

DESIGNING AND IMPLEMENTING A GAME DEVELOPMENT
FRAMEWORK FOR INTERACTIVE STORIES AND ROLE PLAYING
GAMES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OZAN EMIRHAN BAYYURT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MULTIMEDIA INFORMATICS

DECEMBER 2019

Approval of the thesis:

**DESIGNING AND IMPLEMENTING A GAME DEVELOPMENT
FRAMEWORK FOR INTERACTIVE STORIES AND ROLE PLAYING
GAMES**

submitted by **OZAN EMIRHAN BAYYURT** in partial fulfillment of the requirements for the degree of **Master of Science in Multimedia Informatics Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics**

Assist. Prof. Dr. Elif Sürer
Head of Department, **Modelling and Simulation, METU**

Assoc. Prof. Dr. Yusuf Sahillioğlu
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Prof. Dr. Alptekin Temizel
Modelling and Simulation Department, METU

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering Department, METU

Assoc. Prof. Dr. Ahmet Oğuz Akyüz
Computer Engineering Department, METU

Prof. Dr. Ahmet Coşar
Computer Engineering Department, THK

Assist. Prof. Dr. Ulaş Güleç
Computer Engineering Department, HKU

Date:

06 December 2019



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: OZAN EMIRHAN BAYYURT

Signature :

ABSTRACT

DESIGNING AND IMPLEMENTING A GAME DEVELOPMENT FRAMEWORK FOR INTERACTIVE STORIES AND ROLE PLAYING GAMES

Bayyurt, Ozan Emirhan

M.S., Department of Multimedia Informatics

Supervisor : Assoc. Prof. Dr. Yusuf Sahillioğlu

December 2019, 80 pages

Video games are a great medium for storytelling. Taking advantage of this quality is, however, harder than it needs to be. Making role-playing games takes too much time and work on the part of game developers. Existing tools to develop such games, unfortunately, focus more on already proven design features and mostly duplicate the same game mechanics.

This thesis proposes a game development framework to offer a solution to help game developers to build role-playing games in a much easier and quicker fashion. To accomplish this, the framework utilizes common game mechanics and features from existing role-playing games while providing a modular structure to add, remove, change or create new features to built games using a model-view-controller pattern. This modularity also transforms the proposed framework into a highly scalable game development environment to work with for developers.

In this study, an example game is built and several iterations of it developed with this framework to show its capabilities. Also, a user study is concluded to gain the insights of other developers on the framework.

Keywords: game development, framework, role-playing games, model-view-controller, storytelling

ÖZ

İNERAKTİF HİKAYE VE ROL YAPMA OYUNLARINA ODAKLI BİR OYUN GELİŞTİRME SİSTEMİNİN TASARLANMASI VE İMPLEMENTASYONU

Bayyurt, Ozan Emirhan
Yüksek Lisans, Çokluortam Bilişimi Bölümü
Tez Yöneticisi : Doç. Dr. Yusuf Sahillioğlu

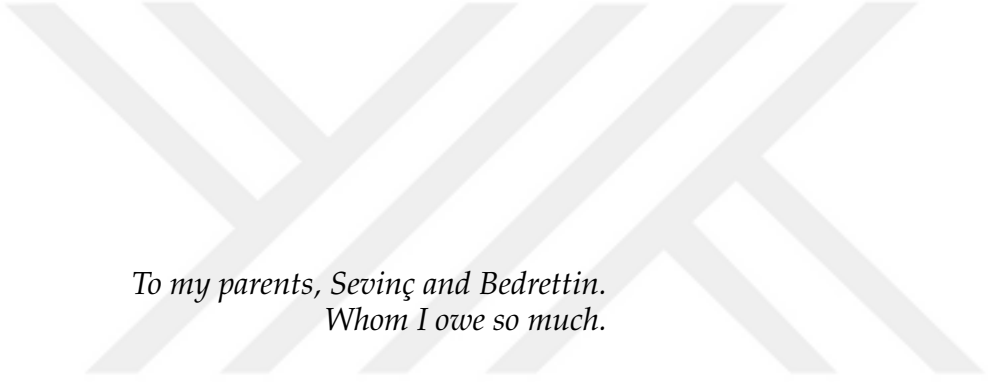
Aralık 2019 , 80 sayfa

Video oyunları hikaye anlatımı için harika araçlardır. Fakat bu özelliklerini ortaya çıkarmak olması gerekenden daha zordur. Rol yapma oyunları yapmak oyun geliştiricilerinden çok fazla zaman ve çok fazla emek gerektirmektedir. Mevcut araçlar ne yazık ki zaten kanıtlanmış dizayn özelliklerine ve çoğunlukla var olan oyun mekaniklerinin kopyalanmasına odaklanmaktadır.

Bu tez oyun geliştiricilere rol yapma oyunlarını çok daha kolay ve hızlı geliştirmelerine yardımcı olacak bir oyun geliştirme yazılım çerçevesi önermektedir. Bunu başarmak için bu tezde önerilen çerçeve varolan rol yapma oyunlarındaki ortak oyun mekaniklerini ve özelliklerini kullanıp, bunları model-view-controller tasarım örüntüsü temel alınarak geliştirilmiş yeni özelliklerin eklenmesi, çıkarılması ve değiştirilmesine imkan sağlayan modüler bir yapı sağlar. Bu modülerlik aynı zamanda önerilen çerçeveyi oyun geliştiriciler için son derece ölçeklenebilir bir oyun geliştirme ortamına dönüştürür.

Bu çalışmada, bu çerçevenin yeteneklerini göstermek için bir oyun hazırlanmış ve çeşitli yinelemelerle geliştirilmiştir. Ayrıca, diğer geliştiricilerin çerçeveye ilişkin görüşlerini almak için bir kullanıcı çalışması sonuçlandırılmıştır.

Anahtar Kelimeler: oyun geliştirme, çerçeve, rol yapma oyunları, model-view-controller, hikaye anlatımı



*To my parents, Sevinç and Bedrettin.
Whom I owe so much.*

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Yusuf Sahillioğlu for invaluable guidance and support.

I would also like to thank Alper Çetin, Serkan Pekçetin and Ayşegül Akyol for their help and incredible technical insights.



TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	v
ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
CHAPTERS	
1 INTRODUCTION	1
1.1 Motivations	1
1.2 Contributions	2
1.3 Structure	2
2 BACKGROUND	3
2.1 Role-Playing Games	3
2.1.1 What is a Role-Playing Game?	3
2.1.2 Computer Role-Playing Games	4
2.1.3 Western vs. Asian Approach	5
2.1.4 Character Development	7
2.1.5 Choices and Consequences	8
2.2 Adventure Games	9
2.2.1 Interactive Fiction	10
3 ANALYSIS OF EXISTING SYSTEMS	13
3.1 RPG Maker	13

3.2	Twine	14
3.3	Unreal Engine 4 and Blueprint	15
3.4	Other Solutions	16
4	DESIGN	17
4.1	Module System	17
4.2	Model View Controller	17
4.2.1	Entity Component System	19
4.3	Technologies Used	19
4.3.1	Unity3D	19
4.3.2	Yarn and Yarn Spinner	20
4.3.3	Tiled	21
4.4	Architecture	21
4.4.1	Game Manager	21
4.4.2	Modules	22
4.4.2.1	Character System	22
4.4.2.2	Combat System	23
4.4.2.3	Dialogue System	24
4.4.2.4	Travel System	25
4.4.2.5	Item/Inventory System	25
5	IMPLEMENTATION	27
5.1	Game Manager and Module System	27
5.2	Combat Module	29
5.2.1	AStar Class	30
5.3	Travel Module	31
5.4	Dialogue Module	32
5.5	Inventory Module	34
6	RESULTS AND DISCUSSION	37
6.1	A Simple Text Adventure Game	37
6.2	Improving The Text Adventure	39
6.3	Switch From 2D to 3D	41

6.4	User Study	44
6.5	Discussion	48
7	CONCLUSION AND FUTURE WORK	51
7.1	Conclusions	51
7.2	Future Work	52
	REFERENCES	53
APPENDICES		
A	UML DIAGRAMS	57
B	CODE SAMPLES	63
C	DEVELOPER TOOLS	67
C.1	Character Generator	67
D	DOCUMENTATION	69
D.1	What is it?	69
D.2	XML Files	69
D.2.1	Gamedata.xml	69
D.2.2	ItemList.xml	70
D.2.3	CharacterList.xml	72
D.2.4	EncounterList.xml	75
D.2.5	MapList.xml	76
D.2.6	SceneList.xml	79
D.2.7	Yarn File for Dialogue Module	79
D.3	Creating a New Module	79

LIST OF TABLES

Table 6.1	What do you think is the most challenging part in developing / most important part in role-playing games?	46
Table 6.2	Which module do you find the most / least useful for developing role-playing games?	47
Table 6.3	In the future, would you consider using this framework to develop your games?	48



LIST OF FIGURES

Figure 2.1 Screenshot from Baldur’s Gate. A CRPG developed by Bioware software.	5
Figure 2.2 Screenshot from Phantasy Star IV. A classic JRPG for Mega Drive/Genesis.	6
Figure 2.3 Screenshot from Zork. One of the earliest interactive fiction computer games.	9
Figure 2.4 Screenshot from Maniac Mansion. A classic point&click adventure game from Lucas Arts.	10
Figure 2.5 Screenshot from an interactive fiction game developed with Twine.	11
Figure 3.1 Screenshot from RPG Maker VX Ace.	13
Figure 3.2 Screenshot from Twine.	15
Figure 3.3 Screenshot from Unreal Blueprint.	16
Figure 4.1 Diagram of the MVC pattern.	18
Figure 4.2 Screenshot from Unity3D.	20
Figure 4.3 Screenshot from Yarn.	21
Figure 4.4 Screenshot from Tiled.	22
Figure 4.5 Diagram of the Combat System Module.	23
Figure 4.6 Diagram of the Dialogue System Module.	24
Figure 5.1 UML Diagram of all the System Modules.	27
Figure 5.2 UML Diagram of the Base Module Class.	28
Figure 5.3 UML Diagram of the Combat Module.	29
Figure 5.4 UML Diagrams of the Agent, CharacterData and Character-visualizer Classes.	30
Figure 5.5 UML Diagram of the Travel Module.	31
Figure 5.6 UML Diagrams of the Travel Node Agent and Travel Party Agent Classes.	32
Figure 5.7 UML Diagram of the Dialogue Module.	33
Figure 5.8 UML Diagram of the Inventory Module.	34
Figure 5.9 UML Diagram of the InventoryManager Class.	35
Figure 5.10 UML Diagrams of the InventoryAgent Class and EquipmentAgent Class.	36

Figure 6.1 Screenshot from our game developed using our proposed framework.	38
Figure 6.2 World map from example project.	39
Figure 6.3 A combat scene from example project.	40
Figure 6.4 A combat scene from example project after the 2D to 3D transition.	41
Figure 6.5 Uml diagrams for the new classes added to the Combat module.	42
Figure 6.6 A screenshot from Travel3d module.	42
Figure 6.7 Uml diagrams for the Travel3d Module.	43
Figure 6.8 Age and Gender charts from the survey.	45
Figure 6.9 Years of experience chart for the participants.	45
Figure 6.10 Which parts of development do you work on?	46
Figure 6.11 How much do you think this framework will contribute to the role-playing game development process?	47
Figure A.1 UML Diagram of all the System Modules.	57
Figure A.2 UML Diagram of the Combat Modules.	58
Figure A.3 UML Diagram of the A* Class.	58
Figure A.4 UML Diagram of the Inventory Modules.	59
Figure A.5 UML Diagram of the Travel Modules.	60
Figure A.6 Uml diagrams for the new classes added to the Combat module.	61
Figure A.7 Uml diagrams for the Travel3d Module.	62
Figure C.1 Main screen of Character Generator Tool.	67
Figure C.2 First screen from Character Generator Tool.	68
Figure C.3 Second screen from Character Generator Tool.	68

LIST OF ABBREVIATIONS

RPG	Role-playing Game
GM	Game Master
D&D	Dungeons and Dragons, a role-playing game originally designed by Gary Gygax and Dave Arneson.
GURPS	Generic Universal RolePlaying System, a role-playing game made by Steve Jackson Games.
PtbA	Powered by the Apocalypse, a gaming system designed by D. Vincent Baker.
LARP	Live Action Role-Playing
CRPG	Computer Role-playing Game
JRPG	Japanese Role-playing Game
MMORPG	Massively Multiplayer Online Role-Playing Game
NPC	Non-Player Character
PC	Personal Computer
IF	Interactive Fiction
MVC	Model-View-Controller
ECS	Entity Component System
XML	Extensible Markup Language
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language
AI	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 Motivations

Video games give us new ways to tell our stories that no other medium has allowed us until now. Games and storytelling are integral parts of the human culture [1]. Throughout history, we use many forms for games and storytelling, but today computers allow individuals to create such games to offer players exciting new experiences. It is a relatively new medium for us, and we are still discovering the extent of the possibilities of this new medium.

Video games began to be a part of the mainstream culture at the same time they became a major entertainment industry. The video game industry has started to generate more income than the film or music industry does today. The video game industry generated a record \$36 billion in revenue in 2017 [2] when box office numbers for North America in the same year was just \$11.7 billion [3] and each year this gap increases.

As regards to their cultural impact, many began to argue whether or not video games are or could be an art form. Notable film critic Roger Ebert famously declared video games as a non-artistic medium [4]. While many others have supported his claims, many opposed their counterarguments. Even though it is not settled whether video games are an art form or not, it can be stated that they inherit enough merits that enable this argument in the first place.

Video games also could be great tools for education [5][6][7] as well as entertainment such that there is a clear connection between the best practices in game design and current learning theories [8].

To achieve this, computer game developers use a wide variety of skills from very different engineering and artistic fields. In addition to the complex skills needed to develop a game, budgets and team sizes for developing a game skyrocketed too [9] [10]. This makes the development process for a computer game difficult than ever. Especially role-playing games need a lot of hands made design work to build one.

To overcome this predicament and fully realize what this medium could offer, developers start to focus on developing tools, engines, and frameworks to help them in developing games [11][12]. This software enables us to create

games faster, better and cheaper. Ability to make games faster and cheaper help game developers hone their skills and improve their craftsmanship. Also, it provides developers with more opportunities to experiment with the role-playing game design formulas and create unique experiences for players.

1.2 Contributions

The main contribution of this work is to create a software development framework to make the development process of a role-playing game faster, easier and reusable. Every module developed as a part of the framework satisfies a certain well known design trope. This study shows how to make those modules communicate with each other with ease and add more modules or remove some of them without creating a problem for the rest of the software.

The proposed method will give developers a great flexibility when they work on their games. What is offered is not the parts of the game development framework we created but the blueprint of it. The study provides an insight on how to build it and connect its parts in such a way that they work seamlessly with each other.

1.3 Structure

In this chapter the motivation and the contribution of our work are presented.

In chapter 2 we try to examine the game genres our framework would be designed to develop games for. We explain how role-playing and adventure games work, their sub-genres and certain design tropes we needed to focus on while we develop our framework.

Chapter 3 is dedicated to the analysis of existing systems used for developing role-playing games and adventure games, what are their strengths and their weak points.

Chapter 4 explains the design of our modular game development framework. This chapter focuses on our design choices and technologies used when we built our development framework.

Chapter 5 explains the details of our novel method, which is a modular game development framework.

Chapter 6 gives detailed information about example games developed with the framework this thesis based on. The user study conducted to evaluate the framework reviewed in this chapter.

The conclusion and possible future work can be found at chapter 7.

CHAPTER 2

BACKGROUND

In this chapter, we will have a discourse on role-playing games and game design philosophies we want to focus on for our game development framework.

2.1 Role-Playing Games

2.1.1 What is a Role-Playing Game?

A role-playing game (will be abbreviated to RPG) is a game in which players assume the roles of different characters in a fictional world while one player assumes to the role of Game Master (GM), prepares a game scenario, simulates the rules and acts all the other non-player characters.

Introduced in 1974 with the game called Dungeons and Dragons (D&D) [13] by gamers who mix their wargaming hobbies with their love of fantasy literature. While D&D was the first tabletop role-playing game, others arose as well as almost at the same time. Every new game offered some different aspects to the players and with every new game RPGs become more diverse. Gaming systems such as GURPS focus on game mechanics to help players better simulate the game they are playing with the rules of the game while other gaming systems such as PtbA offer their players immersion in the game over the complex rules with their relatively light rulesets; and gaming systems such as D&D stays in the middle of these two design methods [14].

As its popularity grew among gamers, people experiment and find new ways to play. Some players prefer tabletop gaming and developed such different rulesets we mentioned earlier, others carry the game away from the tabletop and focus on playing the role of their characters in LARP games in which their players put on costumes and play on as their characters. Eventually, some people carry their hobbies to computers whose processing power will help them to simulate game worlds, rules, and adventures which they can play.

2.1.2 Computer Role-Playing Games

In the 70's some tabletop RPG playing programmers got access to the mainframe computers at their universities and they tried to replicate that gaming experience on those computers. Their efforts led to the first examples of CRPGs and adventure games. Those games which got popularized in those mainframe computers influence future developers to create RPGs, roguelikes and adventure games.

The '80s were the golden age for CRPGs. Some of the developers or players of the early RPGs on university mainframes tried to replicate the experience on their home computers. Many of the legendary CRPGs series started to develop these years. Bards Tale, Ultima, Wizardry and Might&Magic series are some of the early successful examples of CRPGs.

The '90s saw the rise and fall of the genre. As it reached its peak and suddenly fell from grace as 3d graphics technologies of the time improved quite dramatically and both hardware producers and game developers set their minds on other genres like FPS games which is more suited to utilize the power of new 3d graphics hardware, create stunning visuals for its time that captivate gamers and sell more hardware and games.

While the '90s were unfortunate times for the genre some of the best and most influential offerings came in the second half of that decade. Diablo 1 and 2 define what an action RPG should be and inspire a whole lot of copycat games after themselves. While Fallout 1 and 2 were not a financial success, they become cult classic games and also prove that RPGs did not have to be limited in usual medieval fantasy settings with their strange offbeat post-apocalyptic science fiction setting. Bioware and Black Isle Studios, with their famed Infinity Engine, created games such as Baldur's Gate series, Icewind Dale series and Planescape Torment which popularized and revitalized the genre. Square's Final Fantasy 7 proved that there is a market for JRPGs worldwide by becoming one of the best selling games in the genre[15]. Today CRPG genre is a prominent game genre that each year sees new game releases for enthusiastic fans.

In their digitalized form RPGs let the computer be GMs of the game. As the computer handles the simulation of the game world and its rules, players would be free to experience their adventures in the game. Earlier examples of RPGs were all single-player games as their creators saw computers to answer to the problem of finding others to play RPG with. In time there were multiplayer RPGs developed as the technology gives developers a unique chance to create games where all the players could assume the role of their characters and play simultaneously in a living world. Those games called massively multiplayer online role-playing games and in time they too evolved to be their genre. Unfortunately, the MMORPG genre is outside of the boundaries of this work as they are an enormous subject on their own. Today most of the CRPGs are single-player games and this body of work will focus on the development of single-player games as well.

In tabletop RPGs usually, every player has their character to play, they form an adventuring party in which every character offers their abilities to face the ongoing challenges their adventures offer [16]. To emulate this well-known trope of the RPG, single-player CRPGs offer different solutions. In some of them, players control only one character as the main protagonist of the game. Some of them allow players to create their adventuring party and control all of them and some of the games allow players to control the main protagonist only but enable them to recruit NPC characters to their adventuring party during the game.

Another problem with the conversion of the RPGs in digital form is actualizing freedom of choice of the players in tabletop RPGs. When we assign the GM role to the computer, it is stuck with a pre-programmed story that can't be changed on the spot while in the case of a human GM, the story can change [17]. To overcome this predicament CRPGs usually offer multiple solutions for the challenges players face in the game. As Greg Costikyan said "There are generally multiple solutions to problems, and as a result, players feel like they have more freedom" [18] but this design choice consequently makes development harder as each additional choice increases the amount of necessary content exponentially rather than linearly [19].

2.1.3 Western vs. Asian Approach



Figure 2.1: Screenshot from Baldur's Gate. A CRPG developed by Bioware software.

CRPGs become popularized in Japan with the release of the game called Wizardry. After that western CRPG developers continued to evolve their designs;

however, Japanese developers took their formula from Wizardry and deviated from there. This split in design choices created an interesting dichotomy. The North American and European developers worked on with the plan of offering something different from their competitors. This results in making every big RPG series as a different style game than the others. Long-lasting series like Ultima, Wizardry, Might&Magic series and Gold Box games all offer their distinct styles. Other developers either tried to follow a popular series formula or they too created something new, something different to attract attention to themselves.

Japanese role-playing games tend to be developed for game consoles rather than PCs as Japanese and other Eastern markets are mostly game console-based. Because of those games developed for game consoles their control schema's designed to be used with game-pads. Those games have menu-driven turn-based combat. They have mostly linear storylines that focused on pre-generated player character [20] and repetitive combat encounters for leveling up which is called grinding is a common trope that can be found in JRPGs. Not allowing players to create their characters and focusing on linear stories remove player freedom from the games but JRPGs use those constraints to offer proper character development and a narrative structure [20]



Figure 2.2: Screenshot from Phantasy Star IV. A classic JRPG for Mega Drive/-Genesis.

Western RPGs are mostly developed for PCs rather than game consoles. Most

of the time players create their character(s) and/or customize them at their own will. With western RPGs whether the game is fast action based or slow, deep and tactical turn-based game character development is always one of the focus areas of the game. Another definitive thing for western CRPGs is allowing players to make choices in the narrative and make those choices affect the story of the game [16]. Other than this each developer uses a different style with their game, unlike JRPGs which perform in a very similar style to each other. In this aspect, western CRPGs are more diverse than JRPGs but genres are evolving steadily and each of them takes note from each other, so there are western CRPGs with pre-made player characters and there are JPRPGs that break the generic JRPG formula in different areas.

2.1.4 Character Development

Character development is one of the most fundamental parts of RPGs[21]. Playing the role of a character is an essential part of the RPG. To make the player more invested for this most of the RPGs allow them to create their customized characters while some choose to opt this for making player characters more integrated into the games' storyline and let the player take control of a previously generated character. Either way player characters gain experience throughout the game and grow more powerful as they go on their adventures. Starting as a novice character and slowly becoming a mighty hero in the game is a fascinating experience for players.

