University Institute of Lisbon

Department of Information Science and Technology

A Framework for Branched Storytelling and Matchmaking in Multiplayer Games

Vitor Manuel Januário Lopes Pêgas

A Dissertation presented in partial fulfillment of the Requirements for the Degree of

Master in Computer Engineering

Supervisor

Prof. Dr. Pedro Figueiredo Santana, Assistant Professor

ISCTE-IUL

Co-Supervisor

Prof. Dr. Pedro Faria Lopes, Assistant Professor

ISCTE-IUL

September 2018

Resumo

Os jogos de computador são geralmente conhecidos pelas suas experiências de jogo individuais ou multijogador. Nos jogos de um jogador apenas, a história tende a ser excepcionalmente bem escrita. Em contrapartida, o ponto forte dos jogos online é a interacção entre humanos. A solução para a ligação destes dois tipos de jogos é inexistente. Esta dissertação introduz uma estrutura que pode ser usada em toda a fase de desenvolvimento do jogo, desde os desginers aos engenheiros, e que ajuda a criação de campanhas multijogador ao apresentar vários componentes indispensáveis à criação de um jogo com boa história. Com a nossa estrutura, propomos também um sistema de matchmaking capaz de cruzar as diferentes histórias individuais e uma ferramenta para os designers poderem criar, gerir e partilhar o seu trabalho. A estrutura foi validada com um caso de estudo e testada num ambiente controlado. O sistema de matchmaking foi sujeito a diversas simulações e comparado com sistemas usados atualmente. A estrutura e a ferramenta de gestão tiveram resultados positivos, e o sistema de matchmaking equiparou-se com as soluções atuais mas distingue-se pela qualidade do match.

Palavras-chave: Jogos de video, Multijogador, Jogador Individual, Campanha, Estrutura.

Abstract

Computer video games are frequently known for either their single-player or multiplayer experiences. In single-player games the story is exceptionally well written. The interaction against other humans in multiplayer games is the strong point in online games. This dissertation introduces a generic framework aimed at the full game development pipeline, from designers to engineers, to aid the creation of multiplayer campaign stories by providing core components essential to any single-player game. With the framework, we propose a custom matchmaking system to intertwine individual stories and a tool for designers to create, manage and share their work. The framework was validated in a case study and tested in a controlled environment. The matchmaking system was subject to simulations and compared with existing systems. The framework and managing tool results are positive, and our proposed matchmaking system shows close efficiency results but distinguishes itself in terms of matching quality.

Keywords: Video Games, Multiplayer, Single-player, Campaign, Framework.

Acknowledgements

I would like to acknowledge Professor Pedro Santana and Professor Pedro Faria Lopes, that not only had superb availability and patience towards me and my work on this thesis, but also are two of the best teachers I ever had the pleasure to have. Also, important to mention, is the help of other exceptional Professors at ISCTE-IUL, like Pedro Sebastião which since we met gave me a lot of opportunities to develop myself as a person and as a professional, through events that I never would have participated without his support, Professors Luís Miguel Botelho, Luís Nunes, Tomás Brandão and Sancho Oliveira, for being without any doubt the best that ISCTE-IUL has to offer, both in teaching and personality. To them I want to say Thank you for being who you are and doing what you do. I want to also thank my friends that supported me with constructive criticism that ultimately helped me improve my work. Finally, I want to say Thank you to those who doubted I would reach this far, for giving me that extra moral push.

Contents

List of Figures

Abbreviations

AI	**A**rtificial **I**ntelligence
SP	**S**ingle **P**layer
MP	**M**ulti **P**layer
CCU	**C**on**C**urrent **U**sers
MOBA	**M**ultiplayer **O**nline **B**attle **A**rena
FPS	**F**irst **P**erson **S**hooter
MAU	**M**onthly **A**ctive **U**sers
NPC	**N**on **P**layer-**C**haracter
GUI	**G**raphical **U**ser **I**nterface
I/O	**I**nput **O**utput
GDD	**G**ame **D**esign **D**ocument
PCG	**P**rocedural **C**ontent **G**eneration
MMS	**M**atch**M**aking **S**erver
MMR	**M**atch**M**aking **R**ating
UI	**U**ser **I**nterface
BF1	**B**attle**F**ield 1
IP	**I**nternet **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
RPG	**R**ole **P**laying **Game**
AJAX	**A**synchronous **J**avascript **A**nd **X**ML
UX	**U**ser E**X**perience

Chapter 1

Introduction

1.1 Demographics

Nowadays, according to [Poulter, 2009], children as low as six years old frequently play video games. Intuitively, the more technology gets into our daily lives, the earlier children will begin playing video games. According to a recent study by the Entertainment Software Association (see Figure 1.1), 57% of players are younger than 35 years old and the most frequent ones play around 7 hours per week, and the rise of smart-phone games raises even higher the hours spent playing.

