion application own the game and how much do they use the application? More unique companion applications like Tom Clancy’s Ghost Recon Wildlands HQ, which mainly functions as its game, should be analyzed for how many of its users that do not already own the game that then went on to purchase the game.
SOURCES


WATCH_DOGS COMPANION APP REMOVAL. 2018. Ubisoft. WWW-document. Available at: https://support.ubi.com/en-


FIGURES

Figure 1. Distribution of gaming platforms in USA, 2018 among households where games are played.

Figure 2. Map view in the companion application for Assassin’s Creed 4 (Henderson, R. 2013)

Figure 3. View of the tower defence minigame in Star Wars Battlefront (Electronic Arts, 2019a)

Figure 4. Player’s points of view in Fallout 4

Figure 5. Players in-game menu for map and other things like the inventory

Figure 6. Management game in Ghost Recon Wildlands HQ (Ubisoft Entertainment, 2019b)

Figure 7. View of a player’s statistics in Battefield Companion (Electronic Arts, 2019b)

Figure 8. View of a player's statistics in Call of Duty Companion App (Activision, 2019a)

Figure 9. Home page in Final Fantasy XIV Companion (Square Enix, 2019b)

Figure 10. Map view in World of Warcraft Companion App (Blizzard Entertainment, 2019b)

Figure 11. Fallout Pip-Boy mimicking Fallout 4's menu interface (Bethesda Softworks, 2019b)

Figure 12. Main navigation view in RDR2 Companion (Rockstar Games, 2019b)

Figure 13. Map view in Ghost Recon Wildlands HQ (Ubisoft Entertainment, 2019b)

Figure 14. Equipment view in Destiny 2 Companion (Bungie, 2019b)

Figure 15. View of a generated solar system game match in PA.

Figure 16. View of the starting planet in PA, where the purple is the player and red elements are an ally of the player.

Figure 17. View of a project in Unity Editor 2018.3

Figure 18. Unity Hub - Unity game engine installation

Figure 19. View of a C# project in Visual Studio 2019

Figure 20. Visual Studio 2019 workload installation menu

Figure 21. View of a Git project in SourceTree

Figure 22. View of local networks and their activities as seen by WireShark

Figure 23. View of the player's equipment in Fallout Pip-Boy

Figure 24. View of the game map in MGSV companion app (Softonic, 2019)
Figure 25. Proposed style of the visual aspect of a level with some level elements
Figure 26. Proposed look of a humanoid character
Figure 27. Proposed structure for the Unity game
Figure 28. Proposed structure of the mobile application
Figure 29. Proposed look of the character view in the companion application prototype
Figure 30. Proposed look of the character equipment views in the companion application prototype
Figure 31. Navigation flow for the companion application prototype
Figure 32. Structure of the shared library with networking features and datatypes
Figure 33. Proposed flow of connection between server and companion client
Figure 34. Editor view of a character's skill ScriptableObject, that contains its skill types and their current statuses
Figure 35. Editor view of the player character
Figure 36. Editor view of the game level and custom scriptable tiles
Figure 37. Main view of the companion prototype with navigation buttons
Figure 38. Character view in the companion prototype
Figure 39. Game level view in the companion prototype
Figure 40. Server menu in game prototype
Figure 41. Client menu in companion prototype
Overview

The Quest of Companionship or TQoC is an RPG game. The objective is to make enough money to buy your way out of an extensive debt by various means. The player will be able to use many skills to make money, like cutting wood and processing it to furniture, mining rocks for metal and gems to craft into equipment that can be enchanted. But the world is also a hostile place, so the best places with the rarest resources are dangerous, filled with monsters and enemies that the player can fight for loot. As the player starts as a peasant, a nobody, he has very little money to his name, no home and barely any food. But he is not afraid of a fight, so he thinks he can become an adventurer. Towns and random characters in the wild offer quests for the player to do, and these are especially useful in the beginning to gain a starting capital. The player must unlock or discover general skills through quests. These skills can also be mastered through quests after enough experience has been gained for those skills, which grants the character special benefits related to that skill.