Characters in RPGs consist of collections of different attributes and traits. These values define the characters' abilities, skills, special talents, and even their personalities. Each game has its way of defining its characters by its own game rules.

Character classes are another thing used by RPGs to define player characters. Each character class defines a playing style for the character to assume. The selection of a character class for player character puts it in a defined role in the game as well. So this will limit and determine which attributes and traits a player character will get and could improve later on the game.

Be it a part of the power fantasy offered by the RPGs or call it the experience and knowledge the player characters accumulate during their journey each RPG presents a way to player characters to improve and advance their abilities throughout the game. Leveling up is one of the oldest tropes in RPGs used for character development. Aptly named experience points awarded player characters for their accomplishments in the game and through their accumulation of those experience points, player characters level up when those experience points exceed the number required for the next level. This leveling up enables players to improve their characters' attributes, traits and gain new traits as well.

There are other character development methods besides leveling up. Such as awarding small incremental bonuses to related attributes as players use their characters' abilities. Some others forgo leveling up completely and let players

use points awarded to them for completing tasks or quests in the game to directly improve their attributes and traits or gain new traits.

Character development is an important choice left to players as through their choices they choose how their playing style is going to be and how they are going to approach the challenges the game embarks upon them. It is the games' and their designers' jobs to present players with character development systems that have enough depth to offer different and equally enjoyable playing styles to players.

2.1.5 Choices and Consequences

Choices and Consequences is a term coined to define a game design device to allow users to experience the story of the game more interactively. Choices and Consequences are about interactive storytelling. It is not simply having branching plots but it is about presenting the player with different options to choose and creating a game world that reacts meaningfully to those choices and showing players that their choices bring their consequences with them, hence the name.

Choices and Consequences work in games where the narrative of the game is based upon its story. Choices and Consequences allow players to manifest as the characters they created in the game world, such as living in that world and taking actions that in the end shape the game world as well.

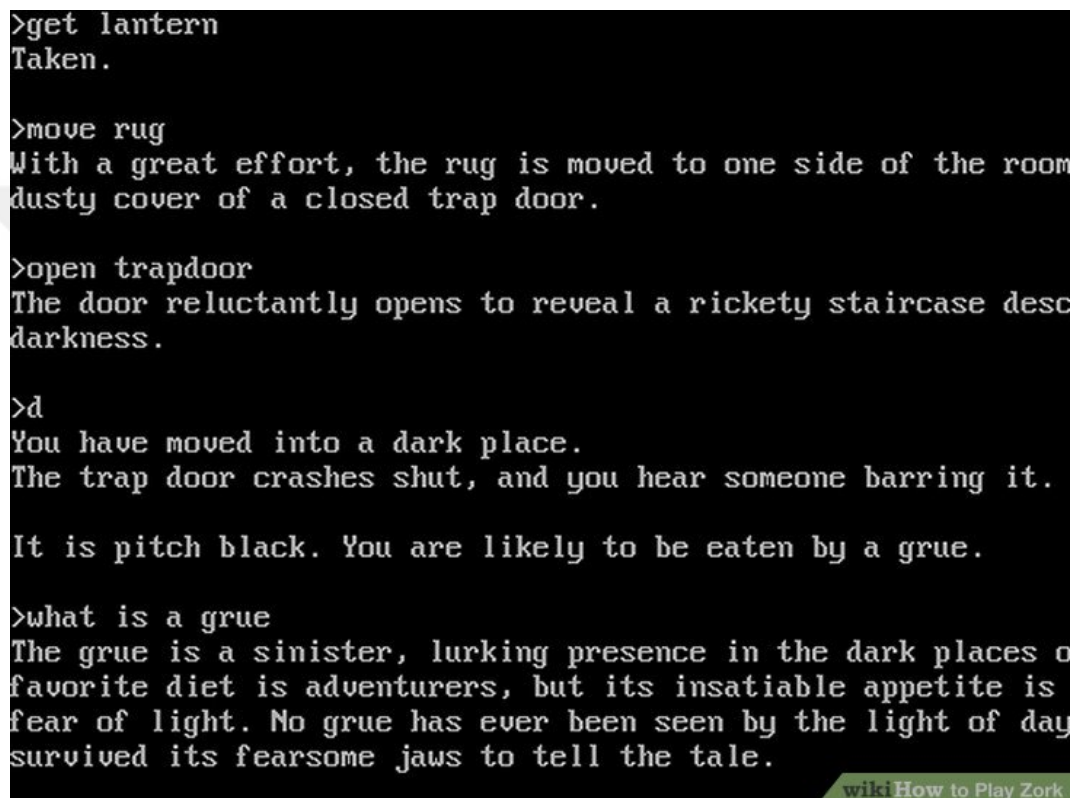
Choices and Consequences are neither about finding the right choice, nor solving a narrative puzzle. It is about having different options to choose and seeing those choices leading to interesting consequences throughout the journey you take during the game.

This is not the be-all-end-all approach to role-playing game design but an important tool to give player agency and a feeling of freedom in their choices in the game. Not all player choices in the game should have important consequences, it is all right to offer conflicts the solutions of which do not create some other consequences for players later in the game. These kinds of choices that don't affect the game afterward are powerful tools as well to treat players with encounters they can play the role they choose in the game without minding about the consequences all the time.

What Choices and Consequences offer are showing players they are in a living game world which would react to their actions throughout the game. This creates a more immersive gameplay experience for players to experience in the game.

2.2 Adventure Games

Adventure games are a video game genre that presents an interactive story to its players through exploration and puzzle-solving challenges[22]. The genre easily borrows elements from other narrative-based media such as films and literature because of its story-based nature and mostly linear gameplay. The first game classified as an adventure game was developed in 1976 and was called Colossal Cave Adventure which is often referred to as Adventure, thus giving the genre its name. Colossal Cave Adventure was influential on role-playing games, roguelikes as well as adventure games.



```
>get lantern
Taken.

>move rug
With a great effort, the rug is moved to one side of the room
dusty cover of a closed trap door.

>open trapdoor
The door reluctantly opens to reveal a rickety staircase desc
darkness.

>d
You have moved into a dark place.
The trap door crashes shut, and you hear someone barring it.

It is pitch black. You are likely to be eaten by a grue.

>what is a grue
The grue is a sinister, lurking presence in the dark places o
favorite diet is adventurers, but its insatiable appetite is
fear of light. No grue has ever been seen by the light of day
survived its fearsome jaws to tell the tale.
```

wikiHow to Play Zork

Figure 2.3: Screenshot from Zork. One of the earliest interactive fiction computer games.

Early adventure games were text-based games that used a text parser to handle player input. They had a limited number of verbs to offer players to interact with the game world. This dictionary of verbs gradually grew with every new game. As the technology improved some text adventures got graphical elements such as the illustrations of scenes player travels and the animations as they act upon the world. Eventually, those verbs turned into icons and the point and click adventure sub-genre was born.

With each new game and improving technology adventure game genre grew and evolved. CD-Rom and 3D graphics created new ways for adventure



Figure 2.4: Screenshot from Maniac Mansion. A classic point&click adventure game from Lucas Arts.

games to advance such as pre-rendered or real-time 3D scenes or full-motion video graphics.

The late '90s saw a decline in the adventure game genre. Improving 3d accelerated graphics card technologies pushed for faster playing 3D games and classic adventure games fell out of favor. The genre had a revival in recent years with the work of the independent game developers. Some revisited the original point&click formula and the others experimented with the genre to create something fresh.

Even though there are exceptions most adventure games present a linear storyline to the player [18]. As the protagonist of the story, players have to solve the puzzles they encounter throughout the game. Especially the point&click adventure games, as they were the most popular subcategory of this genre, have this mentality in game design.

In the context of this thesis, we try to focus on the narrative aspect of the games. For this reason, instead of point&click adventure games, we focus on more narrative-driven sub-genres of adventure games, such as Interactive Fiction games.

2.2.1 Interactive Fiction

Early adventure games consisted entirely of text, mostly because of technological limitations. As the technology improved graphical interfaces devel-

oped an adventure game genre flourished. Nevertheless, text adventures were not forgotten. This sub-genre continued its life with both improvements of the original formula and invention of new formulas.

In the early text adventures player interacts with the game by entering text commands. These commands interpreted by a text parser. Early text parsers were simple verb-noun parsers which handled basic instructions like "open door", "go north" or "get key". Later text adventures and modern interactive fiction games use a more advanced parser that can handle more complex sentences. As the text adventure games left the stage to the graphical adventure games, this medium continued to serve those who want to write interactive fiction (IF). IF genre tends to be more focused on the narrative of the game than the usual puzzle-solving aspect of the adventure games. Even when they include puzzles it would be a part of the narrative rather than a specific gameplay element of the game apart from the narrative of the game.

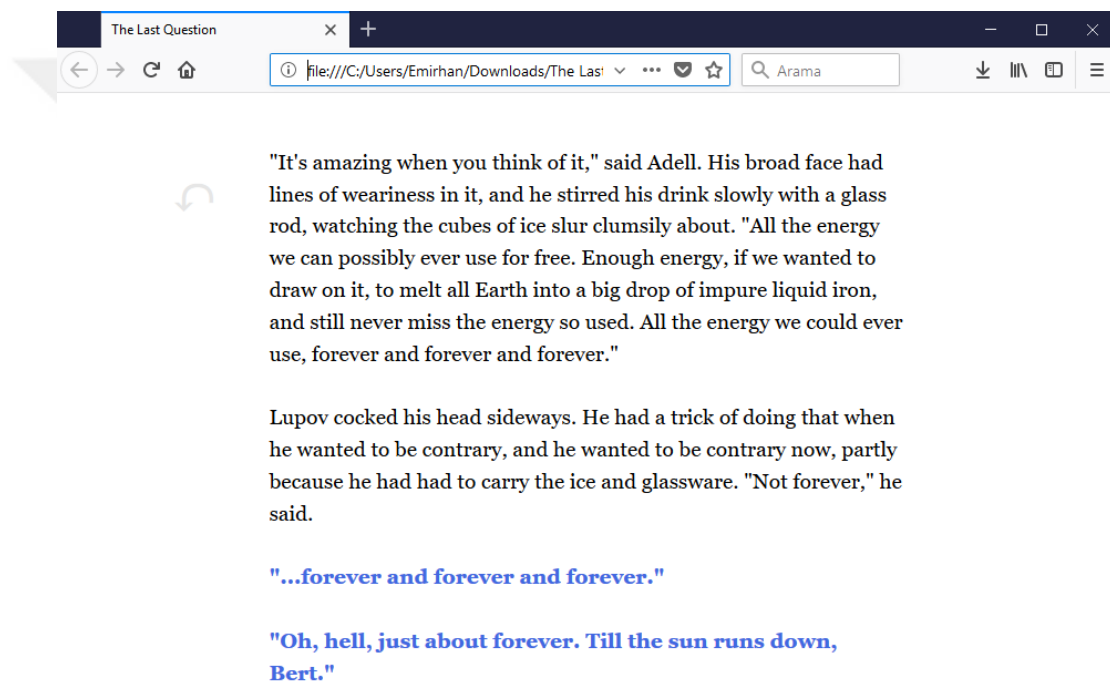


Figure 2.5: Screenshot from an interactive fiction game developed with Twine.

Another approach to the genre is to emulate the Choose Your Own Adventure books in computerized form. Developers got rid of the text parser and gave players options to choose how to continue to the story. This approach produced games that are less focused on exploration but more focused on the flow of the story [23].

Some developers utilize the hypertext format in their games and embed the player choices as short phrases in the main text of the game story. Actions players could take are shown by displaying in a different style, most usually

in bold text and in a different color. While every emboldened phrase does some action it doesn't have to lead the story in a different direction. The flexibility of the hypertext format offers developers to do more and just showing more text. If used well this little change could enable developers to create a feeling of exploration in their games.



CHAPTER 3

ANALYSIS OF EXISTING SYSTEMS

In this chapter, we will discuss other game development tools for Adventure and Role-playing Games. We will talk about their strengths and their shortcomings and why we don't use them but create something new.

3.1 RPG Maker

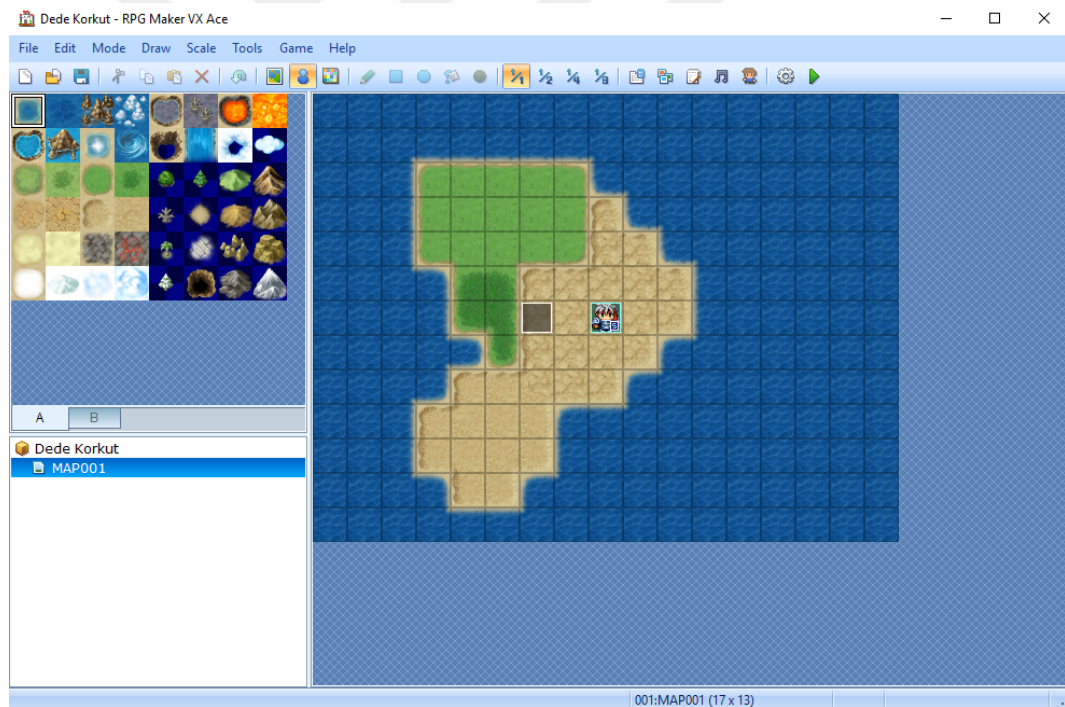


Figure 3.1: Screenshot from RPG Maker VX Ace.

RPG Maker is a game development tool specialized in developing Japanese Role-playing Games (JRPG). It has been first developed by a Japanese group called ASCII, then succeeded by Enterbrain. It is constantly being updated and new versions are being released. While it has been released in Japan first,

with later versions released worldwide it has gained popularity all around the world.

The latest version of the series is called RPG Maker MV and released on October 23, 2015. It works on both Windows and Mac OS X and lets you build your games for Windows, Mac OS X, Android, iOS and Html5. With this version RPG Maker starts to use Javascript as their scripting language instead of Ruby [24].

RPG Maker provides a map editor, a character generator, a built-in battle system and a GUI driven database for game data such as items, characters, skills. Also, RPG Maker has a great online community that users could find support when needed. Because of its popular status, many game assets that are compatible or directly developed for RPG Maker could be found online either commercially or free of charge.

RPG Maker has been used to create countless games ever since the release of its first version. Both for free and commercial projects it is one of the most popular tools used by RPG developers. Some of the notable games created with the series include the followings:

- Aldorlea Games - Millennium and Laxius Force series
- Blossomsoft - Eternal Eden
- Kikiyama - Yume Nikki
- Freebird Games - To the Moon
- Danny Ledonne - Super Columbine Massacre RPG!
- Sherman3D - Alpha Kimori
- Laura Shigihara - Rakuen
- Team GrisGris - Corpse Party
- Dingaling Productions - Lisa: The Painful

Even with all these great features RPG Maker couldn't fit our needs because it was specifically tailored to a certain kind of sub-genre of RPGs. If you want to develop classical JRPGs is trying to emulate the style of the 16-bit game console era, RPG Maker is the best tool you can find. But if you want to deviate from the formula, add something new or different while you still can, it will require much more work that eliminates the positive aspects of using a tool in the first place.

3.2 Twine

Twine is a tool developed by Chris Klimas for creating interactive fiction games. It is open-source software available for Mac OS X, Windows and Linux [25].

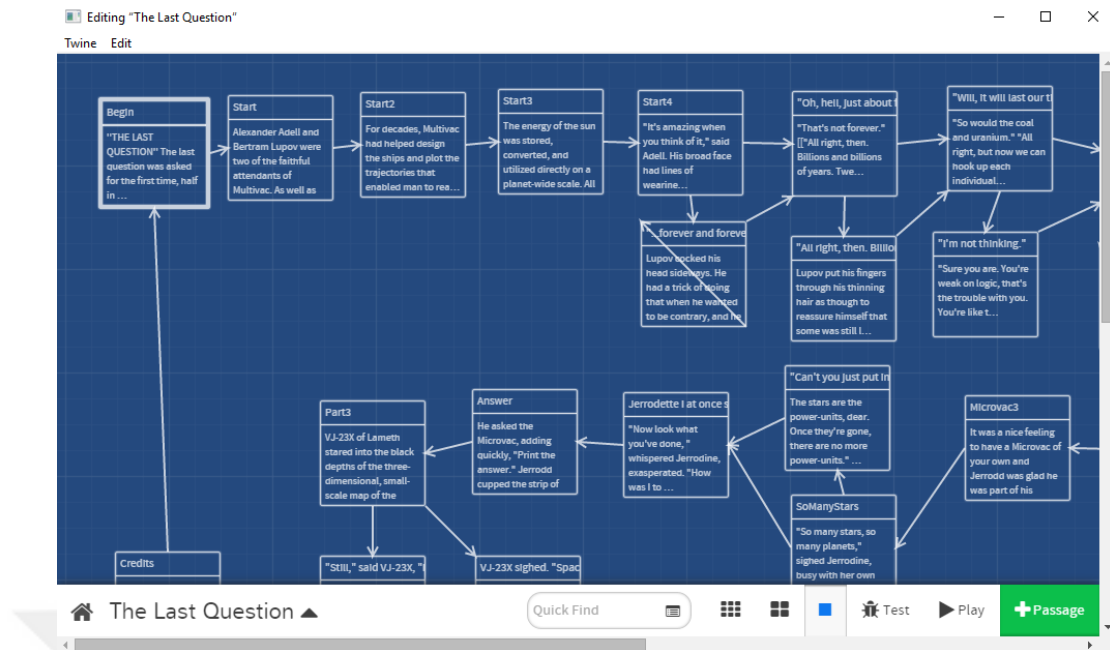


Figure 3.2: Screenshot from Twine.

It is a popular tool for developing interactive fiction games. Twine provides a node-based visual editor to develop your games, uses a simple markup language and exports the games being made in HTML format.

Although it is a valuable tool to develop narrative-driven games Twine's simplistic nature becomes its shortfall as you try to develop more complex games, workload increases and organization and code become messy. While it is quite possible to have a few player stats and dialogue options that could change related to those stats, making a full-blown RPG is too much for its abilities.

3.3 Unreal Engine 4 and Blueprint

Another notable tool for game development in Unreal Engine. It is a AAA game engine developed by Epic Games. It is available for free, and its source code is fully available as well. It is written in the C++ programming language and allows developers to use C++ to make their games. Also, it offers a visual scripting system called Blueprint.

Blueprint system uses a node-based development environment for those who do not want to use C++ (e.g., game designers, artists). Blueprint system offers almost all the functionality you'll get with C++. Developers could develop games entirely with Blueprint or use it partially with C++ classes as well.

Unreal Engine and its greatest competitor Unity3D go toe to toe in terms of features and capabilities. Unreal Engine has a slight advantage of being an

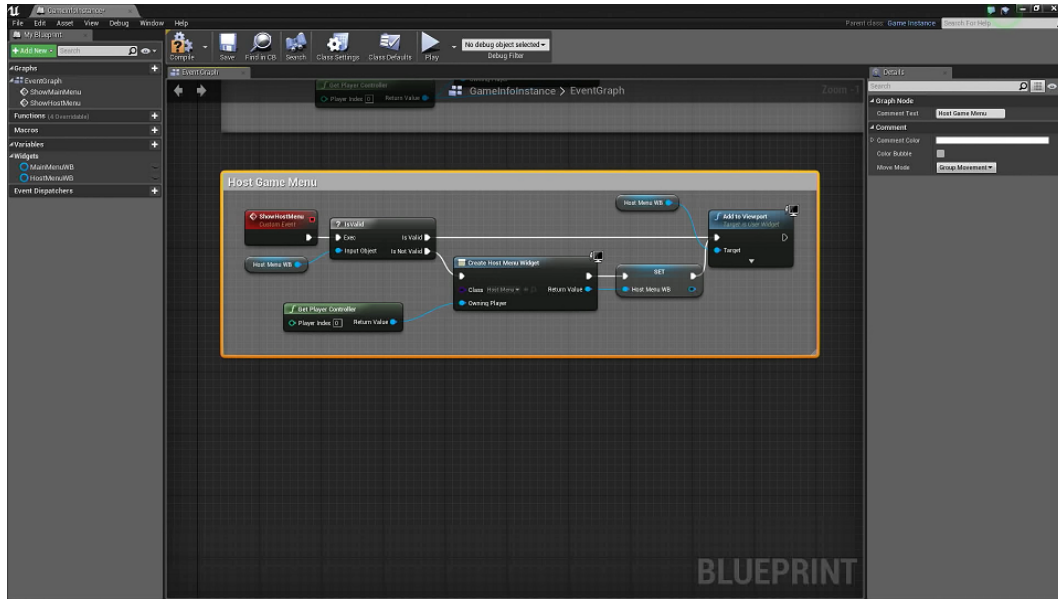


Figure 3.3: Screenshot from Unreal Blueprint.

industry-standard AAA game engine for years, though this advantage brings its issues with itself too. This makes Unreal Engine bigger, more complex and therefore harder to handle than Unity3D.

Using C# for game development far easier than using C++. Even though there is Blueprint visual scripting system to compensate for the complexities of using C++, it is not feasible for most cases. While Blueprint is adequate for developing small games, it is still limited compared to using C++[26]. Also debugging and maintaining a codebase in C# is easier than in a visual scripting system.