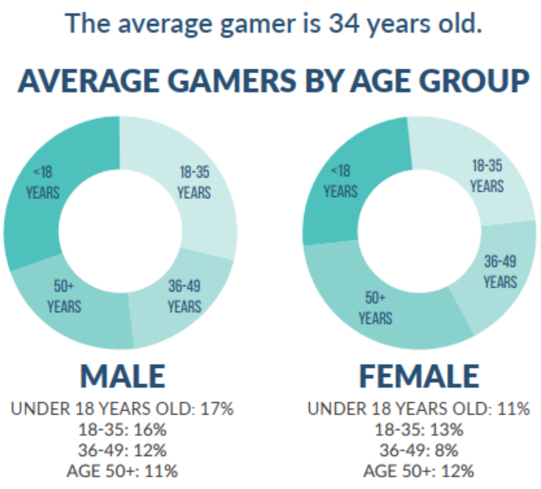


Figure 1.1: Study conducted by the Entertainment Software Association. We can see that male population is generally younger, while females are underrepresented on the early adulthood (36-49).

1.2 Why people play

People play video-games for various reasons. Young-lings play for entertainment, some do it professionally such as streamers, e-sports players play for competition and most people see games as a medium to relax and enjoy some leisure time. According to Lazzaro, people "value the sensations from doing new things such as dirt-bike racing or flying, that they otherwise lack the skills, resources, or social permission to do." [Lazzaro, 2004]. In games like Grand Theft Auto, players are meant to take a life of crime and wrongdoing, and even though in the real world this is considered bad behavior this is justifiable according to Lazzaro since players use games to "calm down after a hard day, or build self-esteem" [Lazzaro, 2004].

1.3 What people play

As Internet quality of service and speeds increase, more players will join the online world. Daily, on just one platform (Steam), in-game players peak at seventeen million (by December 2017) [Steam, 2018] (see Figure 1.2).

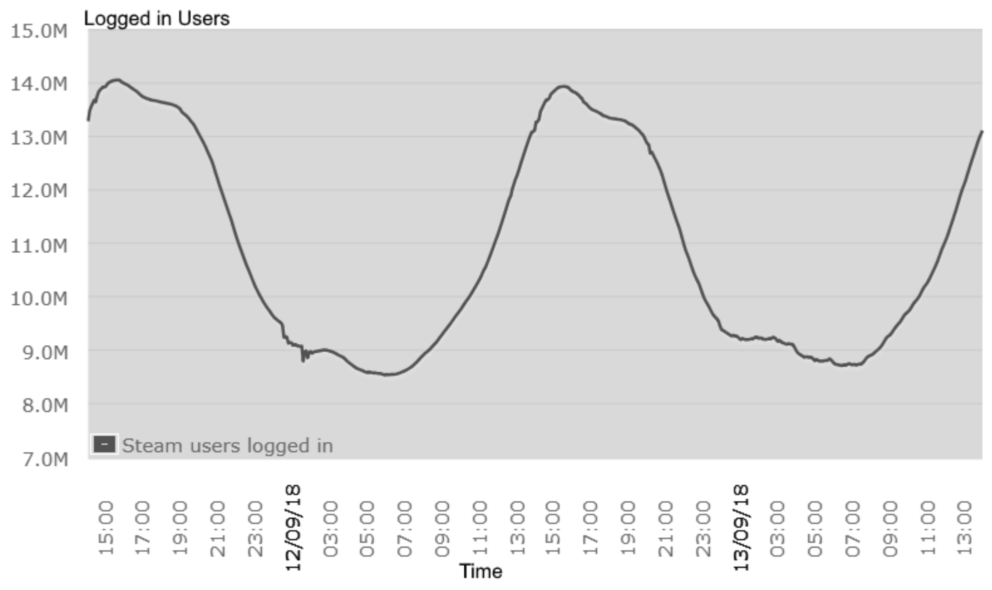
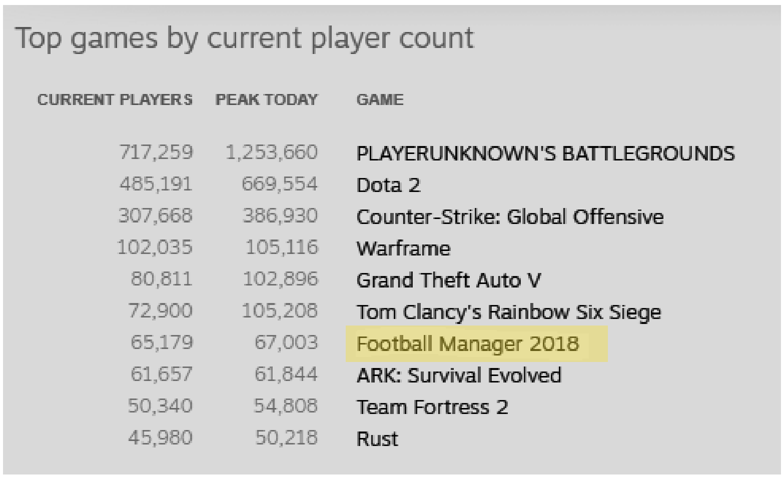