The game is very mouse driven, and many actions are done using the mouse. The players view is centered around the player character, and the player can not see outside it.

The game uses the game engine Unity, specifically the 2018 version. Target platform is Windows, which is where testing for the games function is also done. The game is not to be particularly graphically intensive nor should it be complicated enough to need a power CPU, so minimum requirements are very low.

Starting Game

Player customizes a character to play as, with randomized statistics which can be rerolled. This character starts with little money, some random starting equipment and little to no skills. This character gets a tutorial quest the player can follow, which will explain everything past actual character movement and using the mouse for actions.

Visual Style

TQoC is a 2D game, with sprites working as the main graphics type. The game is supposed to be visually simplistic, with organic or curves not being too present in sprites. Harsh angles and geometric shapes should be the normal appearance of objects. As the style is mainly simplicity, perspective of the sprites is to be flat and from the front. This also applies to visual effects outside of sprites like particles, but also on sprites themselves for elements like lighting on shadowing. Meaning that they should not make the object appear 3D or from a different perspective such as isometric.
Audio Style

As the graphics are angular, simple and somewhat pixelated, the audio effects and music should reflect that by being simple and made with simple instruments. The general theme of music and sound effects should be loosely medieval and 8-bit or chiptune style.

UI

The game consists of 3 distinct UI groups, the main menu, in-game HUD and menus. The main menu UI should be immersive, meaning it should be executed or enabled on a level to have the game world be the background for the main menu. This should be done additively, with the main menu being its own scene in Unity. Being in the main menu or a in-game menu for the player character should not pause the game.

Main Menu

The main menu for the game is purely to start a new or load a game and tweak some settings of the game. It should have an immersive background, with the game world visible on the level the player will start on if he creates a new character, or if there is save data available, the location of the player in that save.

Game HUD

Consists of elements for the most important player character statistics like HP and stamina used for attacks and movement actions. The current status of the players combat stance and currently equipped weapons should be visible, including the statistics they provide for the player, or the end result of them like the amount of damage the player will do and how much defence the player has. As the game is very mouse driven, there should be some buttons on the screen itself for going into the main menu or adjusting some settings. There should also be a large part of the screen for monitoring the players group of characters or companions. You should also be able to change some things like their stance, or for them to return to formation or near the player on the UI using the mouse. There should also be a minimap of the level, which is at its base just a grid of all the tiles of the level displayed as simple colours like green for grass, grey for rock and blue for water. Special objects like the player, resource locations, NPCs, items, or travel zones would get special icons on this minimap.

Game Menus

Most things will be behind some menus, to keep the game view simple and not too cluttered. As there are many different categories of things, only a single category should be visible at once. Meaning things should be separated into different tabs or other ways of separating content on a single page. There should be views for the player character, companions, maps and quests.
**Gameplay**

**Saves**

The game should have a single save game “slot”, so there is only one character the player can save or load. Creating a new character automatically overwrites this save. The save data should contain all the data for the player character, quest data, world data, companion data and unique identifiable data for save game validation for the companion application interaction.

**Game States**

Because there will be important states of the game that could occur during gameplay, like being on the main menu or game being paused, they should be implemented at the highest level and checked by the most important elements like AI, animations or other time important or independently active elements or activities.

**Companion application compatibility**

The game will communicate with a possible companion app using the TCP-protocol if the device for this companion application is in the same local network as the game. The companion application would request all the data to begin with from the game, and afterwards would receive changes to most data when they occurred in the game, like changes to player attributes or skills, player inventory or NPC stats. Things like locations for moving objects would be sent constantly when the map was open on the mobile device. Level base data would also be sent to the mobile device when the level was changed for the map to adjust. This data would include what types all the levels tilemap are and the elements contained in the level, so that the companion app map can visualize the map based on that data.