Graphical improvements that Unreal Engine offers over Unity3D, although considerable still is mostly negligible if one does not have a good technical artist to take advantage of those. Unreal Engine is a more powerful, professional, and coincidentally more complex game engine than Unity[27].

3.4 Other Solutions

While there are many more tools to help us develop RPGs or narrative-driven games they all fall into the same pros and cons like the ones above. There are some other tools for developing RPGs mainly for either JRPGs or Action RPGs as those sub-genres follow the same gameplay tropes more firmly. Likewise, there are lots of great tools to develop IF games but as they focus on the narrative side and utilize the text-based nature of those games they lack in terms of multimedia functions.

CHAPTER 4

DESIGN

In this chapter, we will explain our proposed method for building a modular game development framework for making RPGs.

4.1 Module System

Our design for the framework is based on different modules working separately but communicating with each other to pass the data. Each module is designed to be independent of each other, yet easily communicable with each other to pass data and sync up. Any module could be removed or added without creating coherency problems. This design decision allows the framework to be changed radically according to the wishes of the game developers using it.

In this version of the framework, we focus on developing modules to give developers essential game mechanics to create RPGs. They have been carefully chosen to emulate the aspects of a tabletop RPG on a computer.

4.2 Model View Controller

Model-view-controller (MVC) is an architectural software pattern commonly used for developing user interfaces, that separates an application into three interrelated components. These components are named as Model, View, and Controller. Its name is created by the union of its parts.

The model component is responsible for managing the data, logic, and rules of the application. It is the component where you define your data types, store them, alter them as requested by the Controller component or pass them to View component to be used by.

The view component is in charge of the representation of data supplied from the Model component. It defines how you visualize the data and how to present them to users.

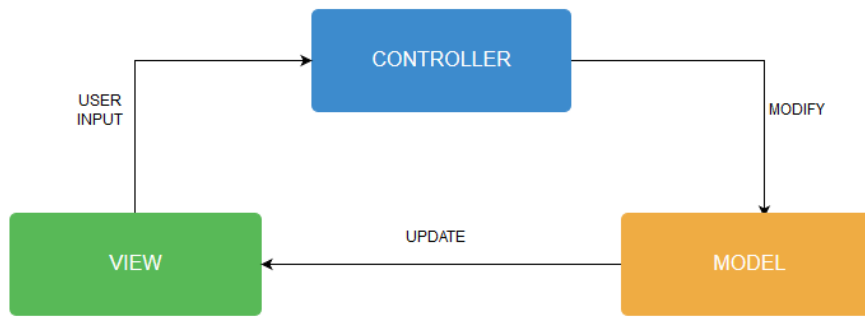


Figure 4.1: Diagram of the MVC pattern.

The controller component allows users to interact with the application. It accepts user inputs and converts those to commands for model or view components. Any modification of the applications state is done by the controller component.

MVC allows developers to be able to work in parallel on different components of the application without causing conflict by means of the way it decouples those components of the application. It also helps developers to implement changes in the application quickly and easily. As parts of the application completely separated from each other, developers can make changes in relevant components without worrying about the rest of the application.

As the components are independent of each other, they enable developers to be able to reuse any of the components quickly and easily in other applications.

We use the MVC pattern for all the modules in our framework. Every module is divided into components as mentioned above. In addition to that, we have implemented interfaces for each module to communicate with each other. Still, this communication mostly took place between the model components of each module, so no other module could modify another model data without permission. Any information change is passed along model components and regulated by them.

Following the MVC pattern too strictly sometimes might cause unnecessary levels of abstraction. In some cases, quick fixes are suggested instead of more abstraction in the code [28]. We follow that advice as well. All the modules built MVC separation in mind but this separation, especially in games, is very hard to maintain and we didn't shy away using quick fixes in small cases instead of redundant abstraction in our code.

4.2.1 Entity Component System

Entity Component System (ECS) is an architectural pattern that values the composition over inheritance principle. In ECS, Entities are empty shells for objects without any previously defined behaviors. Components are properties and behaviors of those entities which assigned dynamically. Systems are processes that run continuously on entities and their components [29].

ECS and MVC patterns function quite similarly. Although ECS requires a different approach from standard Object-Oriented Programming [30]. This approach mostly coupled with the concept called Data-Oriented Design and particularly successful with memory usage optimization.

Unfortunately, this demands an entirely new way of thinking for the developers. For using ECS properly developers would need to understand this new paradigm. The whole codebase and the data must be designed around this paradigm to effectively use ECS. [31]

For the games that are targeted by this framework, performance is not a problem most of the time. So, MVC has been chosen instead of ECS because it can offer similar solutions such as separation of the code from the data, and code reusability without creating a technical debt for the development team.

4.3 Technologies Used

Our goal was to create a flexible and easy to use game development framework. We also want to develop it in an acceptable time frame. To achieve this, we searched for free and whenever possible open source solutions to build our framework.

4.3.1 Unity3D

We chose Unity3D to be the backbone of our framework. Unity3D is one of the most popular game development tools used by game developers today. It uses Mono, which is a cross-platform, open-source .NET implementation for scripting and offers state of art 3D game engine as well as a 2D game engine, physics engine, audio engine, network engine and easy to use GUI to implement in our games in one package. [32]

It is available for Windows and Mac computers and can make builds of your games with only one click for many different platforms. It provides various tools for creators to rapid editing and iterative development cycles. Unity Editor works for both 2D and 3D game development, supports specific functionalities for both of them. Unity3D supports most of the popular file formats out there to be used in the development pipeline without additional work. We can add the files we want to use in our projects just by drag&drop them into

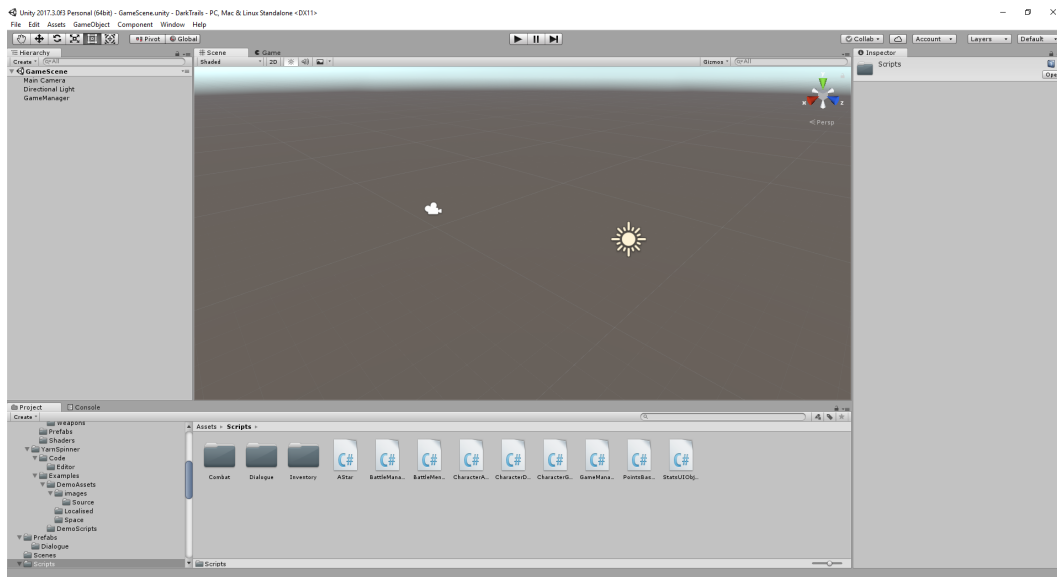


Figure 4.2: Screenshot from Unity3D.

the Unity Editor. Another feature that makes Unity one of the most popular game engines is its rich community support on the Internet.

Unity has a feature called Prefab System. Prefabs are one of the most useful elements in Unity for us. In Unity, objects in the scene are called GameObjects which are composed of several components [33]. Prefabs are GameObject templates whose components, property values and all of its child GameObjects arranged and can be used to create instances of the same object in the scene. Changing a prefab automatically changes all the instances in the scene as well. The modules of the framework are built using those prefabs as the basis.

4.3.2 Yarn and Yarn Spinner

Yarn is a text format that is designed to create interactive dialogue for games easily. Yarn is heavily inspired by and based on the Twine, another very popular interactive fiction game development software. Yarn has got extremely minimal syntax which is very easy to learn. Yarn is a reliable and effective tool that is already used in a number of independent computer games, including Night In The Woods, Knights and Bikes and Far from Noise.

Yarn Spinner is an interpreter for the Yarn, written in C#. It is designed to be simple to integrate into Unity. It works readily out of the box and enables developers to easily modify it for their needs. Yarn Spinner used in the dialogue module of the framework.

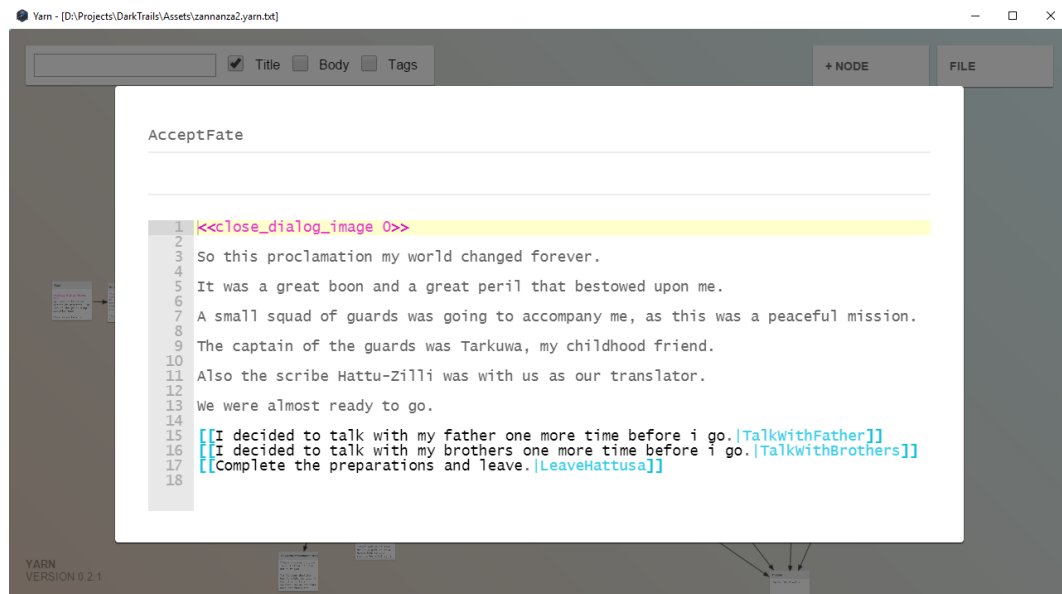


Figure 4.3: Screenshot from Yarn.

4.3.3 Tiled

Tiled is a general-purpose tile map editor that is free and easy to use. Tiled is a flexible 2D level editor. It is capable of creating maps of any size without any restrictions on tile size, or the number of layers or tiles that can be used. Travel modules map files created by using Tiled.

Maps, layers, tiles, and objects can all have custom properties assigned by the user. The custom properties let developers add necessary information in map files needed by the framework. Interactivity in the maps defined with the information embedded in them as custom properties.

4.4 Architecture

4.4.1 Game Manager

Every game, even the most basic ones have a game loop. This is such an integral part of every game. To satisfy this need we created a manager class named Game Manager. We designed this manager class to control the game flow, maintain and manage all the game modules and keep them operational. This manager is not meant to run the main game loop but handle the transition between modules and let them execute their game loops without creating conflict between each other.

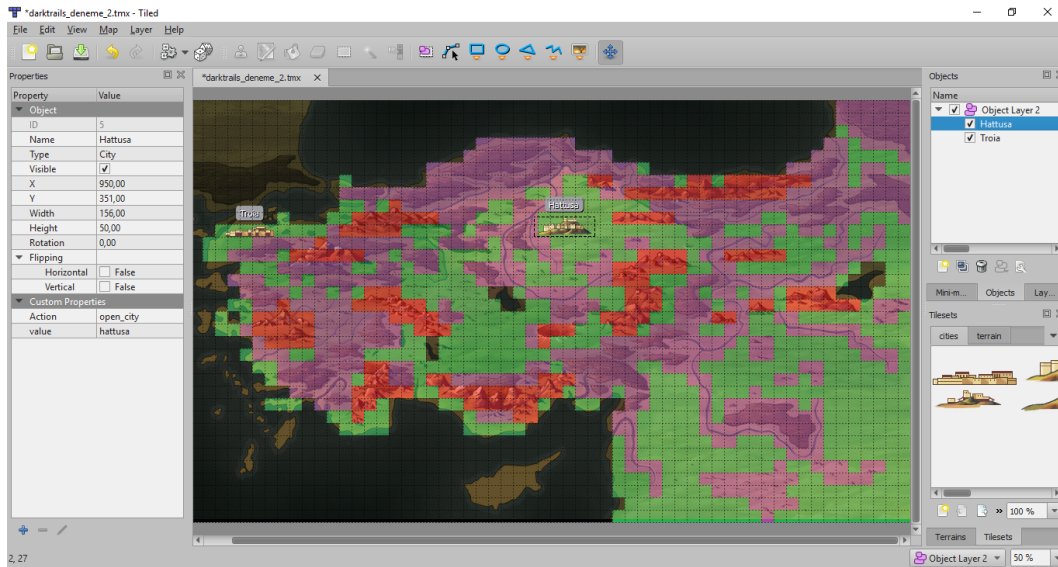


Figure 4.4: Screenshot from Tiled.

4.4.2 Modules

For our framework, we created some modules beforehand. We choose them to handle the most frequently used features of an RPG. The first module we created was the character system module that helped us to keep track of the characters in the game, both player characters, and non-playable characters. The second module created was the combat module. Fighting is one of the most used features of a game. Most games feature some sort of combat in themselves, so it was an obvious choice. The combat module features a turn-based combat system for the game. It will also have special encounters and role-playing moments to give players immersive game experience. The third module was the dialogue module. It is for players to talk and engage with other characters in the game. The fourth module was the map Module. It was designed for players to travel around the game world, visit places and have special encounters. The last module we chose to build was the Inventory module to keep track of the items in the game world. It is also responsible for equipping the characters in the game with those items and keep a list of the items carried by those characters.

4.4.2.1 Character System

As the backbone of the rule system of the game, the character system module was an important one to build early. Success or failure of action all depends on the characters taking that action and the characters that acted on. Without any character data, we didn't have any meaningful way to interpret an action's result.

In our character system, each character is defined by their different attributes

that are called Stats and Skills. Stats are more about that character's inner characteristic abilities, physical and mental abilities, abilities they have been born with. We divide those Stats into two different groups, which are named as Primary and Secondary Stats.

Primary stats are base attributes for characters. They are the first things players have to decide on when creating a new character. They affect both Secondary Stats and Skills. What they do is define the base physical and mental capabilities of the characters.

Secondary Stats are mostly derived from Primary Stats. They define the inner characteristics and abilities of the character. Despite the name, these attributes are more direct effects on gameplay than the Primary Stats as they represent the character's health, speed, morale, etc.

Skills are attributes that represent the abilities of the characters on subjects that can be learned and developed throughout the game. Each Skill has one major and one minor Primary Stat that affects them. Temporary or permanent changes to these Stats causes similar changes in the Skills related to them. Each skill is associated with an action the characters in the game can do. Skills cover various subjects from physical ones like fighting to social ones like trading. Their diversity helps game designers to offer varied solutions to the challenges they create for the players.

4.4.2.2 Combat System

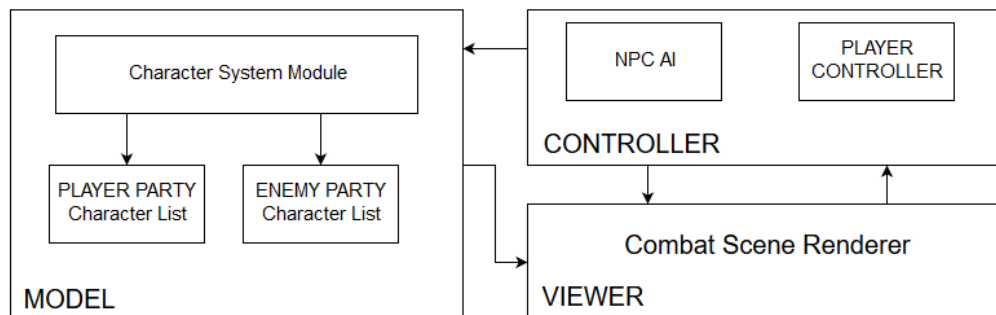


Figure 4.5: Diagram of the Combat System Module.

Combat is one of the most frequently used features in computer games. It is easy to offer a variety of challenges to players in the form of combat. They can be tutorial encounters to teach some aspects of the game, they can be filler encounters to give player option to hone their combat skills before going to face the bigger enemies, they can be scripted encounters as the part of important milestones in the story of the game, player will face an important enemy as the last challenge of the game or at least that chapter of the game or they

can be random encounters to give the player feeling of being inside a living, breathing world.

Turn-based combat systems are the most used ones in the RPGs. JRPGs use menu oriented turn-based combat systems in which players choose an action from a menu for their characters to act upon. Western RPGs mostly use a tile grid where characters wander around and fight each other. This is a very similar approach most tabletop RPGs handle combat as well.

We take the same approach when we design our combat system. It is a turn-based combat system based on a tile grid. Characters take turns according to their initiative scores and use their action points to move and take action when it is their turn.

4.4.2.3 Dialogue System

Our dialogue system uses a slightly modified version of the Yarn Spinner developed by Secret Lab. Modifications we made are to create an interface for our dialogue system and character system to communicate and share data. Because of this, we can offer branching dialogues according to the abilities of the player characters and skill checks inside our dialogue system.

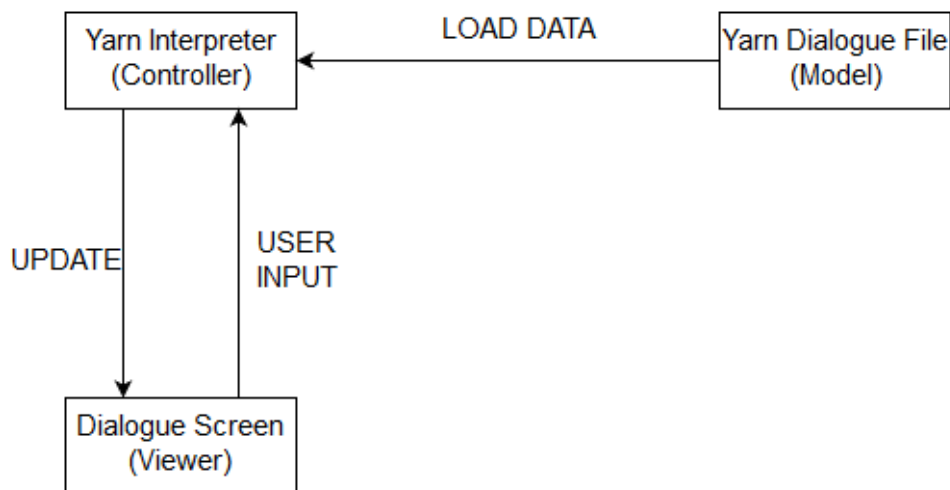


Figure 4.6: Diagram of the Dialogue System Module.

Dialogue System is the one that drives the narrative of the game and handles non-fighting related challenges. Yarn has a node-based visual dialogue editor. It helps us to effortlessly create dialogue for characters and encounters.

4.4.2.4 Travel System

To allow the player to travel around in the game world, we designed a simple travel system. It is influenced by tabletop RPGs. In those games, GMs prepare maps to their players to show them the game world and help them navigate themselves during their adventure.

With that concept in mind, we designed a simple solution. Our travel system will show an illustration of the region player traveling as a background image. Walkable and unwalkable spaces will be marked beforehand. The player will interact with several points of interest distributed around the map. Some of those points of interest will be visible from the start, some of them would be invisible and the player has to discover them before to see them on the map. Each of these points of interest will have an action defined to them. Those actions could be opening a new map for the player to travel around, opening a dialogue system to begin an encounter or opening the combat system to start a skirmish. Also while traveling around the map player could have random encounters not related to any of those points of interest.

4.4.2.5 Item/Inventory System

Item System is there to create a data model for the items that will be used in the game. Some items are personal items for characters such as weapons and armors they wear and wield. Others are consumables such as food or healing potions. Some are there to use in a trade or crafting and some unique ones are there for specific quests. This system is responsible for its modeling, categorization, and storage.

Also, it provides solutions for the occasions these items are used such as inventory window for dressing up characters, the trading window for selling and buying items, stocking up your party's resources.



CHAPTER 5

IMPLEMENTATION

In this chapter, we use UML diagrams to explain the implementation of our systems. We aim to present a greater picture first then focus on smaller parts to comment on how all of them work together.

The curious reader may consult Appendix A for detailed UML diagrams and Appendix B implementation details including code samples.

5.1 Game Manager and Module System

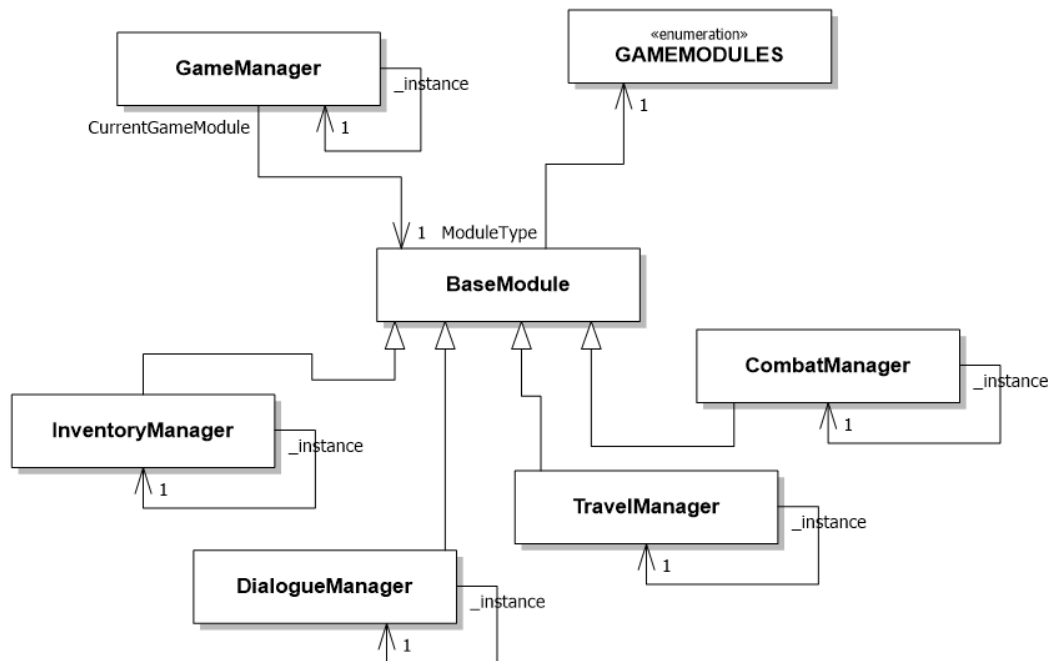


Figure 5.1: UML Diagram of all the System Modules.