FIGURE 1.2: Steam Users Online in the last 48h showing a peak of 15 million users. [Steam, 2018]

In the TOP 10 most played games (see Figure 1.3) on Steam, the most played game is PLAYERUNKNOWN'S BATTLEGROUNDS, a Multiplayer Battle Royale themed game (a game where 100 players face each other and only one can survive), with more than two million concurrent users (CCU), followed by DOTA 2, a Multiplayer Online Battle Arena (MOBA, a game where teams of five players try to destroy each others base) with 600,000 CCU, followed by the all-time popular First Person Shooter (FPS), Counter-Strike Global Offensive with more than 500,000 playing simultaneously.



Top games by current player count

CURRENT PLAYERS	PEAK TODAY	GAME
717,259	1,253,660	PLAYERUNKNOWN'S BATTLEGROUNDS
485,191	669,554	Dota 2
307,668	386,930	Counter-Strike: Global Offensive
102,035	105,116	Warframe
80,811	102,896	Grand Theft Auto V
72,900	105,208	Tom Clancy's Rainbow Six Siege
65,179	67,003	Football Manager 2018
61,657	61,844	ARK: Survival Evolved
50,340	54,808	Team Fortress 2
45,980	50,218	Rust

FIGURE 1.3: Top 10 most played games on Steam. In yellow the only SP game in the TOP 10. [Steam, 2018]

On this TOP 10, only one game is mostly single player, Football Manager 2018, a popular sports team management simulation game. Outside of the Steam platform, are two multiplayer giants, Overwatch and League of Legends. Although Overwatch, a mix of MOBA and FPS, does not release its player count often, it was reported that it passed 30 million players [Barrett, 2017]. League of Legends, also a MOBA like DOTA 2, does not have a public player count either, but one of its creators admitted that the game had reached 100 million Monthly Active Users (MAU) [Volk, 2016]. These numbers show a portion of the reality of todays video game industry, especially multiplayer games.

1.4 Context

Currently, there are two main types of commercial video games, Single-Player (SP) and MultiPlayer (MP) games. In SP games, a player plays against Non-Player Character (NPC) controlled by Artificial Intelligence (AI), whereas in MP games players can play against each other (Human vs Human) or in cooperative modes (Humans vs NPC).

In 2017 there were five SP games in the top ten sales, with millions of copies sold worldwide [VGChartz, 2017]. This shows that although MP games dominate the market in revenue, SP games continue to be bought and played by millions. On the other hand, MP games are not only strong in sales, but the revenue of such games surpasses SP revenue, mainly due to micro-transactions. Micro-transactions consists of selling in-game items (virtual items) for real money. It began by being small in value, hence the name, but on the Steam market (the most popular and biggest in-game item market) items can go as high as 2000 USD (see Figure 1.4). For instance, games like Grand Theft Auto Online (GTA V Online) and League of Legends (LoL) made more than 500 Million USD [Tassi, 2016] and 2.1 Billion USD [Volk, 2016] respectively in in-game item sales alone. A problem that SP games may suffer is that their quality is closely linked to the quality of the AI controlling the Non-Playable Characters (NPC). An improper AI may render a game either too easy or too hard to overcome. On the other hand, players in MP games may find themselves immersed in what is known as toxic communities, in which players engage on hostile behavior against each other, a phenomenon known to naturally occur in competitive gaming scenarios, such as MP online games [Märtens et al., 2015]. Popular online MP games such as Call of Duty and League of Legends are well known for their community toxicity [Kwak and Blackburn, 2014]. Since both genres have their pros and cons, establishing a link between single player stories and multiplayer interaction could create a overall better game experience. To overcome these limitations we propose a generic framework agnostic to the game genre in order to create campaigns with multiplayer interaction that prevent the use of bad AI controlled characters and