**World**

The game world consists of a two-dimensional map, and this map is a grid with every cell being the same size. The size of this map is up to some arbitrary point but is meant to be flexible to future needs or demands for a bigger map, as unused grid cells can not be accessed. Non-accessible squares on the maps can be used for visual purposes, such as appear as impassable water or mountains or other terrain difficult to pass through. Each of the accessible grid cells are levels or zones. These levels are where the player actually controls his character directly. On the world map you can travel to different these different zones if you exit out of a level. This map will display these zones in a simplified matter, where their general tilemap setup will determine their colours or appearance otherwise on the world map. These levels can also have locations or structures on them which can be entered, and these are visible with an icon on the world map. Travelling on the world map is limited to nearby zones and is more or less instant. There should be no danger or events in this travelling.
Level

Levels or zones are where the player actually controls characters and where companions do their actions. These levels are expansive and contain various objects or entities to interact with. The level itself is a tilemap, consisting of different terrain types like ground or dirt, grass, sand, rock, rock wall, river or water and other types of natural terrain. The level will also contain trees, rocks and other natural objects that fit the general theme of the level. These biomes can be desert, forest, cave, coastal and so on. Things that can be harvested using skills will respawn after their respective spawn counters finish, like trees, minerals, bushes, fishes and other natural resources that can be found in the wilderness.

For a player to be able to move on a tile it needs to be possible through the tile type itself. This means that smaller structures need to have their floors be the tiles, they can not only be separate level objects. Tiles can also be walls, in which case they block movement. Levels are also on a single level, meaning there is no actual logical elevation. This can possibly be cheated visually by creating appropriate tile map sprites that appear to take the player above other levels like cliff faces and so on.

Location

There can be inner locations in some levels, like towns, camps, outpost, caves or other structures with something inside like treasure, NPCs, hostile entities, resource locations or potential quests. These can only be travelled to by first going into the level where the locations is in. These locations are also usually persistent, where they can be emptied of high value treasures and rare enemies. But essentially, locations are just more unique levels. These locations are usually man-made, and as such the tilemap will contain much more built walls and finished floor materials like gravel, road, wood flooring.

Level Objects

Level objects can be things like static objects that can not be moved through, static objects that can be moved through like small things on the ground where it would not be logical to be able to move through them. These level objects should be above the levels tilemap, meaning above or on top of the ground and do not cover the entire tile.

Interactive Elements

Interactive elements are special level objects that in some way can be interacted with. These can also be things that either block or do not block movement. These elements can also be purely logical, meaning they are invisible and instead work as triggers.
Player

The player character, as do other NPCs, have a level that is determined by the overall level of one's attributes or in other words combat skills. Some have less effect on the combat level than others. All characters have an inventory and equipment can be equipped. All entities that move and are alive use the basic stats and attributes, characters have skills that they can use. All entities also belong to specific general factions, like neutral, hostile, civilian or specific groups that have special relations to others. These factions are static and can not be changed. All entities also use the combat stance and the different types of attacks they can in combat. AI entities also use different combat stances or personalities, which govern how they will react to combat. Visually, the player will consist of the player character itself, the separate equipment on top of the character, and a visual highlight so that the player can be distinguished from the background tiles and objects on the level like trees or such. This highlight should always be invisible in the case that the player is behind something and the character itself cannot be seen.

Statistics

The characters statistics contain 3 main areas, the basic stats, attributes and skills. Basic stats consist of hitpoints, stamina and combat stance. The character also has a base amount of damage that is done when unarmed, and this is added to weapons based on proficiencies with attributes. Attributes are general skills that improve mostly the combat ability of a character but can be requirements in skills or quests for actions. Skills are the same, where they can be requirements for actions like in quests, gathering resources, slaying enemies or using certain tools or items. Especially crafting is very skill based.