GameManager class is where the game begins. It starts the game, loads all the game data, initializes all the modules and handles which module is activated

and which one deactivated. It utilizes the singleton pattern. This way we ensure that there is only one instance of GameManager class created during the execution of the game. Also, it will give us a global access point for the GameManager class to easily reach its functionality.

Each game module has a manager class. Each of these classes is using the singleton pattern as well. At any time only one game module would be active, so this will help us to ensure there is only one instance of every module.

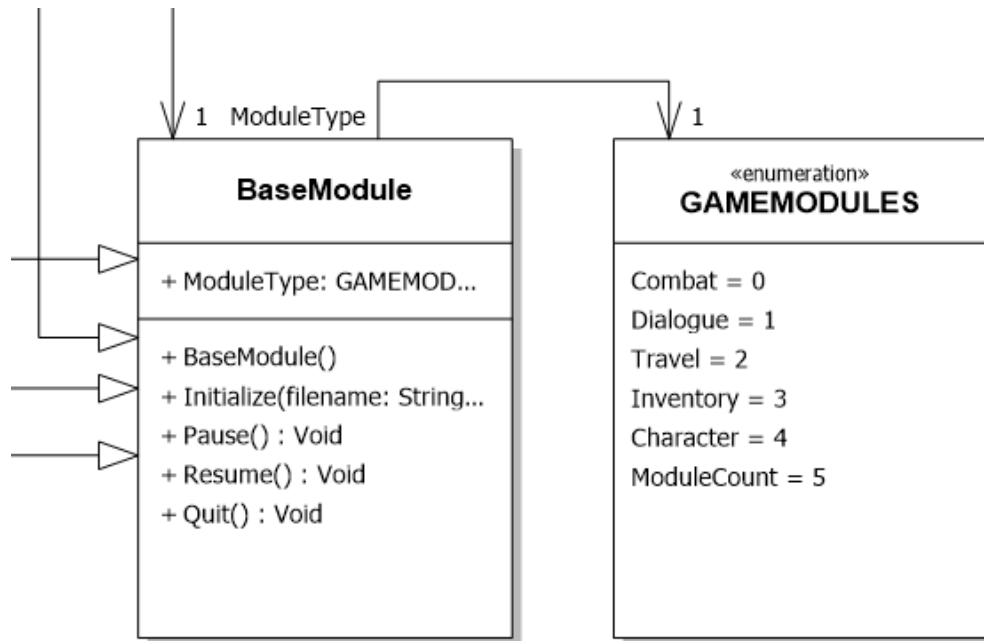


Figure 5.2: UML Diagram of the Base Module Class.

The BaseModule class is implemented as a base class that other Module's manager classes would be derived from. The basic functionality we wanted to be shared by all our modules will be implemented here. All the member functions are defined as virtual functions so each class derived from it has to override and implement their versions of them.

The Initialize method called when a module first loaded and added to the modules list. It runs only once. The Initialize method takes a string parameter to denote the related XML file for the module.

As stated before at any time only one module is active and working. The Pause and Resume methods used when the active module is switching with some other module. The active module calls its pause method to safely stop its inner workings. After the change new active module calls its resume method to start working from the point when it was last paused. Quit method is called when the player exits the game and the application is going to be closed. These methods are implemented to enable the module to stop, continue and quit safely and seamlessly.

5.2 Combat Module

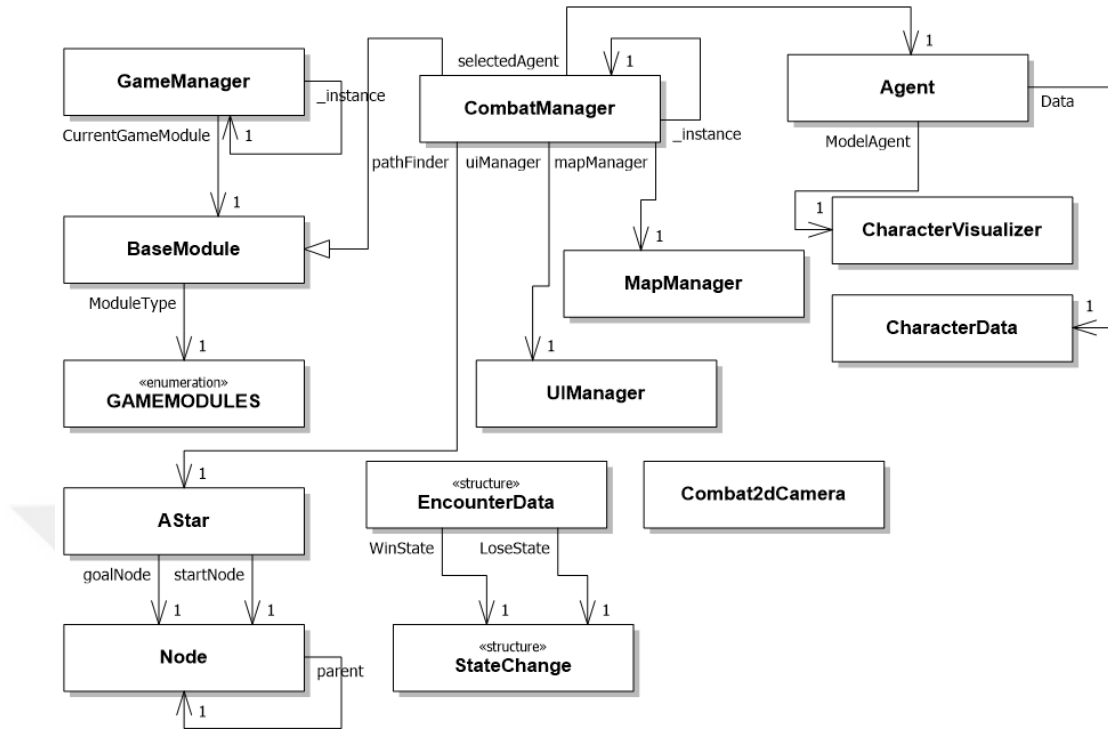


Figure 5.3: UML Diagram of the Combat Module.

The **CombatManager** class is the one prepares the encounter and oversees the main game loop. It uses two other manager classes to work. The **MapManager** class is a manager for the area where the combat will take place. The **UIManager** class is to both update the UI and execute UI commands.

The **Agent** class is used for every character in the combat. It is the main controller class for both AI and player characters. It utilizes a **Data** class for the character data model and **Visualizer** class to handle how to render the character graphic on the screen.

CharacterData class only stores data from **Character System Module** about the character referred at the **Agent** class. The **Agent** class only gets data from it but doesn't make any changes to this class.

CharacterVisualizer class works side by side with the **Agent** class. When **Agent** class calls **MovePath** method this method will call the **Run** method from **CharacterVisualizer** class as well. The agent's position, the path it will pass and its moving speed are all decided by **Agent** class. The **run** method from **CharacterVisualizer** class will decide how the agent sprite will look like while traversing the path decided by **Agent** class. This is the same for other actions of the **Agent**. The agent class calls the **AttackTo** method. It makes calculations using the data it gets from the **CharacterData** class. The **AttackTo**

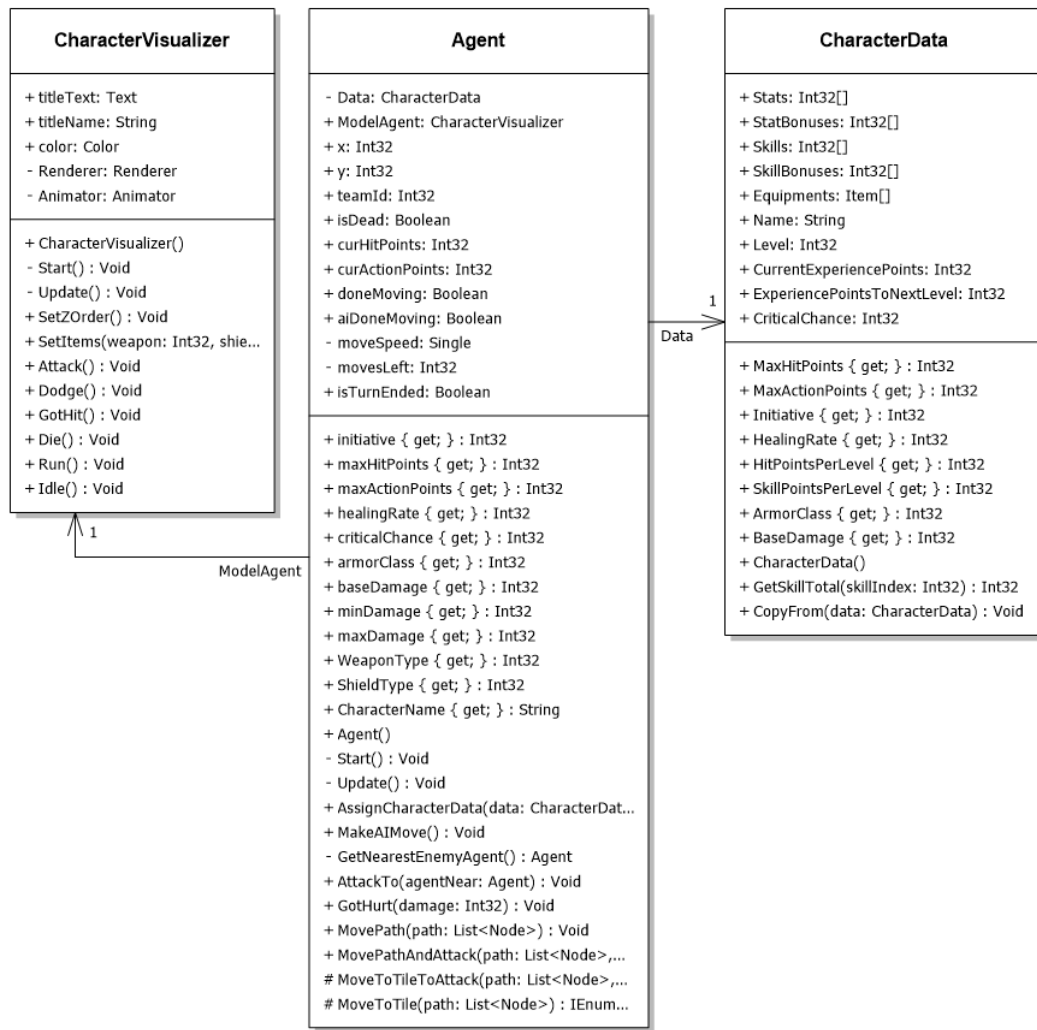


Figure 5.4: UML Diagrams of the Agent, CharacterData and Charactervisualizer Classes.

method calls the Attack method from the CharacterVisualizer class for the attacking agent and GetHurt, Dodge or Shield methods for the defensive agent depending on if the attack is successful or not.

Combat in the combat module takes place in a grid map. MapManager class oversees all the action over this grid map. Astar class which is an implementation of A* path-finding algorithm handles path-finding in this grid map.

5.2.1 AStar Class

A* search is the most widely known form of best-first search algorithm known [34]. It's a flexible algorithm that can be used in a wide range of contexts. In our framework, two different modules use A* search for path-finding.

Path-finding is the process of finding the path between multiple points that are called nodes. A* search algorithm evaluates each node's cost to travel by dividing it into two separate functions. The first function $g(n)$, calculates the cost to reach the node from the start and the second function $h(n)$, calculates the cost to get to the goal from that node. As the algorithm starts from beginning to evaluate when it reaches a node the algorithm knows how much it costs to get there already. The estimated cost to reach the goal is still unknown and to find that the algorithm uses a heuristic function $h(n)$ that satisfies certain conditions of the problem.

5.3 Travel Module

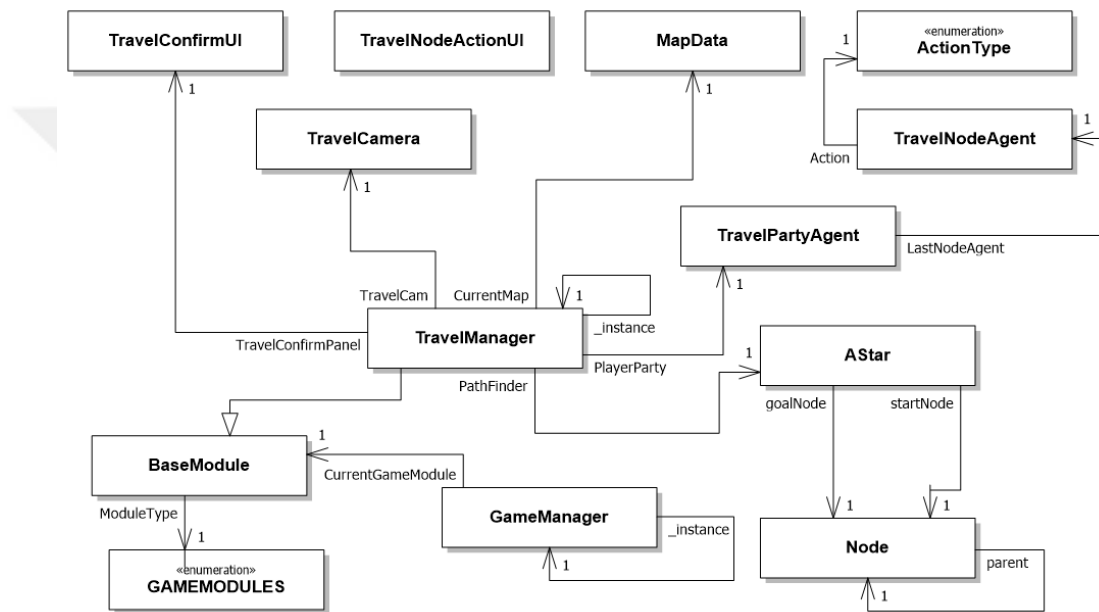


Figure 5.5: UML Diagram of the Travel Module.

`TravelManager` has the list of all the maps travel module could render. Each map is stored in a dictionary with a map name string used as a key. With this key, we could load any map we want by using the `LoadMap` method.

Travel module's maps are very big image files with various interactable Node entities but it still employs an invisible grid map to both mark inaccessible places and also to implement different terrain types that have varied travel times to overpass.

An action is triggered when the player party object comes in contact with any node object which contains the `TravelNodeAgent` class. This action is resolved by the `OnTriggerEnter2D` method from the `TravelNodeAgent` class. `OnTriggerEnter2D` method comes from inheriting the `MonoBehaviour` class. It is connected to the Unity's 2D physics engine. This method will conclude

what will happen according to the data from the TravelPartyAgent and TravelNodeAgent classes collided and call the related methods from the TravelManager class.

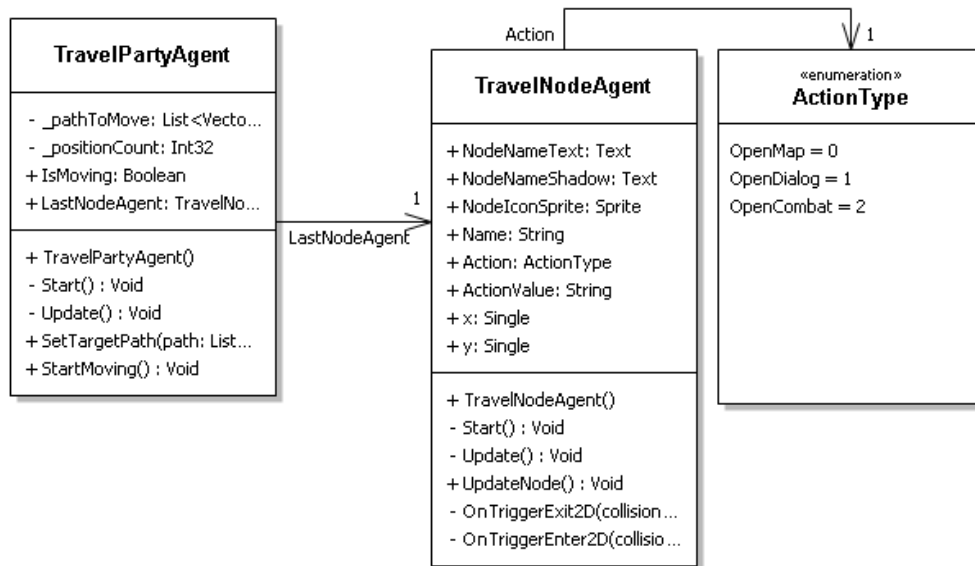


Figure 5.6: UML Diagrams of the Travel Node Agent and Travel Party Agent Classes.

5.4 Dialogue Module

DialogueManager is the core of the Dialogue module. Its main task is to start the specified dialogue node and handle the commands implemented in Yarn to communicate with the Unity.

DialogueVariableManager is the class needed for Yarn to work in Unity. Its task is to pass any variable used in the yarn scripts from the game data model to Yarn. Besides that, any variable from other modules from the game is passed to the Yarn through this class.

The job of the DialogueUI class is to manage the module's UI, update it and pass the user input to Yarn to keep going. Its visual design is divided into three parts. The first and main part is the panel that shows the text directed to the player. The second part is for the panel that keeps conversation options available to the player. The last part is for the panel to show an associated image with the text if available.

Yarn has the ability to let developers call their C# methods inside of the Yarn dialogue files. It is done with Yarn commands. Any number of Yarn com-

mand can be added. This allows developers to increase the dialogue modules capabilities greatly. For now only Yarn commands added are OpenTravel, OpenCombat, SetDialogueImage, CloseDialogueImage and RollDice.

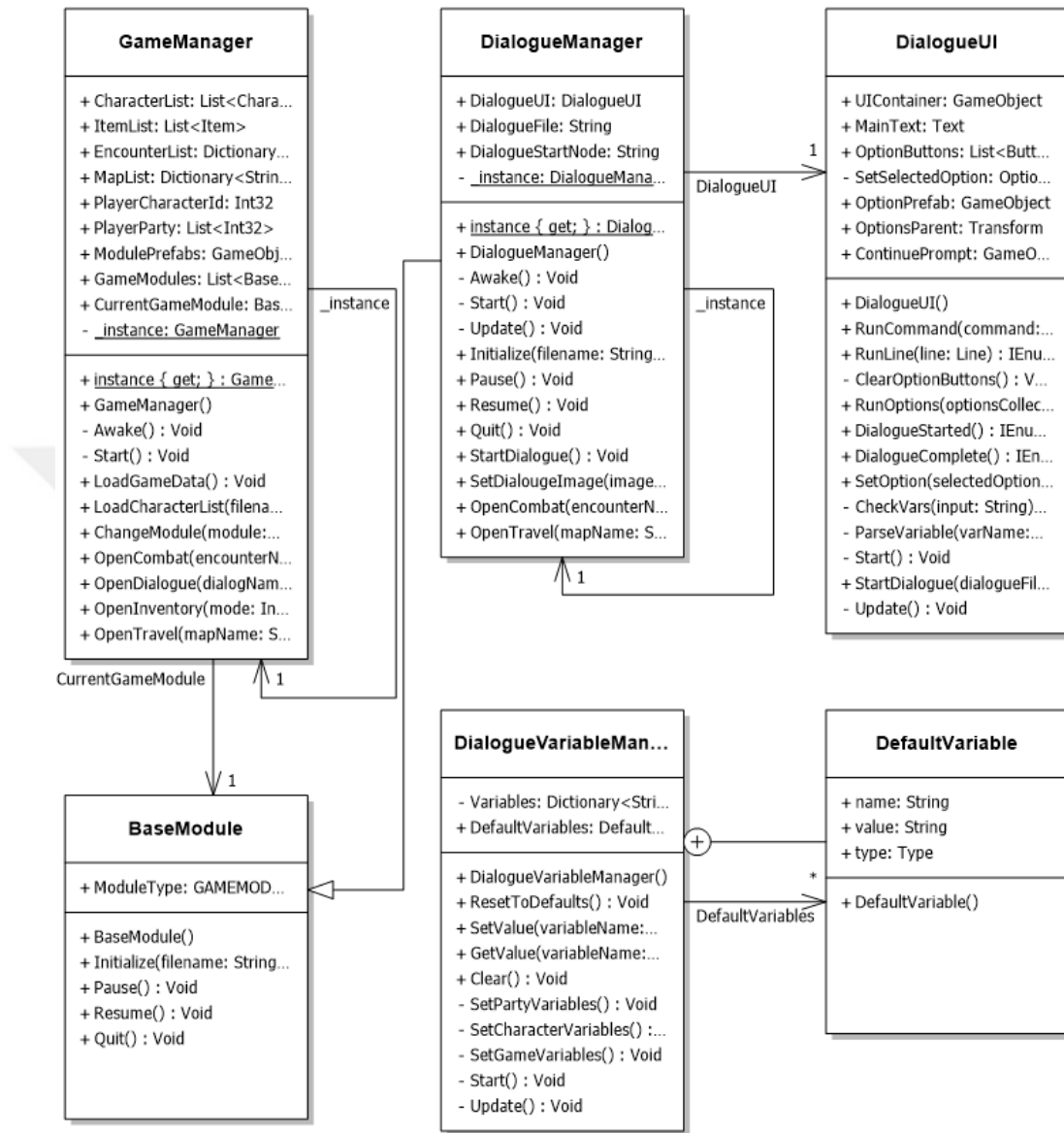


Figure 5.7: UML Diagram of the Dialogue Module.

OpenTravel and OpenCombat commands take one parameter. They start their respective modules and use the parameter given as the value for the map file or the encounter name for opening. SetDialogueImage takes one parameter and changes the image file shown in the dialogue module. CloseDialogueImage closes dialogue image shown. Lastly, RollDice is for generating a random number and it takes a parameter to specify the maximum value for the random number. The random number generated is stored in \$roll_result

variable in Yarn.