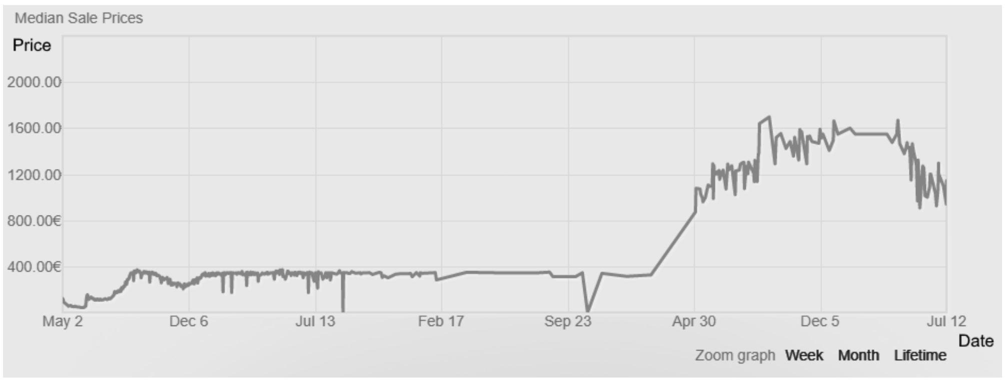


Figure 1.4: Median Sale Price of a micro-transactions item for CS:GO with that item being sold for over 400€.

a matchmaking system that attempts to provide a more suited matching between players by allowing the usage of multiple criterions.

1.5 Research Questions

If we analyze both genres problems, we could come to a middle term where developers offer the best of both worlds to players. This dissertation aims at the creation of that missing link to aid the development of multiplayer campaigns.

- Is it possible to connect immersive and rich in design single-player stories with multiplayer interaction? - Our framework should be able to link single player stories with multiplayer interaction seamlessly. This was made possible with the use of core generic components key to every story driven game.

- Can a matchmaking system be more accurate with more than one matching criterion? - Current matchmaking systems use one criterion for matching, the skill rating for each player. Our proposed matchmaking system can use more than one criterion. Our results show that current solutions do not guarantee balanced matches, whereas our proposed system matches players in a more coherent and balanced way.

- Can a framework be created to generalize both game design and implementation of such features? - We will systematize the link between single player and multiplayer stories with a standard framework that can be used for multiple game types and genres. Our framework is game genre agnostic, meaning that it can be used for a multitude story-driven multiplayer game.

1.6 Objectives

Our main objective is to conceptualize, design, create, and implement a framework to aid the development of MP campaigns. Our framework is supposed to be used in every stage of the game development pipeline, from conceptualization, to artistic and engineering development then used by designers, artists, and programmers. We then propose several algorithms to be used alongside the framework for the matchmaking system. For story management we propose a Graphical User Interface (GUI) application that can be used by designers on initial development stages and then by programmers in the implementation phase. Finally, we test the framework on different aspects like implementation, usefulness, flexibility, and if it works to deliver a multiplayer campaign experience.

1.7 Structure

This dissertation is structured as follows:

- **Chapter 2** introduces and explains what has been made regarding Game Design, SP and MP campaigns with story crossing, branched story telling and existing matchmaking systems.
- **Chapter 3** we define the framework and its components, our proposed matchmaking system and our developed Designers GUI. We also introduce our case study where we tested our framework implementation and matchmaking system.

- **Chapter 4** presents our testing methodology and results for implementation, story generation, matchmaking and our GUI application.
- **Chapter 5** exposes our conclusions with this work and possible future work that could be developed on top of this dissertation.

Chapter 2

Related Work

2.1 Game Design

According to Schell [Schell, 2014], "Game design is the act of deciding what a game should be" with the designer(s) having to make "hundreds, usually thousands" of decisions about the way the game will play out and simply be. There is no magic or unified framework or formula to design a game, but Hunicke et al. [Hunicke et al., 2004] developed a framework to ease the process for "developers, scholars and researchers" to develop their work related to game design and gaming principles. Hunicke et al [Hunicke et al., 2004] break games in three components: Rules, System and Fun, and from a design point of view this translates to Mechanics, Dynamics and Aesthetics (MDA) (see Figure 2.1). A Game Designer must start by defining the rules or mechanics of its game which represent the core components of the game. Then the designer should focus on the player interaction with the game, how the game will react to player input and subsequently to its outputs. Games should be interactive so this is a key part of the gaming experience for both players and developers. Finally the designer worries about Aesthetics, the look and feel of the game, i.e how players should feel while playing.