Attributes

The player character has the following attributes: Strength, Vitality, Resilience, Agility, Intelligence. Strength determines your ability with melee weapons by allowing you to equip larger and stronger weapons, deal more damage outright and carry more things. Vitality governs how many hitpoints and stamina you have. Resilience increases your defence against attacks, and together with strength allows you to wield stronger and heavier armour. Agility determines your ability with ranged weaponry and general movement ability and efficiency. Increasing your agility also improves the benefit you get from lighter armour and melee weaponry. Intelligence determines your effectiveness with magical attacks and robe or magical armour. Improving all of these attributes occurs by doing the things the affect, like fighting with melee, ranged or magical weaponry, taking hits or blocking them, using movement abilities and so on.
Skills

Characters have the following skills: Woodworking, Digging, Crafting, Survival, Alchemy, Metalworking, Hunting, Magic Smithing, Cooking and Fishing. Many of them are for specific categories of tasks, like woodworking is to interact with trees and wood in general and be able to use specific tools for the skill, fishing is for identifying fish types and being able to catch them, able to use certain fishing tools. To increase a character’s proficiency with these skills the character needs to use them. Quests and tasks as rewards can also increase these. By mastering a skill, meaning reaching the last level which is achieved by completing certain quests, one unlocks special abilities and requirements for powerful items.

Crafting

Some skills allow you to craft with materials gained from using skills. You can craft various things with each skill, like alchemy being used to craft potions, metalworking to create metal weapons and armour, magic smithing to create magical weapons and magical stones for magic attacks. This crafting is another activity of skills that increases one's proficiency in them.

Attacking

In TQoC attacking is done using 3 different types: Melee, Ranged and Magic. They don’t specifically oppose others, but heavier armour will be somewhat susceptible to ranged attacks and more so magic attacks. Ranged characters usually benefit from using lighter and weaker armour, so they are more vulnerable to melee focused characters. Magic users benefit greatly from using staffs which aren't really useful in melee and armour that is really clothing, so it provides minimal protection for ranged and melee attacks. Magic users are however the lightest and can potentially evade others more easily. Heavier armour and inventory although fairly slightly reduce movement speed.

Inventory

All characters have inventories and items equipped on that character. Characters have a low base number of items they can carry, but this can be improved by equipping a backpack or other larger container item designed for this purpose like a carry sack. These inventories are equally sized, and all items always take up only one slot. A character has 8 equipment slots. First 3 are for armour, gloves, helmet or hat, body armour or clothing. There are 2 slots for the hands of a character, one for each hand. Larger weapons or tools equipped may take both slots. There are 2 accessory slots for amulets and rings, that provide some stat changes or effects. The last slot is for backpacks or carrying containers.
**Faction**

Characters can belong to a few factions, but if they belong to more than one, they have numbered priorities for situations when they are overwritten. Factions govern what can be attacked and what a character will attack when seen. For example, if an NPC is neutral, the player can attack them and factions that are hostile in general will attack them on sight. If they’re a friendly faction to the player like civilians, the player can not attack them, but hostile factions can and will.

**NPC**

NPCs or non-player characters would consist of companions or allies of the player, neutral friendly NPCs which can be traded with, they can give the player quest or guards protecting towns, or lastly NPCs can be hostile entities ranging from humanoids to beasts that attack the player and neutral NPCs on sight. These would have a similar inventory simulator when compared to a player, but they are set with a specific inventory size to hold all their items when they spawn, and so cannot gain extra items. They may carry consumables, which would not free up an inventory slot when used.

**AI**

The artificial intelligence of these NPCs would vary based on their main faction relation to the player. If the NPC is allied, it will follow the player and attack hostile enemies close by, and neutral ones when the player attacks them first, although this may depend on the allies own personal faction relations. Neutral NPCs only attack the player and its allies if the player attacks them first. Only specific neutral NPCs fight hostile entities, namely the guards or otherwise more combat oriented NPCs. All hostile entities attack everyone on sight. Neutral NPCs generally patrol a location, unless they’re set to be more static and guards patrol guard specific areas. Hostile entities patrol their spawn locations in varying complexities, like in circles or specific paths.