5.5 Inventory Module

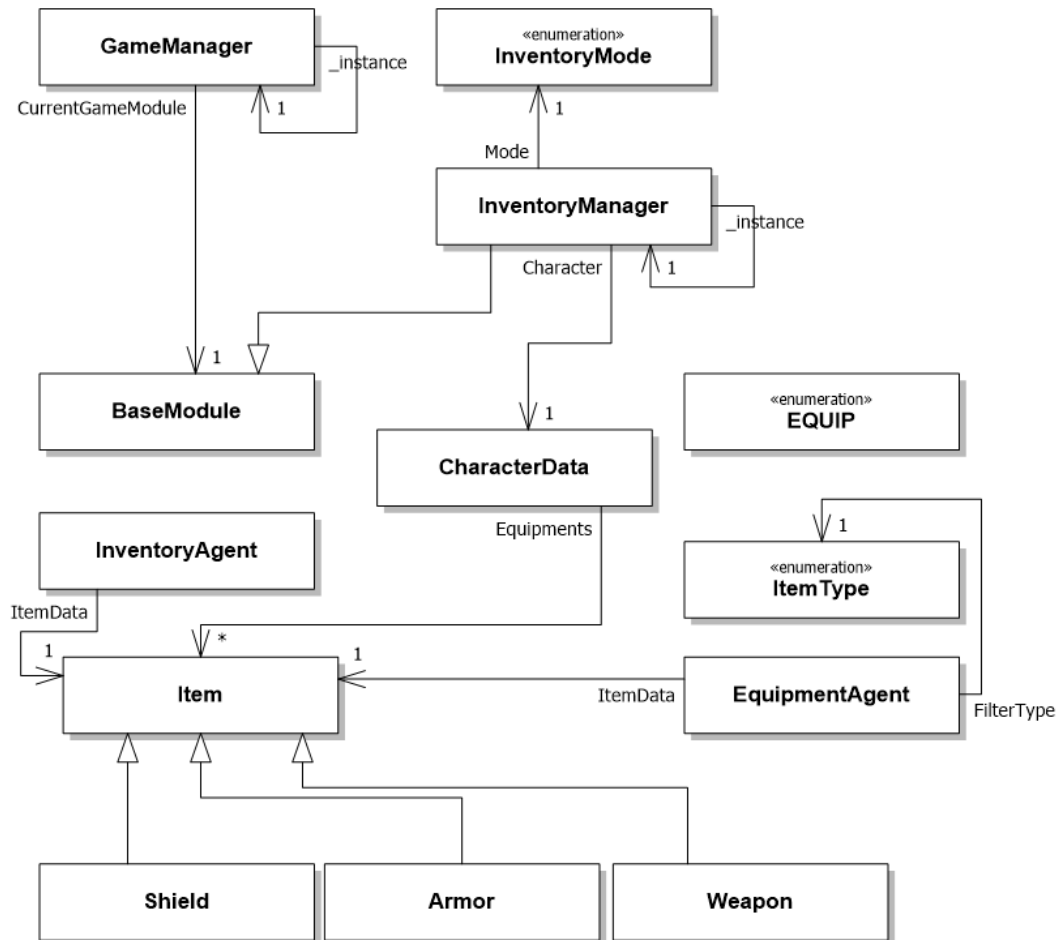


Figure 5.8: UML Diagram of the Inventory Module.

InventoryManager class oversees operations involving management of the items possessed by characters and all of the item transaction actions between characters. It has three different working modes. When the module activated, one of these modes selected as well by developers to specify what kind of actions this module is going to handle.

Personal mode offers actions to manage items belonging to a specified character. In this mode, two panels are shown to the player. The first one lists the items carried by the selected character and the second one shows the items already wielded or worn by that character. Players can change the items their characters wearing or wielding at the time, use items from those characters'

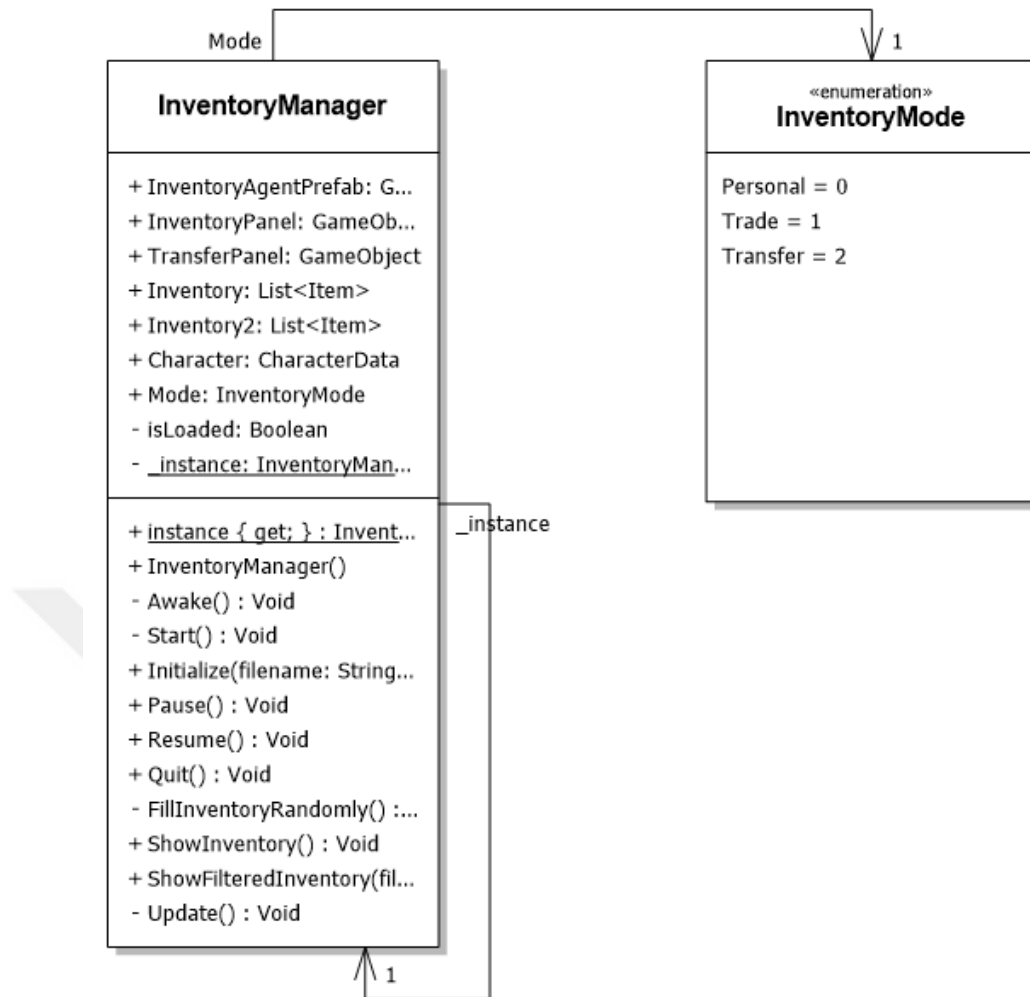


Figure 5.9: UML Diagram of the InventoryManager Class.

inventory if those items are consumables or drop one or more items if they are deemed unnecessary for them.

Transfer mode involves two different characters exchanging items they are carrying between each other. In this mode, two panels are shown to the player with each of them listing the items carried by the characters respectively. Players can swap items from one inventory to others freely. This mode is used for transferring items between player party members as well as can be used after a successful stealing attempt by the player character on some NPC.

The last mode is called Trade mode and involves trading items between characters. This mode works similar to the Transfer mode with only one exception, in which case exchanging items costs money to involving characters. This money cost is affected by the characters Trading skill as well.

InventoryAgent class is used by items inside of a character's inventory. It handles related items' transfer actions and how it is represented in the game.

EquipmentAgent class has a similar job as well. It does the same job of InventoryAgent class but instead of items inside an inventory, it handles items worn or wielded by a character, hence equipment.

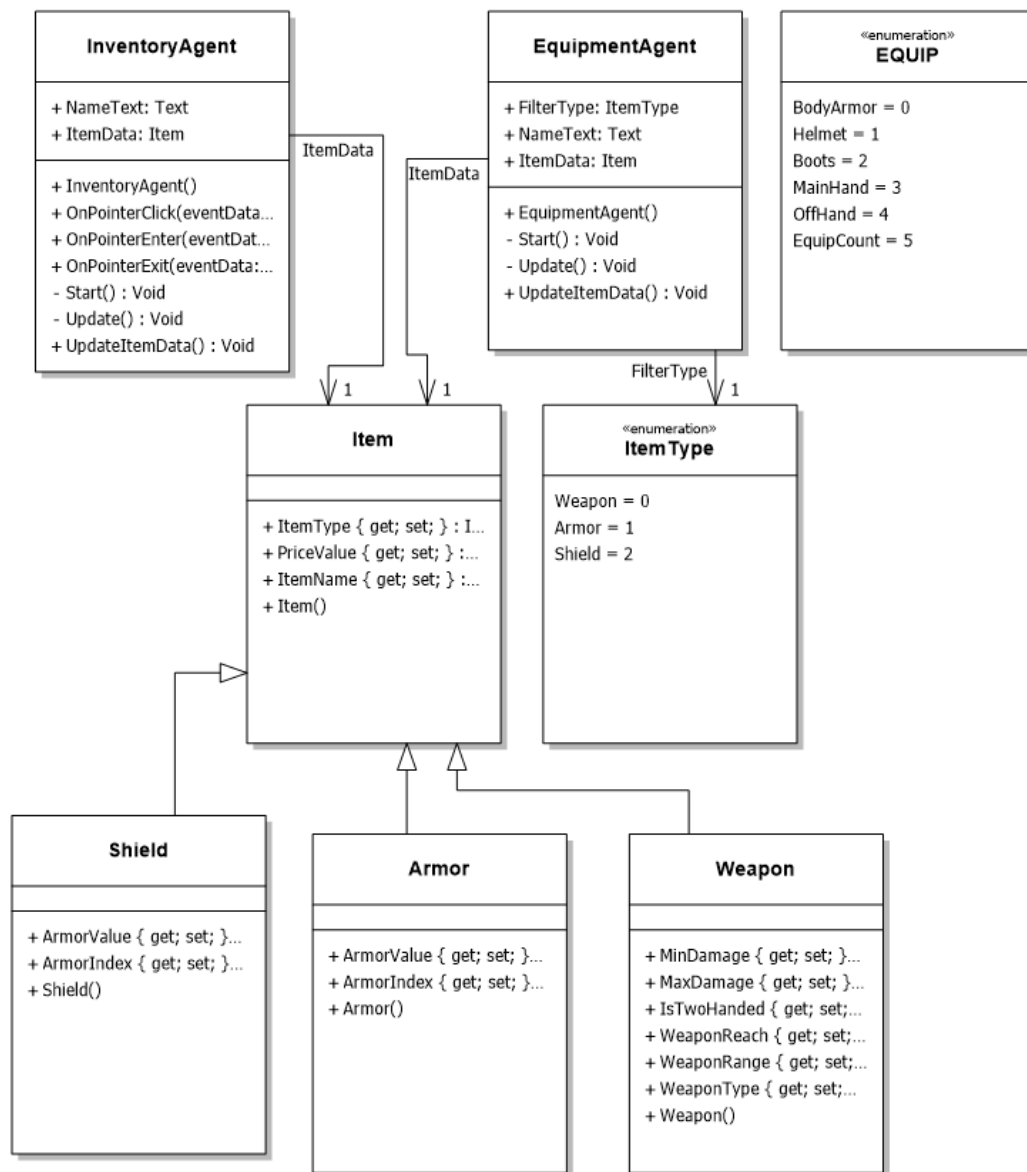


Figure 5.10: UML Diagrams of the InventoryAgent Class and EquipmentAgent Class.

CHAPTER 6

RESULTS AND DISCUSSION

This chapter will present an evaluation of the framework that was created for this thesis. This objective was achieved through the examination of the game projects created with the framework.

6.1 A Simple Text Adventure Game

Building a fully developed RPG would need a great amount of time and effort even with the help of this framework. It is best to keep it small when to build a game for testing the capabilities of the framework. With this reasoning, the first game built using the framework designed as a simple text adventure game. The main goal of this project is to build the smallest game possible with our framework. The game must be playable and narrative-driven. Although it is not a fully developed RPG, the game must focus on storytelling. To accomplish this, the Dialogue Module is selected to be the base of the game. Even with just one module, a game could be made with the framework built for this thesis.

The game featured a short story about a real-life event that took place in the bronze age Anatolia. The recently widowed Queen of Egypt, who was most often identified as Ankhesenamun, sent a letter to the Hittite King Suppiluliuma I to ask him to send her one of his sons to be a husband to her as she didn't have any heirs and didn't want to marry one of her subjects. The unprecedented thing about this is in the history of ancient Egypt while Pharaohs marry many foreign princesses as a part of political arrangements, Egypt never sends a princess to marry a foreign king. Unfortunately, Zannanza, the Hittite prince chosen to be sent to Egypt never reached his destination and only the news of him being killed on the road to Egypt reached back to Hattusa, the Hittite capital. This led to animosity between two empires which later resulted in warfare. [35]

For the being historically accurate, this game stays true to the events transpired. The event selected for the basis of the game story defined where and when the game starts, who is the game's protagonist and how the adventure ends, but what happened in between was lost in time. This part of the story was used to build the game upon itself.

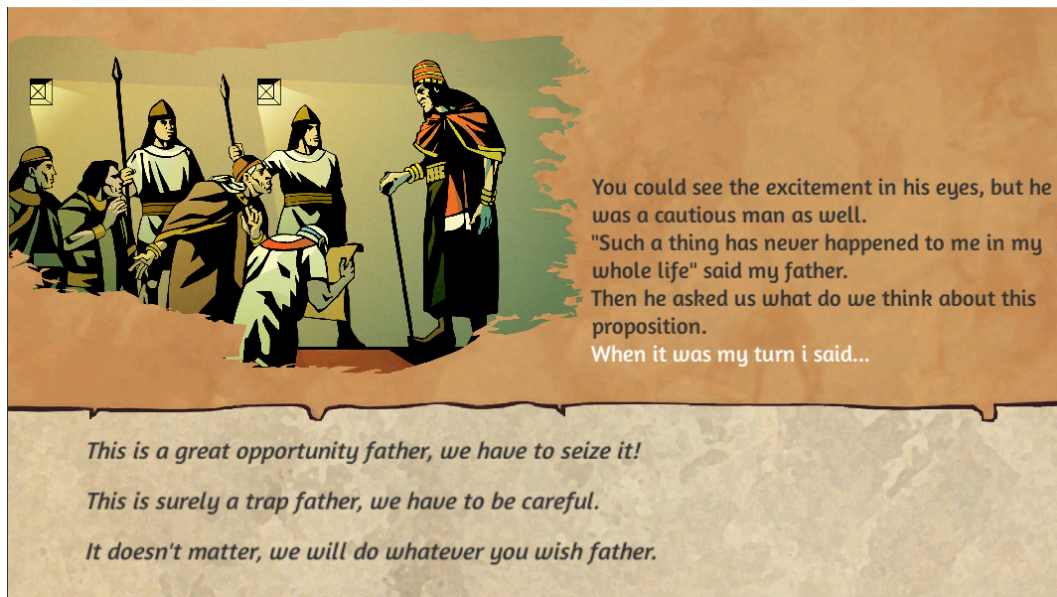


Figure 6.1: Screenshot from our game developed using our proposed framework.

These restrictions, first of all, eliminate the need for character selection or creation process at the start of the game as the protagonist of the game was already chosen. In real life, our protagonist couldn't reach his destination and only the news of him getting killed on the road reached back his home. While this eliminated the possibility of various good and bad endings the game still contains 3 different endings. Unfortunately, the protagonist dies in two of these endings and in the other one he just quits from his journey to start a new life and is presumed dead by others.

The greatest artistic liberty took place in the middle of the story between how the protagonist started his journey and how the news of his death reached his home. To make the journey interesting three random encounters were prepared. These encounters were placed in the time the protagonist leaves Hattusa and the border between the Hittite lands and the Egyptian lands. There were five stops planned between that time and on each stop a dice is rolled to see if the player encountered one of these or his journey was uneventful for now. Every encounter is a special occasion so each of them could only happen once. However, the randomized nature of these encounters leads to situations in which players could encounter none, one or more of those special encounters.

Since the player couldn't change the outcome of the game by way of his choices, these random encounters were all designed to be the experiences about the game world and opportunities for the player to role-play. As discussed before in previous chapters about role-playing games and their design philosophies while the player choice and agency is paramount there is still room for choices that do not change anything in the story but still allows players to role-play their characters in the game. [36]

The development of the game took approximately a month with irregular working days. Most of the work has been done over creating graphical assets and writing the dialogues. The framework's capabilities were already enough for the game so no additional programming was needed.

6.2 Improving The Text Adventure

The second game built using the framework was built upon the text adventure game that was previously made. The text adventure game that was previously built used only the dialogue module in the framework. This time more modules were used to show the capabilities of the framework better.

The same story has been told through this game as well. Besides the dialogue module, this game uses the travel module, the character system module and the combat module. The travel module shows the player the game world and the journey which the player makes during the game. The combat module is there to add a much-needed challenge element as the first game only offers dialogue choices but nothing else to challenge the player during the game. The character system module is used by the combat module, but no character development or leveling up offered during the game as the length of the game is too short to create opportunities for it. All the characters started with their base stats and skills and did not improve during the game. Only character development offered in the game is in the form of storytelling.



Figure 6.2: World map from example project.

The game starts as usual but after leaving Hattusa it passes over to the travel module. In the travel module, travel nodes were placed between Hattusa and

Egypt. The journey continues by visiting the nodes along the way. On each node, there is a chance to run across to one of those special encounters.

Another thing added to the game was combat encounters. In the text adventure, all of the combat took place in text form and players' only choice about these combat encounters could be made before and doesn't really change the outcome. In the new game, players have to face their enemies on a tactical battlefield. Players' choices and efforts would change the outcome of those combat encounters. Only the last battle's outcome couldn't be changed as the player must be killed before reaching Egypt. To ensure this in the last combat encounter players will face enemies stronger than themselves to force the player to lose. This is to show the players the futility of their journey.



Figure 6.3: A combat scene from example project.

The character system module is used to define combatants in the combat module but leveling up for any character is omitted for this game. Each character's data, including Zannanza, the protagonist of the game, is pre-defined in the character system module.

The development of the game took shorter than the former example game. That was understandable as the main part of the game was built earlier and only retrofitted to accommodate for the other modules used by the new game. The bulk of the work was on creating art assets. Module systems didn't need any additional coding as well.

6.3 Switch From 2D to 3D

In the third example project, graphics of the game were changed from 2D to 3D. The constant need for change and iteration throughout the development process is a common occurrence and one of the reasons why game development is difficult. During the development of a game project, many parts of it might be subjected to change. Game design decisions might change. New features could be added to the project or some features might be cut. All of these also create additional workload for the project as every change could affect other parts of the game. This example is built to simulate this kind of trouble for a game project and to show how our framework helps to handle issues like this.

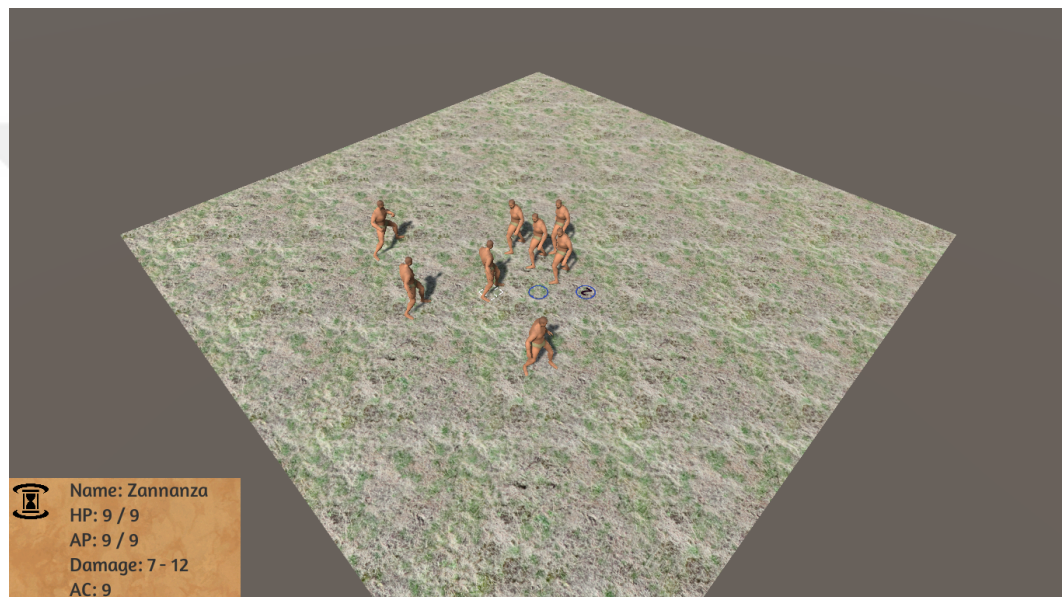


Figure 6.4: A combat scene from example project after the 2D to 3D transition.

To improve the graphical fidelity of the example game it has been decided to make the change for the 3D graphics. A normal game would need a considerable amount of work to overcome this kind of change. Fortunately, the distributed nature of our framework allows developers to easily make this kind of changes to their projects. The dialogue system still will be based on UI, the changes will only occur on the combat system and the travel system.

In the combat system, all the fighting takes place on a 2D grid. For the transition to 3D, this 2D grid is kept. This allows us to use the same game logic and AI with the new look. The only things that need to be changed are in the Viewer component of the combat system module, which is compromised of graphic assets, CharacterVisualizer and Combat2dCamera classes and Unity game object components used to render the images.

MeshRenderer component used instead of the SpriteRenderer component.

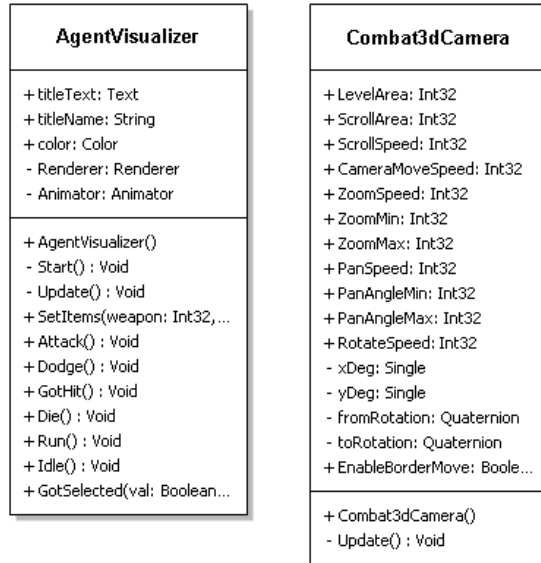


Figure 6.5: Uml diagrams for the new classes added to the Combat module.

Additionally CharacterVisualizer class changed to work with this component and use a 3D mesh to represent agents in the Combat module. Another camera class named Combat3dCamera was written for handling camera movements in 3d space. Lastly, new 3d character models were made for it.



Figure 6.6: A screenshot from Travel3d module.

After some inspection, it is revealed that the Travel module needs many more

changes than the Combat module. This time changing only the Viewer components won't be enough as the Travel module's Model component is based on 2D maps. It would be a very limiting and aesthetically poor choice to use those maps to render in 3D. In light of these revelations, it has been decided to make a new module for this task.

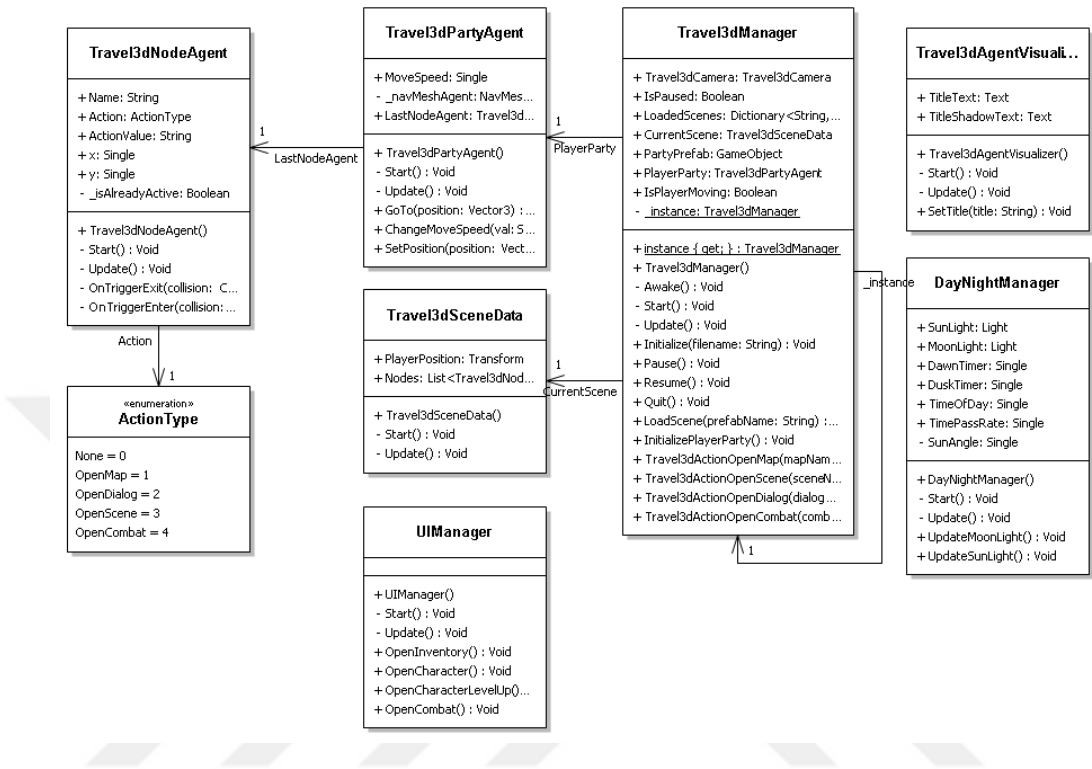


Figure 6.7: Uml diagrams for the Travel3d Module.

The new module is called the Travel3d module. The terrain of the mesh was generated with Unity's terrain tools. Points of Interest that the player agent could interact with were placed into the map as prefabs with Travel3dNodeAgent classes. Maps of this module consist of a terrain object and the node agents for points of interest that spread over the terrain. This allows developers to use Unity's own scene editor for building their own maps and save them as prefabs.

For independent agents that could travel across the map Travel3dPartyAgent class is used. In the example project, there is only the player party agent available, but it is possible to add other party agents that are controlled by AI. Travel3dAgentVisualizer class is used by both node and party agents to handle the visualization of their agents. For handling camera movement, the same class that has been used in the combat module was used here as well.

Changes with the code took a little shorter than a week and creating new art assets took nearly twice as longer than that. Apart from these changes, this is the same game from the first example. While they look like two different

games under the hood the same engine powers them and both games are played similarly.

6.4 User Study

A user study has been conducted to evaluate the framework with the users themselves. A group of local game developers and doctoral students were used to conduct this survey. The survey consisted of 15 questions divided into 3 parts. The first part is to get to know them and their involvement with game development. The second part is to learn about their thought and experiences of role-playing games. The last part is focused on the evaluation of the abilities of the framework.

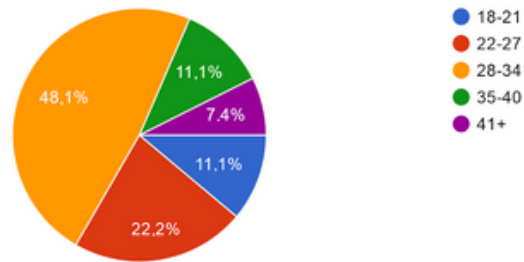
The total number of participants in this survey is 27. They have been asked about their age, gender, their experience in game development and area of their expertise. The biggest difference was the gender gap. From all of the participants of the survey, only 18.5% were females and 81.5% were males. In terms of age, there were some participants for every age group with 28-34 years old group has twice as much as people in it than every other group.

The participants' years of experience in game development are evenly distributed. In the survey, 18.5% of the participants reported that their game development experience was 0-1 year, 22.2% of the participants reported that their experience in game development was 1-3 years. As the smallest group, only 7.4% of the participants reported their experience in the field were 3-6 years. After that 18.5% of the participants reported with 6-10 years of experience and 33.3% of the participants reported with 10+ years of experience in game development as the biggest group.

The participants have been asked about the areas they were interested in. Multiple choices were allowed for people working in different areas. Every area except the sound design has been selected. The most selected answer was Programming with 66.7%. The second most popular area was Game Design with 51.9%. 3D Modelling was the third most popular area with 14.8%. It was followed closely by Producers and people selected the Other option with 11.1%. All the other options left except Sound Designer has been selected once with 3.7%. This shows that participants of the survey as a whole have mastery over game development. It can be argued that this distribution shows the adequacy of the survey study.

In the next part, the participants asked about their experiences and opinions on role-playing games. When asked how hard is making a role-playing game 37% of them answered as the hardest genre to develop, 29.6% of them answered as a bit harder than normal, 29.6% of the participants answer as normal and only 3.7% of them answered as not at all. It can be said that a considerable amount of them find developing role-playing games is harder than other games.

How old are you?
27 answers



What is your gender?
27 answers

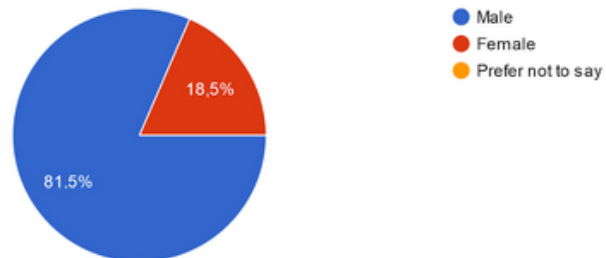


Figure 6.8: Age and Gender charts from the survey.

How many years have you been interested in game development?
27 answers

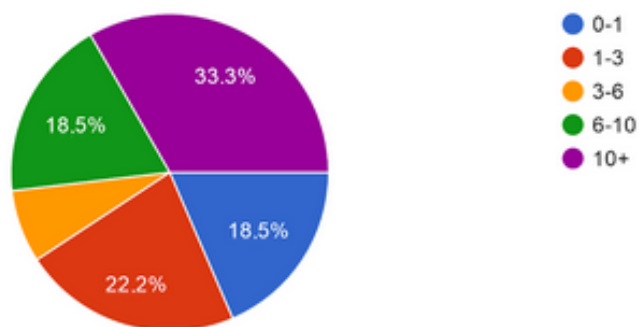


Figure 6.9: Years of experience chart for the participants.

Which parts of development do you work on?
27 answers

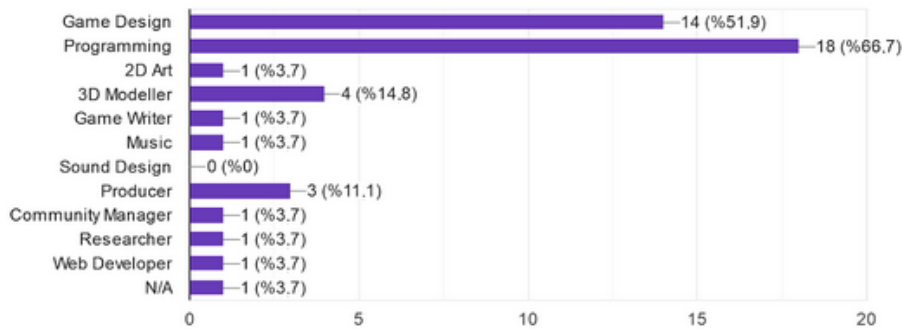


Figure 6.10: Which parts of development do you work on?

When asked about what is the most important part of a role-playing game more than half of the participants answered it as the story of the game. As the second most popular answer with 33.3% was the game mechanics. Lastly, the visuals have been answered with 11.1%.

When asked about what is the most challenging part of developing a role-playing game more than half of the participants answered it as it was the game mechanics. Followed closely by 40.7% of them answering it as the story. Only 7.4% of them answered it as the visuals. This shows us that what is important for role-playing games are also the most challenging parts of its development.

Table 6.1: What do you think is the most challenging part in developing / most important part in role-playing games?

	Story	Visuals	Game Mechanics	Musics
Most Important	55.6%	11.1%	33.3%	0%
Most Challenging	40.7%	7.4%	51.9%	0%

Lastly, the participants asked about whether they made a role-playing game before. 70.4% of them answered with no. Only 29.4% of them previously worked on a role-playing game. When they have been asked whether they used a specific game development tool for role-playing games before only 11.5% said yes.

For the last part of the survey, we let the participants review our framework and answer the following questions with their impressions. The first question was how much do they think our framework would contribute to the development process of role-playing games. None of the participants selected the not at all option. Most of the participants thought our frameworks' ef-

fect would be fairly on role-playing games with 33.3%. It has been followed closely by those who thought its effect would be medium with 29.6%. While 22.2% of them thought its effect would be a little bit, 14.8% of the participants thought its effect would be very high.

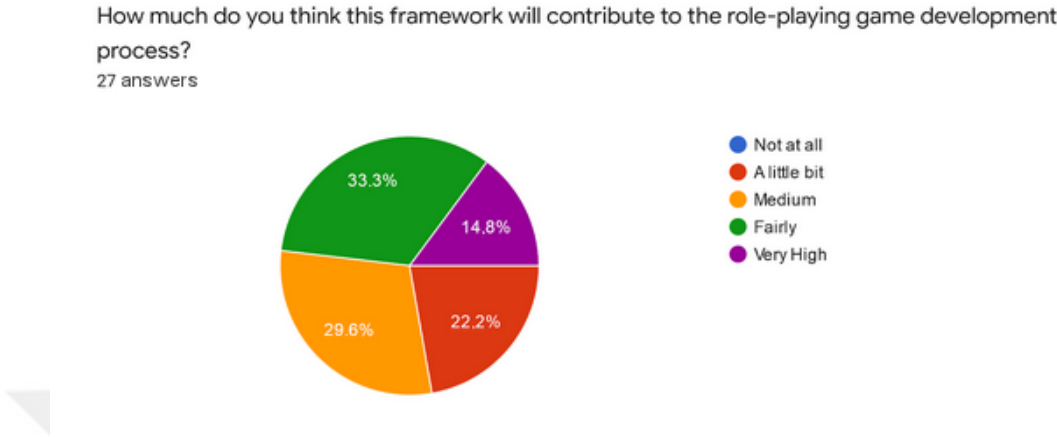


Figure 6.11: How much do you think this framework will contribute to the role-playing game development process?

After that, the participants asked to choose which module they find the most useful and the least useful for developing role-playing games. Unfortunately, everyone's choice contradicts someone else's with those questions. Still, it can be said that the most useful module for the participants was the Dialogue module with 29.6%, followed by Character module with 22.2% and Combat module with an 18.5% approval rate. The least useful modules were Travel and Travel3d modules with 25.9%, followed by Combat module with an 18.5% disapproval rate.

Table 6.2: Which module do you find the most / least useful for developing role-playing games?

	Dialogue	Combat	Travel3d	Inventory	Character	Travel
Most	26.9%	18.5%	7.4%	11.1%	22.2%	11.1%
Least	11.1%	18.5%	25.9%	11.1%	7.4%	25.9%

As they reach the end of the survey, the participants asked whether or not they would use our framework to develop role-playing games in the future. Also as a follow-up question, they have been asked whether or not they would use our framework to develop games that aren't role-playing games. For developing role-playing games 44.4% of the participants answered it with maybe, 18.5% of them answered it with probably and 22.2% of them answered it with definitely. Only 14.8% of them answered the question with not at all. For developing games other than role-playing games 55.6% of the partici-

pants answered it with maybe, 14.8% of the participants answered it with probably and 7.4% of them answered it with definitely. This time those answered with not at all choice increased to 22.2%.

Table 6.3: In the future, would you consider using this framework to develop your games?

	Not At All	Maybe	Probably	Definitely
For RPGs	14.8%	44.4%	18.5%	22.2%
For Other Games	22.2%	55.6%	14.8%	7.4%

6.5 Discussion

Through the example projects, the framework of the thesis is proven to be capable of building different kinds of games that more or less focused on their narration and/or world-building. It is shown that it can be used to build adventure games with its dialogue module. Its combat module can be used to create tactical combat games along with the character system module. When all the modules combined it is a powerful tool to create RPGs.

While it does help to make the development process go faster than usual, using only one module like in the first example game might be overkill in some cases. In this case, using Twine would probably be a valid solution as well. By using it with other modules our framework can be used to create better adventure games all the while retaining the convenience of using Twine.

The second example game successfully showcases the framework’s scalability by making the same game much bigger in size and new game mechanics. The framework allows developers to add new mechanics and features to their games relatively easy. Other solutions like Twine couldn’t offer this kind of scalability with the same convenience. By carefully choosing which modules they will use and which modules they will omit, the developers can create adventure games or RPGs using our framework with the same ease of development.

The third example game successfully demonstrates the framework’s capabilities to help developers work in an iterative game development process with ease. Changes in the projects are inevitable during the development process. Our framework enables developers to handle changes and iterations through the development with grace. This is a feature that no other solutions in this problem space offer properly.

The user study has shown us that our framework received some positive reactions from the participants. In the user study, we see that developers need and want for role-playing games are in line with our initial problem statement. The participants’ assessment of our framework shows that they think it would make a significant contribution to the development process of a role-

playing game. When asked if they would use our framework for their project in the future, the participants have shown some real interest in the framework. Nevertheless, most of the developers just answered it with maybe. The framework looks interesting for developers but it lacks features to create enthusiasm for them. This can be seen in the questions for the most and least useful module. While some modules shine, not one module gets praise from the majority of the participants. Adding a new feature or module that will create excitement in developers might solve this problem and change those "maybe" answers to "definitely". Nonetheless, we can argue that we achieved our goal of creating a framework for small developers to make role-playing games more easily.





CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

With the example games, it has been clearly shown that the framework proposed in this thesis is a capable solution for the development of role-playing and adventure games. Its use of a diverse set of free to use third-party development tools and the ability to combine it with its module system that offers many common features of said games create an environment that helps developers work faster with relative comfort.

Also, its modular structure and use of MVC pattern in the design of those modules create opportunities for developers to iterate quickly with creating less complication to do so. When the importance of iterative development is taken note of, this quality is maybe the strongest point of the framework proposed.

Users could create different kinds of games by using different modules. By changing the existing modules or adding new ones, developers could create vastly different kinds of games. This also makes this framework a valuable tool for developers. From the first example to the last one, playable games were built with this framework. In the overall workload of these projects coding part always took the smallest burden. Also, most of the time rather than adding new features, cutting the existing features is more needed.

The framework in this form might be a bit redundant to use in developing smaller games like game jam games. It might be insufficient in some respect for commercial projects with big budgets. Though it is a valued tool for small to middle independent game developers with smaller budgets, non-commercial game developers like hobbyists, researchers and serious game developers.

The user study showed us that our initial problem statement was right. It also showed that there is some interest in our framework with the game developers. However, it also showed areas where our framework needs improvement.

Firstly there was a genuinely positive reaction from the participants for the framework. Secondly, the participants of the user study were highly diverse.

The age groups of the participants were equally distributed and the peak was in the middle age group of 28-34. Also, interest areas of the participants were highly diverse too. The only interest area that was left unselected was the sound designer. Every other area of interest was selected by some participants. This diversity among the participants increases the reliability of the study.

The answers of the participants shed light on many aspects of the framework. Their positive approach to the framework was encouraging. There was a certain interest in using the framework. Yet it seems possible to increase this interest further. The results obtained from the user study will be overseen in this regard. Particularly, the sections in which they specify the modules they like and dislike show which parts are successful and which parts fail. Thanks to the distributed structure of the framework, it will be quite simple to change problematic parts or to remove and add new modules if necessary.

7.2 Future Work

The modular structure of the framework allows it to grow to any direction the game developer using it wants or needs. For starters, procedural content generation could be a useful addition for the developers. Improved AI is another area that could be useful for developers as well. An effective tool for game designers might also help us to entice the undecided users to choose our framework.

Right now the framework's pipeline uses many great tools which are proved their capabilities by the many free and commercial projects which used these tools. Still, all of the tools are free of charge and available to be used by everyone. But this distributed situation creates a messy development environment. While this does not hinder the development process it does make it harder for new developers as they have to learn these tools along with the framework.

Another area of improvement would be better MVC abstraction in various modules. As stated before while MVC principles were carefully considered there were some parts that principles ignored for the sake of simplicity and ease of use. With more time those parts could be refactored to have better abstraction without losing their simplicity and ease of use.

The last important subject that needs to be added was modding support. Game modding is popular among many gamers and aspiring game developers and it has been proved again and again that mods developed by the community do greatly increase a game's longevity and its reach. Many RPGs feature some kind of modding capabilities for their players. It is an important feature that needs to be developed.

REFERENCES

- [1] Johan Huizinga. *Homo Ludens* IIs 86. Routledge, 2014.
- [2] The Entertainment Software Association. US Video Game Industry Revenue Reaches \$36 Billion in 2017. <http://www.theesa.com/article/us-video-game-industry-revenue-reaches-36-billion-2017/>, 2019. Online; accessed 23-January-2019.
- [3] Statista. North American box office revenue 1980-2018 | Statistic. <https://www.statista.com/statistics/187069/north-american-box-office-gross-revenue-since-1980/>, 2019. Online; accessed 23-January-2019.
- [4] Roger Ebert. Video games can never be art: Roger ebert's journal: Roger ebert.
- [5] Leonard A. Annetta. The "I's" have it: A framework for serious educational game design. *Review of General Psychology*, 14(2):105–112, 2010.
- [6] Mansureh Kebritchi et al. Examining the pedagogical foundations of modern educational computer games. *Computers & Education*, 51(4):1729–1743, 2008.
- [7] Marina Papastergiou. Exploring the potential of computer and video games for health and physical education: A literature review. *Computers & Education*, 53(3):603–622, 2009.
- [8] Becker Katrin. How are games educational? learning theories embodied in games. In *DiGRA ཁ - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*, 2005.
- [9] Dominique Archambault, Thomas Gaudy, Klaus Miesenberger, Stéphane Natkin, and Rolland Ossmann. Towards generalised accessibility of computer games. In *International Conference on Technologies for E-Learning and Digital Entertainment*, pages 518–527. Springer, 2008.
- [10] Stéphane Natkin. *Video games and interactive media: A glimpse at new digital entertainment*. AK Peters/CRC Press, 2006.
- [11] Luís Valente and Aura Conci. Guff : A game development tool. 2005.

- [12] Ahmed BinSubaih and Steve Maddock. Game portability using a service-oriented approach. *Int. J. Comput. Games Technol.*, 2008:3:1–3:7, January 2008.
- [13] Jennifer Ann Grouling. Cover. Tabletop role-playing games [electronic resource] : perspectives from narrative, game, and rhetorical theory /. 01 2019.
- [14] Travis M Gasque. *Design agency: Dissecting the layers of tabletop role-playing game campaign design*. PhD thesis, Georgia Institute of Technology, 2016.
- [15] John E Laird. History of computer games. *EECS Department*, 9(7):05, 2005.
- [16] Christopher Dristig Stenström. *Gameplay design for role-playing battle systems*. 2012.
- [17] Jonas Carlquist. Playing the story: Computer games as a narrative genre. *Human IT: tidskrift för studier av IT ur ett humanvetenskapligt perspektiv*, 6, 01 2002.
- [18] Greg Costikyan. Where stories end and games begin. *Game Developer*, 7(9):44–53, 2000.
- [19] Aaron A Reed, Ben Samuel, Anne Sullivan, Ricky Grant, April Grow, Justin Lazaro, Jennifer Mahal, Sri Kurniawan, Marilyn Walker, and Noah Wardrip-Fruin. A step towards the future of role-playing games: The spyfeet mobile rpg project. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.
- [20] M. Barton. *Dungeons and Desktops: The History of Computer Role-Playing Games*. Ak Peters Series. Taylor & Francis, 2008.
- [21] Brenda Brathwaite and Ian Schreiber. *Challenges for Game Designers*. Charles River Media, Inc., Rockland, MA, USA, 1 edition, 2008.
- [22] Barbaros Bostan and Ugur Kaplancali. Explorations in player motivations: Game mechanics. pages 5–11, 01 2009.
- [23] Teresa Dillon. Adventure games for learning and storytelling. *UK, Futurelab Prototype Context Paper, Adventure Author*, 2005.
- [24] RPG Maker. RPG Maker MV. <http://www.rpgmakerweb.com/products/programs/rpg-maker-mv>, 2019. Online; accessed 1-February-2019.
- [25] Twinery. Twine. <http://twinery.org/>, 2019. Online; accessed 1-February-2019.