**Items**

All entities have an inventory in some capacity, although only the player has a fully-fledged inventory where there can be space that is empty that the player can find an item to occupy. Items can either be used, equipped on the character, or just be used as a part of an action such as trading, selling or buying items, using interactive objects that “consume” a specific item from the player inventory or be used in a crafting recipe which requires a specific item. Some usable items might be consumable and thus will disappear from the inventory when used. Items that have no other use other than to be sold to a trader would be called junk or valuable, depending on its worth.
All items have a base worth that is used during trading. It can be affected by the players skills to benefit the player more. All items also have a name, a description and an icon which the item is displayed as in the inventory. Where its valid items also have a type, category and material the item is made of. Items also have a rarity level, which determines how often characters receive them in the game, and how powerful its benefits and potentially how big the disadvantages are for using this item. These levels go from common, uncommon, rare and then to legendary. For example, for all these would be a sword. The sword is called “The Sword of Adventure”, with a detailed description noting that it increases the strength of the player. It is of the melee weapon category; a cutting weapon type and it is made of wood. It is a legendary item. Inventories for the NPCs are a fixed size, but the players inventory size can vary by what backpack one wears.

**Usable items**

Usable items provide a response to their activation, either directly in the game world like an ally or enemy, or just on the player character by affecting its statistics. These can either be consumable items, which disappear after their activation, items that consume something else to be activated like hitpoints or stamina, but will be retained after an activation, or items that can just be used with a simple requirement or limitation to its activation like a time cooldown. NPCs can also use these items, but consumable items still occupy a inventory slot, but they just cannot be reused. What these usable items would vary between items. They can have both benefits and disadvantages. For example, the player could use a potion that causes the player to lose hitpoints but instead gain stamina for using more abilities. In general, items that consume a player’s resource or have simpler activation conditions would have lesser benefits, unless offset by larger disadvantages. This is to avoid these items to undervalue the benefits of using single use consumable items.

**Wearable items**

Wearable items are all items that players and NPCs can equip in various slots on the character. The default slots are Head armour, body armour, hand armour, leg armour, hand accessories, neck accessory, backpack, left hand and right hand.

The slot determines the majority of the items given benefits and disadvantages when equipped on a character. For example, armour slots are more likely to increase the overall defence statistics of the character, but there may be shields that can be worn in either hand that does that as well. Accessories are also more likely to provide magical benefits, but armour pieces may also do that depending on their type. Weapons are also the major contributor to damage output apart from character skills, but items may directly affect this damage output as well.
Quests & Tasks

The world of TQoC is filled with various quests and tasks the player can partake in to receive many riches and bonus experience for skills. One way a player can begin these quests is finding interactions that give them. These interactions can be for example reading letters or notes found in the world, talking to certain NPCs or interacting with certain objects in the game world like book objects or other static objects that can be inspected.

Mastering skills are always a part of a quest reward. Otherwise the rewards can be general experience to skills used as a part of the quest, monetary rewards like gold coins or valuables and special or great equipment. The monetary and equipment rewards amount are randomly generated for quest, although some quest will have at least partly static rewards.

Progress Quest

The way a player progresses in a quest is trigger states. Quests can have from a few to a dozen of these statuses that need to be triggered sequentially or possibly partly skipped. Such a status can be operating an interactive element in the game world, having an item in your inventory, giving an item to an NPC or using an item on a level element. A quest is considered complete when the player hits the last Finish state and receives the rewards then. These rewards are spawned into the players inventory.

Companion system

The game would have NPCs that the player can befriend to follow them into battle. These alliable NPCs can be gained by completing quests from NPCs that solely exist to become companions of the players. These quests are like personal favours to these NPCs. Some quests may have temporary companions that are removed from the player’s party after the quest or some other point. Once a player gets a companion, it will follow the player around the world. The player has some settings he can impose on these companions to control their actions like how they will follow the player, how they attack enemies, how they use their given items or if they should stay at a location. The player can also select the equipment of the companion they use for combat. Here, the companion can be given consumables for them to use in combat, like hitpoints restoring potions.

Companions are a character like the player, but additionally to that they can level up the same way the player can. Companions level up automatically based on their companion type, which is determined by the NPC turns into one and it cannot be changed by the player. Companions cannot interact with the world other than by combat. Companions will target NPCs that are hostile to their faction, which may in some cases be different from the player’s. Companions can die, and they will die permanently. For game purposes these companions would respawn as NPCs that require the player to redo the companion quest for them if the player wants them back.