- [26] Ioannis Pachoulakis and Georgios Pontikakis. Combining features of the unreal and unity game engines to hone development skills. 11 2015.
- [27] A. Andrade. Game engines: a survey. *EAI Endorsed Transactions on Game-Based Learning*, 2:150615, 11 2015.
- [28] Ilja Astahovs. Use of design patterns for mobile game development, 2012.
- [29] Thibault Raffailac and Stéphane Huot. Polyphony: Programming interfaces and interactions with the entity-component-system model. *Proceedings of the ACM on Human-Computer Interaction*, 3(EICS):8, 2019.
- [30] Dieter Schmalstieg and Andre Stork. Unified patterns for realtime interactive simulation in games and digital storytelling. *IEEE computer graphics and applications*, 39(1):100–106, 2019.
- [31] Toni Härkönen. Advantages and implementation of entity-component-systems. 2019.
- [32] Unity Technologies. Unity 3D. <https://www.unity3d.com/>, 2018. Online; accessed 23-December-2018.
- [33] J. Xie. Research on key technologies base unity3d game engine. In *2012 7th International Conference on Computer Science Education (ICCSE)*, pages 695–699, July 2012.
- [34] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [35] Trevor R Bryce. The death of niphururiya and its aftermath. *The Journal of Egyptian Archaeology*, 76(1):97–105, 1990.
- [36] Michaël Oosterlinck. Role-playing in role-playing games.



Appendix A

UML DIAGRAMS

Below you can find more UML Diagrams for the framework built for this thesis to provide additional details.

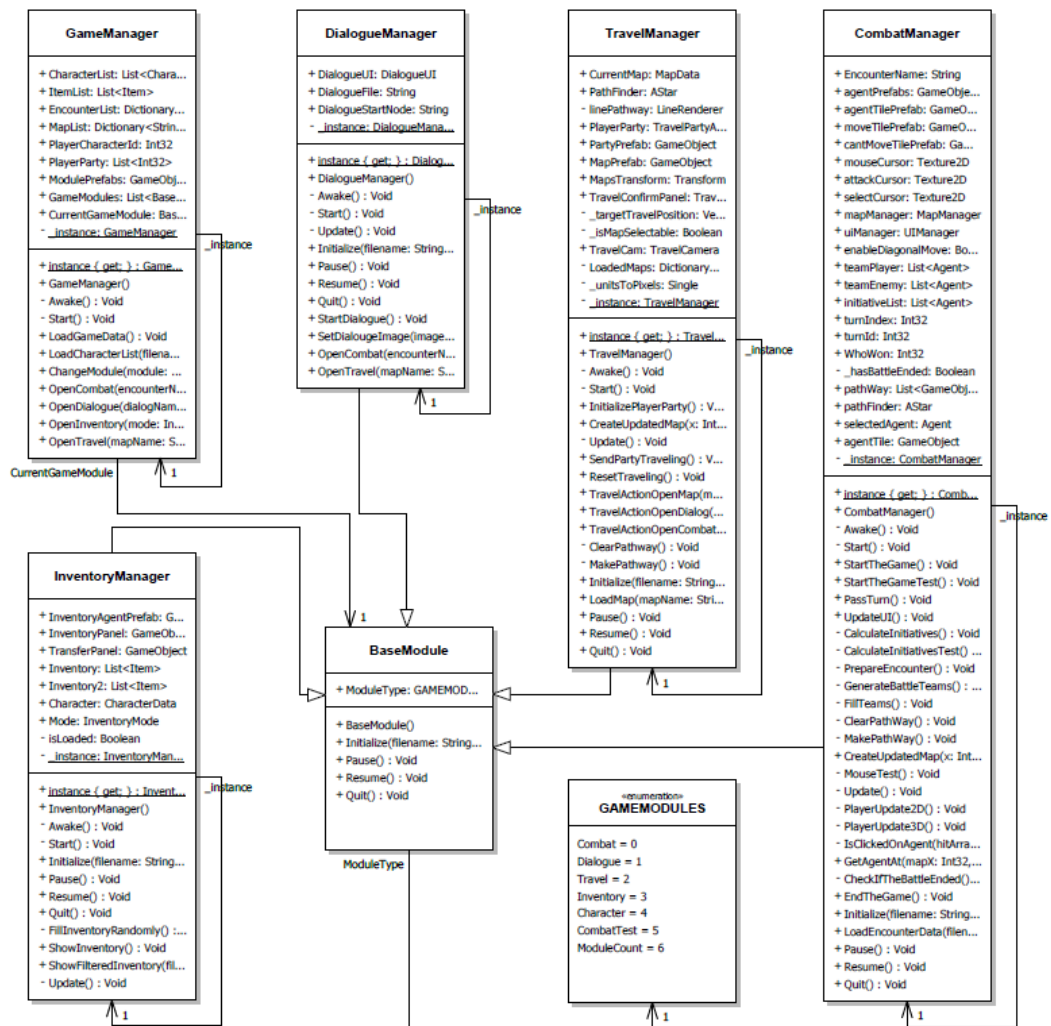


Figure A.1: UML Diagram of all the System Modules.

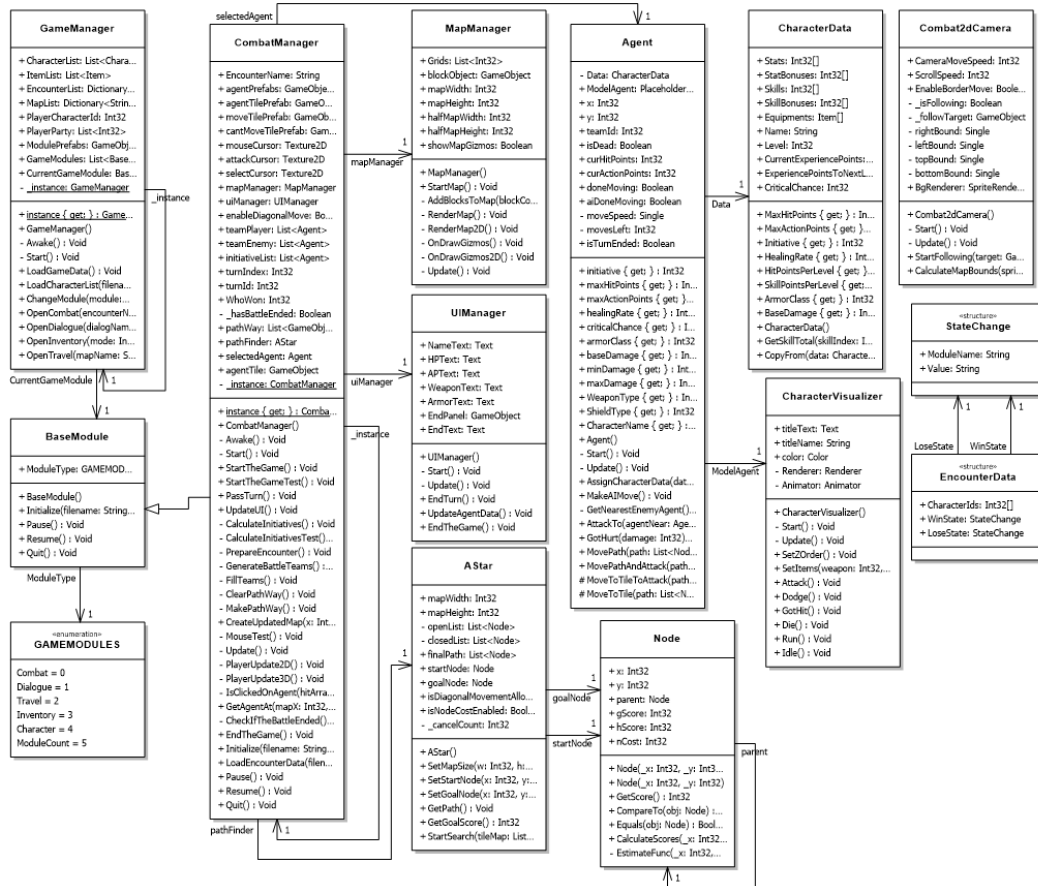


Figure A.2: UML Diagram of the Combat Modules.

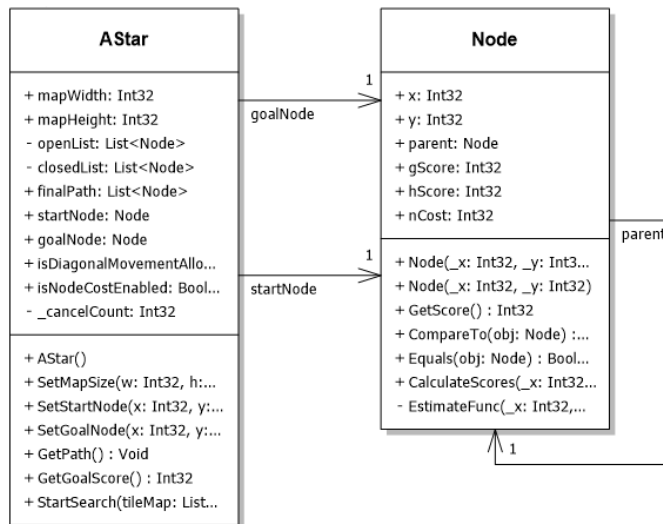


Figure A.3: UML Diagram of the A* Class.

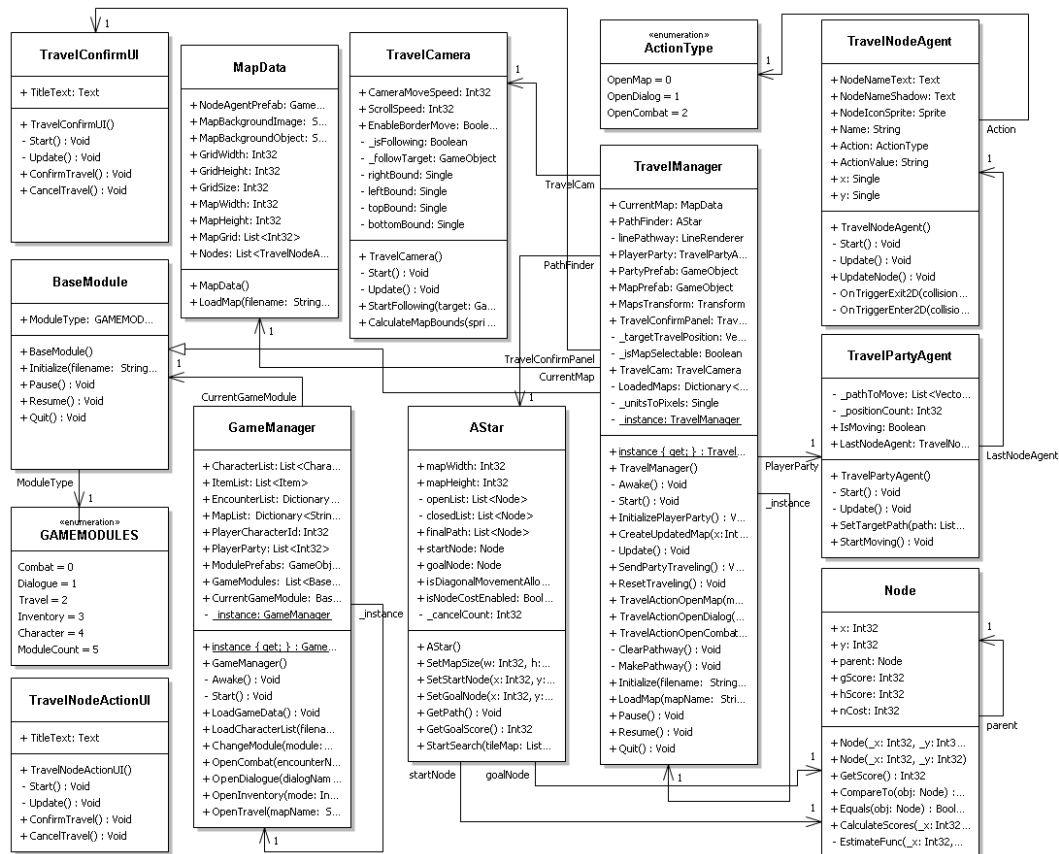


Figure A.5: UML Diagram of the Travel Modules.

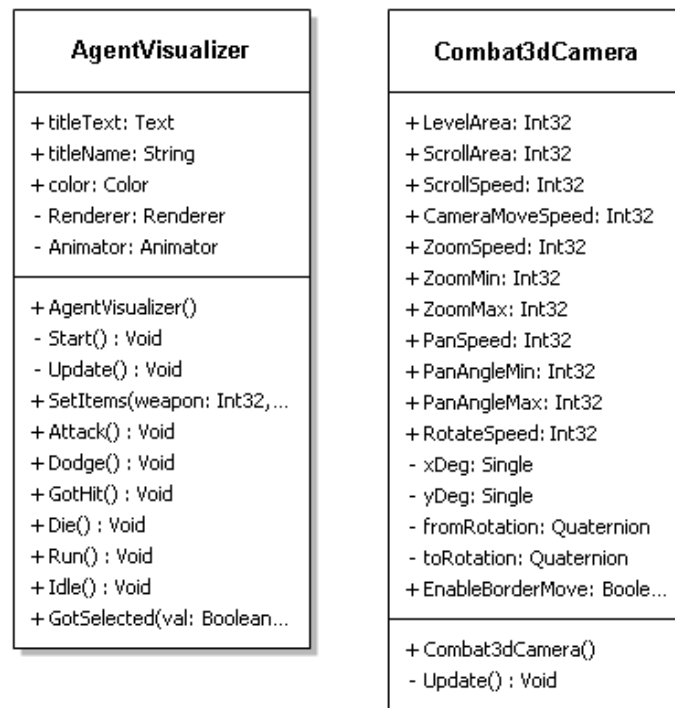


Figure A.6: Uml diagrams for the new classes added to the Combat module.

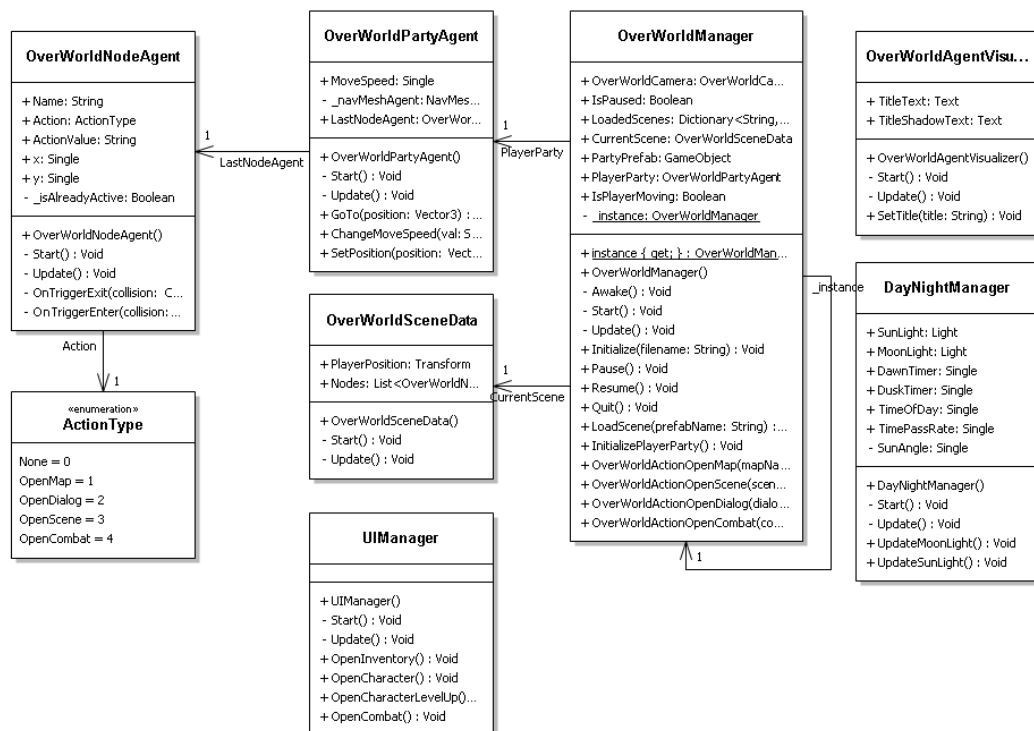


Figure A.7: Uml diagrams for the Travel3d Module.

Appendix B

CODE SAMPLES

Below you can find some code examples from the framework built for this thesis. They are listed here to provide an insight into the implementation of the framework.

```
public enum GAMEMODULES
{
    Combat = 0,
    Dialogue,
    Travel,
    Inventory,
    Character,
    ModuleCount
};

public class BaseModule : MonoBehaviour
{
    public GAMEMODULES ModuleType;

    public virtual void Initialize(string filename) { }
    public virtual void Pause() { }
    public virtual void Resume() { }
    public virtual void Quit() { }
}
```

Code: B.1: BaseModule class

```
public void ChangeModule(GAMEMODULES module)
{
    foreach(var gameModule in GameModules)
    {
        if (gameModule.ModuleType == module)
        {
            if (CurrentGameModule != null)
            {
                CurrentGameModule.Pause();
                CurrentGameModule.gameObject.SetActive(false);
            }
            gameModule.gameObject.SetActive(true);
            gameModule.Resume();
            CurrentGameModule = gameModule;
        }
    }
}
```

```

    }
}

public void OpenCombat(string encounterName)
{
    ChangeModule(GAMEMODULES.Combat);
    Combat.CombatManager.instance.EncounterName = encounterName;
    Combat.CombatManager.instance.StartTheGame();
}

public void OpenDialogue(string dialogName)
{
    ChangeModule(GAMEMODULES.Dialogue);
    Dialogue.DialogueManager.instance.DialogueStartNode =
        dialogName;
    Dialogue.DialogueManager.instance.StartDialogue();
}

public void OpenInventory(Inventory.InventoryMode mode)
{
    ChangeModule(GAMEMODULES.Inventory);
    Inventory.InventoryManager.instance.Mode = mode;
}

public void OpenTravel(string mapName)
{
    ChangeModule(GAMEMODULES.Travel);
    Travel.TravelManager.instance.LoadMap(mapName);
}

```

Code: B.2: The code that is responsible for Module switching and activation

```

private void OnTriggerEnter2D(Collider2D collision)
{
    var player =
        collision.gameObject.GetComponent<TravelPartyAgent>();
    if (player != null)
    {
        if (player.LastNodeAgent == this) return;

        player.LastNodeAgent = this;
        if (Action == ActionType.OpenCombat)
        {
            TravelManager.instance.TravelActionOpenCombat(ActionValue);
        }
        else if (Action == ActionType.OpenDialog)
        {
            TravelManager.instance.TravelActionOpenDialog(ActionValue);
        }
        else if (Action == ActionType.OpenMap)
        {
            TravelManager.instance.TravelActionOpenMap(ActionValue);
        }
    }
}

```

Code: B.3: Resolving a triggered node action

```
[YarnCommand("set_dialog_image")]
public void SetDialogueImage(string imageName)
{
    DialogueUI.ImagePanel.SetActive(true);
    DialogueUI.ImageObject.sprite =
        (Sprite)Resources.Load(imageName, typeof(Sprite));
}

[YarnCommand("close_dialog_image")]
public void CloseDialogueImage()
{
    DialogueUI.ImagePanel.SetActive(false);
}

[YarnCommand("open_combat")]
public void OpenCombat(string encounterName)
{
    Debug.Log("open_combat_command: " + encounterName);
    GameManager.instance.OpenCombat(encounterName);
}

[YarnCommand("open_travel")]
public void OpenTravel(string mapName)
{
    Debug.Log("open_travel_command: " + mapName);
    GameManager.instance.OpenTravel(mapName);
}

[YarnCommand("roll_dice")]
public void RollDice(int maxRnd)
{
    int roll = UnityEngine.Random.Range(0, maxRnd);
    VariableManager.SetValue("$roll_result", new Yarn.Value(roll));
}

[YarnCommand("enable_map_node")]
public void EnableMapNode(string mapName, string nodeName)
{
    Travel.TravelManager.instance.TravelActionEnableNode(mapName,
        nodeName);
}

[YarnCommand("disable_map_node")]
public void DisableMapNode(string mapName, string nodeName)
{
    Travel.TravelManager.instance.TravelActionDisableNode(mapName,
        nodeName);
}

[YarnCommand("enable_map_node_by_id")]
public void EnableMapNodeById(string mapName, int nodeId)
{

```

```
        Debug.Log("enable_travel_node: " + nodeId.ToString());
        Travel.TravelManager.instance.TravelActionEnableNodeById(mapName,
            nodeId);
    }

    [YarnCommand("disable_map_node_by_id")]
    public void DisableMapNodeById(string mapName, int nodeId)
    {
        Debug.Log("disable_travel_node: " + nodeId.ToString());
        Travel.TravelManager.instance.TravelActionDisableNodeById(mapName,
            nodeId);
    }

    [YarnCommand("quit_game")]
    public void QuitGame()
    {
        Debug.Log("quit game");
        Application.Quit();
    }
}
```

Code: B.4: Some Yarn commands

Appendix C

DEVELOPER TOOLS

C.1 Character Generator

As the first of the developer tools provided with the framework a character generator tool is developed.

A list of characters provided from the CharacterList.xml file is listed in the main window. Every character in the list can be deleted or edited.

Users can also add new characters to the list. Another UI prepared for creating a new character.

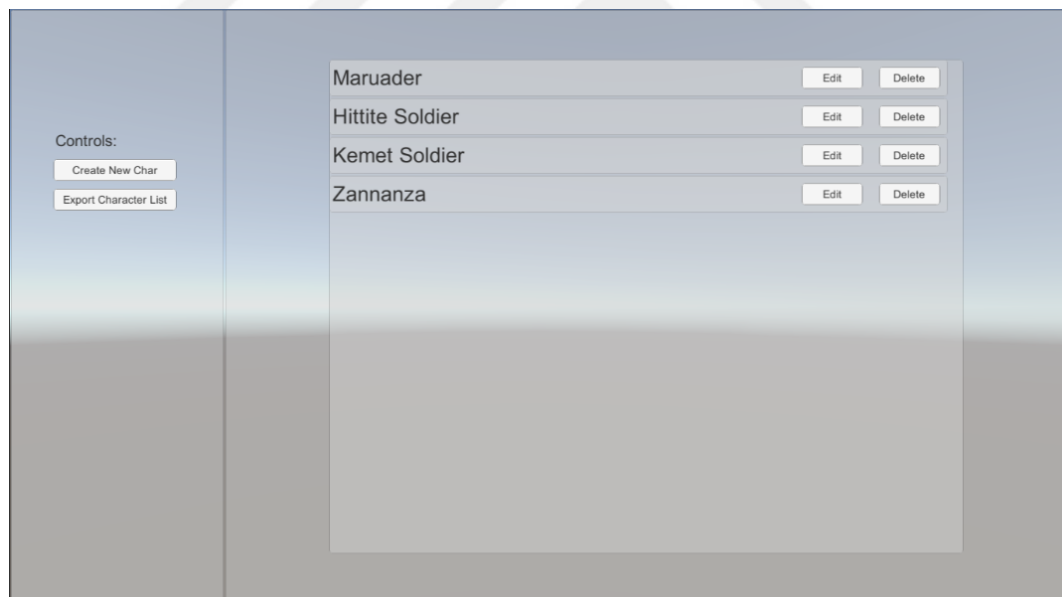


Figure C.1: Main screen of Character Generator Tool.

In the first screen, users can decide upon primary and secondary stats for the character they are creating.

The second screen is for users to assign the characters' skills. Base skill points and points they can spend on skills are determined from primary stats setup

STATS		STATS	
Points Left: 25			
Strength	6	+	-
Agility	4	+	-
Endurance	6	+	-
Intelligence	4	+	-
Willpower	7	+	-

Figure C.2: First screen from Character Generator Tool.

before.

Skills		SKILLS			
Points Left: 26					
Melee 13	+	-	Herbalism 12	+	-
Ranged 20	+	-	Leadership 18	+	-
Dodge 25	+	-	Trading 23	+	-
Shield 10	+	-	Healing 8	+	-
Mysticism 22	+	-	Persuasion 15	+	-
Crafting 16	+	-	Nature Lore 25	+	-

Figure C.3: Second screen from Character Generator Tool.

The same screens are used for editing the existing characters as well.

When all the characters users wanted in their game are created and/or edited then users can use the export button to create a new CharacterList.Xml file to use in their games.

Appendix D

DOCUMENTATION

D.1 What is it?

It is a game development framework for creating computer role-playing games. It consists of several modules each handling a specific game mechanic and communicating with other modules to pass data as needed to run the game. Each module designed to run as a single instance. XML files are used to store data for the game. Every module used by the game defined in XML files and each of them has specific XML files to read their data. Developers could also create and add their modules to use with the other modules or they can alter the existing modules according to their needs.

D.2 XML Files

Games developed with this framework uses several XML files to specify the game data.

D.2.1 Gamedata.xml

Gamedata.xml file used for stating the modules used in the game. The main tag it uses for its root element is "Game". It is to define the game developer is working on. It has two attributes. The first one is "startModule" and the second one is "startValue". Former is for stating the module the game starts with and the latter one is for stating that modules starting value.

The root element contains several child elements to specify the modules that will be used in the game. "Module" tag is used for child elements. Every child represents a module used in the game. As stated before every module can be used once. Child elements have two important attributes. The first one is called "type" and it is to define which kind of a module it is. The second one is called "filename" and it is to point to the XML file which is storing the data of this module. Developers could use other attributes in the modules they developed but these two are a must.

```
<Game startModule="Travel3d" startValue="worldmap">
```

```

<Module type="Inventory" filename="ItemList.xml" />
<Module type="Character" filename="CharacterList.xml" />
<Module type="Travel" filename="MapList.xml" />
<Module type="Combat" filename="EncounterList.xml" />
<Module type="Dialogue" filename="zannanza3.yarn.txt" />
<Module type="Travel3d" filename="SceneList.xml" />
</Game>

```

Code: D.1: Gamedata.xml file

D.2.2 ItemList.xml

ItemList.xml is used by the Inventory module. This module oversees all the items in the game and all the inventories used by players and non-player characters.

Its root element is the "Items" tag. It has two child elements to represent two different things these modules manage, Items and inventories. Although their ordering is not important in the XML file, the first one is "InventoryList" and the second one is "ItemList".

"InventoryList" element has child elements with the tag "Inventory". Each of these elements represents an inventory in the game. Their only attribute is called "Name" and used for giving them a name to be able to access them whenever needed. Each item in those inventories represented by a child element with the "InventorySlot" tag. Each of those child elements has two attributes. The first one is "ItemId" to link them to an item in the Item List with its index number. The second one is "Amount". Most items in the game are single in their inventory slots but some items are stackable. This attribute is for signifying how many of these items in that slot. For single items always use the value of 1.

The "ItemList" element lists all the items in the game as its child elements. Each child element with the tag "Item" represent a different item in the game. Its attributes change accordingly to the type of item it is representing. There are different types of items defined in the game like Weapons, Armor, Shield and such. Whatever its type, every "Item" element has the same basic attributes. They are "Type", "Name", "PriceValue". Other than them every other item type has several attributes according to their needs. There are also optional attributes "IsStackable" and "MaxStack".

```

<Items>
  <InventoryList>
    <Inventory Name="PlayerInventory">
      <InventorySlot ItemId="0" Amount="1" />
      <InventorySlot ItemId="5" Amount="1" />
      <InventorySlot ItemId="10" Amount="1" />
      <InventorySlot ItemId="30" Amount="1" />
    </Inventory>
    <Inventory Name="MarketInventory">
      <InventorySlot ItemId="0" Amount="1" />

```

```

        <InventorySlot ItemId="5" Amount="1" />
        <InventorySlot ItemId="10" Amount="1" />
    </Inventory>
</InventoryList>
<ItemList>
    <Item Name="Flint Dagger" Type="Weapon" PriceValue="1"
        MinDamage="1" MaxDamage="3" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="0" />
    <Item Name="Dagger" Type="Weapon" PriceValue="1" MinDamage="1"
        MaxDamage="4" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="0" />
    <Item Name="Dirk" Type="Weapon" PriceValue="1" MinDamage="3"
        MaxDamage="6" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="0" />
    <Item Name="Short Sword" Type="Weapon" PriceValue="1"
        MinDamage="3" MaxDamage="8" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="0" />
    <Item Name="Khopesh" Type="Weapon" PriceValue="1"
        MinDamage="4" MaxDamage="9" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="0" />
    <Item Name="Sickle" Type="Weapon" PriceValue="1" MinDamage="1"
        MaxDamage="6" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="0" />
    <Item Name="Flint Axe" Type="Weapon" PriceValue="1"
        MinDamage="1" MaxDamage="5" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="2" />
    <Item Name="Hand Axe" Type="Weapon" PriceValue="1"
        MinDamage="3" MaxDamage="8" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="2" />
    <Item Name="Labris" Type="Weapon" PriceValue="1" MinDamage="3"
        MaxDamage="10" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="2" />
    <Item Name="Battle Axe" Type="Weapon" PriceValue="1"
        MinDamage="3" MaxDamage="8" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="2" />
    <Item Name="Staff" Type="Weapon" PriceValue="1" MinDamage="1"
        MaxDamage="6" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="1" />
    <Item Name="Club" Type="Weapon" PriceValue="1" MinDamage="1"
        MaxDamage="6" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="0" />
    <Item Name="Mace" Type="Weapon" PriceValue="1" MinDamage="2"
        MaxDamage="7" WeaponRange="0" WeaponReach="1" TwoHanded="0"
        WeaponType="0" />
    <Item Name="Spiked Club" Type="Weapon" PriceValue="1"
        MinDamage="4" MaxDamage="9" WeaponRange="0" WeaponReach="1"
        TwoHanded="0" WeaponType="0" />
    <Item Name="Flint Spear" Type="Weapon" PriceValue="1"
        MinDamage="1" MaxDamage="7" WeaponRange="0" WeaponReach="2"
        TwoHanded="0" WeaponType="1" />
    <Item Name="Bronze Spear" Type="Weapon" PriceValue="1"
        MinDamage="1" MaxDamage="8" WeaponRange="0" WeaponReach="2"
        TwoHanded="0" WeaponType="1" />
    <Item Name="Short Spear" Type="Weapon" PriceValue="1"
        MinDamage="3" MaxDamage="8" WeaponRange="0" WeaponReach="1"

```

```

        TwoHanded="0" WeaponType="1" />
<Item Name="Javelin" Type="Weapon" PriceValue="1"
    MinDamage="1" MaxDamage="6" WeaponRange="8" WeaponReach="1"
    TwoHanded="0" WeaponType="1" />
<Item Name="Sling" Type="Weapon" PriceValue="1" MinDamage="1"
    MaxDamage="4" WeaponRange="6" WeaponReach="0" TwoHanded="0"
    WeaponType="-1" />
<Item Name="Bow" Type="Weapon" PriceValue="1" MinDamage="1"
    MaxDamage="6" WeaponRange="10" WeaponReach="0"
    TwoHanded="1" WeaponType="-1" />
<Item Name="Composite Bow" Type="Weapon" PriceValue="1"
    MinDamage="3" MaxDamage="8" WeaponRange="14"
    WeaponReach="0" TwoHanded="1" WeaponType="-1" />
<Item Name="Whip" Type="Weapon" PriceValue="1" MinDamage="1"
    MaxDamage="6" WeaponRange="0" WeaponReach="2" TwoHanded="0"
    WeaponType="-1" />
<Item Name="Trident" Type="Weapon" PriceValue="1"
    MinDamage="3" MaxDamage="10" WeaponRange="0"
    WeaponReach="2" TwoHanded="1" WeaponType="1" />
<Item Name="Leather Vest" Type="BodyArmor" PriceValue="1"
    DamageResistance="6" ArmorIndex="0" />
<Item Name="Leather Boots" Type="Boots" PriceValue="1"
    DamageResistance="2" ArmorIndex="2" />
<Item Name="Bronze Vest" Type="BodyArmor" PriceValue="1"
    DamageResistance="8" ArmorIndex="0" />
<Item Name="Curiass" Type="BodyArmor" PriceValue="1"
    DamageResistance="10" ArmorIndex="0" />
<Item Name="Leather Cap" Type="Helmet" PriceValue="1"
    DamageResistance="1" ArmorIndex="1" />
<Item Name="Bronze Helmet" Type="Helmet" PriceValue="1"
    DamageResistance="3" ArmorIndex="1" />
<Item Name="Wooden Small Shield" Type="Shield" PriceValue="1"
    DamageResistance="1" ArmorIndex="3" />
<Item Name="Wooden Shield" Type="Shield" PriceValue="1"
    DamageResistance="3" ArmorIndex="3" />
<Item Name="Leather Shield" Type="Shield" PriceValue="1"
    DamageResistance="2" ArmorIndex="3" />
<Item Name="Bronze Buckler" Type="Shield" PriceValue="1"
    DamageResistance="3" ArmorIndex="3" />
<Item Name="Bronze Shield" Type="Shield" PriceValue="1"
    DamageResistance="4" ArmorIndex="3" />
<Item Name="Large Bronze Shield" Type="Shield" PriceValue="1"
    DamageResistance="6" ArmorIndex="3" />
</ItemList>
</Items>

```

Code: D.2: ItemList.xml file

D.2.3 CharacterList.xml

CharacterList.xml file used by Character module. This module handles all the character data in the game.

The root element uses the "CharacterList" tag. Every character will be defined as its child elements with a "Character" tag.

Each "Character" element has "Stats", "Skills" and "Equipments" elements as its child elements. They list character stats, skills and the equipment that character puts on as their child elements. "Stat" and "Skill" tags used as child elements to represent the characters' primary stats and skills. They have two attributes. The first one is the "id" attribute which defines the index number of it and the second one is the "value" attribute which numerical value of it. "Equipment" tag is used for the child elements of the "Equipments" element to represent each item equipped by that character. It uses the same two attributes of "id" and "value" for defining which equipment slot it represents and the index number of the item that slot uses.

```
<CharacterList>
  <Character name="Maruader" level="1">
    <Stats>
      <Stat id="0" name="Strength" value="6" />
      <Stat id="1" name="Agility" value="7" />
      <Stat id="2" name="Endurance" value="8" />
      <Stat id="3" name="Intelligence" value="4" />
      <Stat id="4" name="Willpower" value="5" />
    </Stats>
    <Skills>
      <Skill id="0" name="Melee" value="44" />
      <Skill id="1" name="Ranged" value="20" />
      <Skill id="2" name="Dodge" value="40" />
      <Skill id="3" name="Shield" value="26" />
      <Skill id="4" name="Mysticism" value="18" />
      <Skill id="5" name="Crafting" value="16" />
      <Skill id="6" name="Herbalism" value="12" />
      <Skill id="7" name="Leadership" value="14" />
      <Skill id="8" name="Trading" value="17" />
      <Skill id="9" name="Healing" value="11" />
      <Skill id="10" name="Persuasion" value="13" />
      <Skill id="11" name="NatureLore" value="24" />
    </Skills>
    <Equipments>
      <Equipment id="0" value="-1" />
      <Equipment id="1" value="27" />
      <Equipment id="2" value="24" />
      <Equipment id="3" value="3" />
      <Equipment id="4" value="-1" />
    </Equipments>
  </Character>
  <Character name="Hittite Soldier" level="3">
    <Stats>
      <Stat id="0" name="Strength" value="8" />
      <Stat id="1" name="Agility" value="6" />
      <Stat id="2" name="Endurance" value="7" />
      <Stat id="3" name="Intelligence" value="4" />
      <Stat id="4" name="Willpower" value="5" />
    </Stats>
    <Skills>
      <Skill id="0" name="Melee" value="60" />
    </Skills>
  </Character>
</CharacterList>
```

```

    <Skill id="1" name="Ranged" value="20" />
    <Skill id="2" name="Dodge" value="18" />
    <Skill id="3" name="Shield" value="48" />
    <Skill id="4" name="Mysticism" value="18" />
    <Skill id="5" name="Crafting" value="16" />
    <Skill id="6" name="Herbalism" value="12" />
    <Skill id="7" name="Leadership" value="14" />
    <Skill id="8" name="Trading" value="17" />
    <Skill id="9" name="Healing" value="10" />
    <Skill id="10" name="Persuasion" value="13" />
    <Skill id="11" name="NatureLore" value="22" />
  </Skills>
  <Equipments>
    <Equipment id="0" value="-1" />
    <Equipment id="1" value="27" />
    <Equipment id="2" value="-1" />
    <Equipment id="3" value="15" />
    <Equipment id="4" value="31" />
  </Equipments>
</Character>
<Character name="Kemet Soldier" level="6">
  <Stats>
    <Stat id="0" name="Strength" value="10" />
    <Stat id="1" name="Agility" value="6" />
    <Stat id="2" name="Endurance" value="8" />
    <Stat id="3" name="Intelligence" value="3" />
    <Stat id="4" name="Willpower" value="3" />
  </Stats>
  <Skills>
    <Skill id="0" name="Melee" value="61" />
    <Skill id="1" name="Ranged" value="22" />
    <Skill id="2" name="Dodge" value="52" />
    <Skill id="3" name="Shield" value="32" />
    <Skill id="4" name="Mysticism" value="12" />
    <Skill id="5" name="Crafting" value="12" />
    <Skill id="6" name="Herbalism" value="9" />
    <Skill id="7" name="Leadership" value="9" />
    <Skill id="8" name="Trading" value="12" />
    <Skill id="9" name="Healing" value="9" />
    <Skill id="10" name="Persuasion" value="9" />
    <Skill id="11" name="NatureLore" value="20" />
  </Skills>
  <Equipments>
    <Equipment id="0" value="-1" />
    <Equipment id="1" value="27" />
    <Equipment id="2" value="24" />
    <Equipment id="3" value="8" />
    <Equipment id="4" value="-1" />
  </Equipments>
</Character>
<Character name="Zannanza" level="7">
  <Stats>
    <Stat id="0" name="Strength" value="7" />
    <Stat id="1" name="Agility" value="8" />
    <Stat id="2" name="Endurance" value="6" />

```

```

        <Stat id="3" name="Intelligence" value="5" />
        <Stat id="4" name="Willpower" value="4" />
    </Stats>
    <Skills>
        <Skill id="0" name="Melee" value="77" />
        <Skill id="1" name="Ranged" value="23" />
        <Skill id="2" name="Dodge" value="64" />
        <Skill id="3" name="Shield" value="30" />
        <Skill id="4" name="Mysticism" value="18" />
        <Skill id="5" name="Crafting" value="20" />
        <Skill id="6" name="Herbalism" value="15" />
        <Skill id="7" name="Leadership" value="13" />
        <Skill id="8" name="Trading" value="19" />
        <Skill id="9" name="Healing" value="13" />
        <Skill id="10" name="Persuasion" value="14" />
        <Skill id="11" name="NatureLore" value="23" />
    </Skills>
    <Equipments>
        <Equipment id="0" value="23" />
        <Equipment id="1" value="27" />
        <Equipment id="2" value="24" />
        <Equipment id="3" value="4" />
        <Equipment id="4" value="-1" />
    </Equipments>
</Character>
</CharacterList>

```

Code: D.3: CharacterList.xml file

D.2.4 EncounterList.xml

EncounterList.xml file is used by the Combat module. In this file all the pre-determined combat encounters are detailed.

The root element uses the "Encounters" tag. It has child elements that use the "Encounter" tag. Each of these child elements represents a combat encounter in the game. These elements have one attribute called "name" to define themselves. They have several child elements that defining the information about that encounter. Elements with the tags "WinState" and "LoseState" defines how the game will continue after the outcome of that encounter. The attribute "moduleName" defines the module that the game will continue with and the attribute "value" provides the value that the module will use. The child elements with "Character" tag is to define the enemies the player will face in that encounter with the attribute "id" represent their index number in the character list.

```

<Encounters>
    <Encounter name="girl_combat">
        <WinState moduleName="Dialogue"
            value="AttackForJusticeWin"/>
        <LoseState moduleName="Dialogue" value="DeadAfterAttack"/>
        <Character id="0" />
    </Encounter>
</Encounters>

```

```

        <Character id="0" />
    <Character id="0" />
        <Character id="0" />
    </Encounter>

    <Encounter name="first_attack">
        <WinState moduleName="Dialogue" value="AfterAttack"/>
        <LoseState moduleName="Dialogue" value="DeadAfterAttack"/>
        <Character id="2" />
    <Character id="2" />
        <Character id="2" />
        <Character id="2" />
    </Encounter>

    <Encounter name="second_attack">
        <WinState moduleName="Dialogue" value="WinSomehow"/>
        <LoseState moduleName="Dialogue" value="DeadAfterAttack"/>
        <Character id="2" />
    <Character id="2" />
    <Character id="2" />
        <Character id="2" />
    <Character id="2" />
    <Character id="2" />
        <Character id="2" />
    </Encounter>
</Encounters>

```

Code: D.4: EncounterList.xml file

D.2.5 MapList.xml

MapList.xml file used by the Travel module. It lists all the map files that will be used by that module.

The root element uses the "MapList" tag and its child elements use the "Map" tag. Each child element has two attributes. The first one is "filename" and it defines the filename that stores the map data and the second one is "name" that states the name of that map for developers to access the map in the game.

```

<MapList>
    <Map filename="SampleMap.xml" name="worldmap" />
</MapList>

```

Code: D.5: MapList.xml file

SampleMap.xml file is a sample map file that can be used in the game by the Travel module. It is created in the Tiled map editor.

The root element uses the "Map" tag. It has several child elements. The first one uses the "MapInfo" tag. It defines the image file to the map, size of the map and the players starting position in that map with its attributes.

The child element that uses the "Grid" tag is to define the pathfinding data. The map is divided into a grid which its size defined as its attributes. For each grid cell, there is a numerical value that represents how long does it takes to travel over that cell. Value 0 means that cell is unwalkable and bigger the numerical value it is harder to walk over that cell.

Besides that, there is a "Nodes" tag with its child elements that use "Node" tags which represent each interactable map node. It uses several attributes to define the data of the map node. The position of the node is defined in "x" and "y" attributes. The image asset of the node is defined by the "iconName" attribute. The display name of the node is defined in the "name" attribute. Interaction type is defined by the "actionType" attribute and the value for that interaction is defined in the "actionValue" attribute. The visibility of the node is defined by the "State" attribute.

[illegible]

[illegible]

</Map>

Code: D.6: SampleMap.xml file

D.2.6 SceneList.xml

SceneList.xml file is used by the Travel3d module. It lists all the scenes that can be used by the Travel3d module.

The root element uses the "SceneList" tag. Its child elements use a "Scene" tag. Each of them represents a scene for the Travel3d module. It has two attributes. First, one is "name" and used for accessing the scene in the game. The second one is "prefabName" and used for defining the Unity prefab for the scene.

```
<SceneList>
  <Scene name="worldmap" prefabName="TestTerrain" />
</SceneList>
```

Code: D.7: SceneList.xml file

D.2.7 Yarn File for Dialogue Module

Dialogue module uses a file created by Yarn Editor. It is easy to use, WYSIWYG node-based editor. Several commands added for developers to use them in their yarn files. They were listed in Appendix B. For detailed information developers could consult to Yarn Spinners website.

D.3 Creating a New Module

Adding new modules is one of the most powerful features of this framework. How to do it is not that complicated as well. Every module in the framework prepared as a Unity prefab to be instantiated and managed by GameManager class. Every module manager class is inherited the BaseModule class as its basis. BaseModule brings four methods to implement for the new module's manager class. The initialize module takes one parameter for the filename of the XML file the new module is going to use for its data. Pause and Resume methods used for transition between the modules. Quit method is for safely quitting the game. Besides these four methods, developers can design their modules however they want the only caveat is module system is designed as there will be only one instance of each module, so making new modules manager class a singleton might be a good idea.

Keeping UI elements and a camera gameobject for the module as child elements of the prefab is another need. This might create an overload for manag-

ing different cameras every time a change needed but still enables developers to handle each module separately.

The last thing needed is making necessary changes to the GameManager class to recognize a new module during load, adding new modules prefab to the ModulePrefabs list in the GameManager prefab and adding a new module into GameData.xml file.