On other hand, Crawford [Crawford, 1984] defines three main structures inherent to game design: I/O, game, and program structures. Compared to the

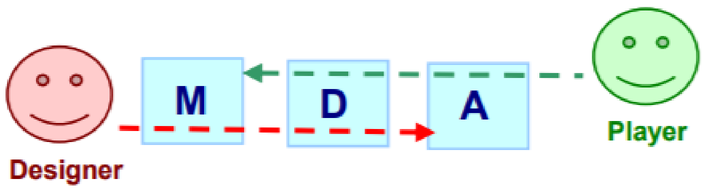


FIGURE 2.1: Different perspectives from the MDA framework [Hunicke et al., 2004]

MDA framework, the game structure can be compared with MDA's Rules/Mechanics, the I/O structure can be compared to MDA's System which translates to the interaction between the game and the player and, finally, the program structure represents the later development phase of implementation which takes both previous structures and turn them into a real product.

2.1.1 Game Design Document

The Game Design Document (GDD) is probably the only standard in the gaming industry [Bertolo, 2014][Tomlison, 2013] being used widely by big or small companies in the design stage of game development [Rouse and Illustrator-Ogden, 2000]. The GDD is a document where the designer writes all the concepts of a game such as the game's specification (Story, Players, Action, Objectives)[Inc., 1994], game's Gameplay (World, Landscapes, NPCs)[Hamilton, 1995] and is always being analyzed and improved with feedback from other team individuals. When the GDD is finished it is passed onto producers, programmers and artists so that the game can be created exactly like specified on the document. This document is used for both game and technical related information regarding the game being created, however our framework aims to streamline the game's story related design and implementation process.

2.2 Single Player and Multi-player Crossing

A player engagement study [Lim and Reeves, 2010] found that players show higher engagement levels when facing humans in a competitive scenario instead of A.I controlled characters. This may also be influenced by the fact that NPCs quite often produce either too easy or too hard gameplay scenarios. Nevertheless, players do not refrain from playing and enjoying SP games as this type of games often features a well written story (e.g, Until Dawn [Brew, 2015]). For this reason, our framework aims at allowing players to enjoy the flow of a storyline while playing in a MP setting, that is, making use of NPCs when no online players are available.

To improve the lack of engagement or challenge imposed by NPCs, game developers blurred the line between SP and MP by having the SP game with its well written and designed story and levels and with occasional MP interactions. Dark Souls is a good example of the crossing between a players' individual story and the MP interaction with another human player. In this game there are certain missions where a player can be invaded by another player, either friendly or enemy. That second player will replace a NPC and participate in that mission alongside the first player. In Journey video-game [ThatGameCompany, 2012], each player can be coupled with another human at random and both players cannot speak to one another, but they can explore and progress in the world together, or part ways. Ashen video-game [Grubb, 2017] replicates this sort of passive multiplayer by inserting strangers in player sessions at random with the main goal of having spontaneous cooperation between players. After a player leaves a session, one's avatar remains in the world as a NPC for future interactions. In Absolver video-game [DIGITAL, 2017], each player game instance can also host other players which will join the first one seamlessly through the game. It is up to each player to either fight or cooperate with one another. The above examples are games in which there is a sense of individual story with multiplayer interaction. However, there are other games that expressly create linear multiplayer campaigns such as Left 4 Dead or Borderlands, in which player control specific characters with predefined back stories through a set of cooperative missions that try to create a sense of narrative for each player.

These games are not open sourced, meaning that it is not possible to know how these systems were implemented or designed. There is no standard for such features and systems and a framework is needed to assimilate the essential concepts for multiplayer campaign creation in video games. Our framework systematizes the core components of multiplayer campaigns to solve the lack of uniformity in the industry.

2.3 Branched Storytelling and Narrative

Replay value is a term used in video game industry that states whether a game is good to play more than once. Usually, due to their linear design, SP games do not hold much replay value. This means that if players complete the story and then try it again, the story (and ending) will be the same. However, there are SP games that try to branch their narrative influenced by player input, thus increasing their replay value. Bearing replay value in mind, our framework handles both linear and non-linear stories.

MP games are quite often repetitive. For instance, First Person Shooters (FPS) usually only have as goals killing the opposite team, destroying a given objective, and capturing a given item or position. In spite of this, players keep playing FPS motivated by [Yee, 2006]: competition, as seen in games like DOTA 2 and CS: GO; socialization, as seen in games like Second Life and Habbo Hotel; and role-playing as seen in games like GTA V, and World of Warcraft.

In SP games, developers must find a way of showing the player that their actions matter and can change the outcome of the game, pushing the player to play the same game several times while producing different outcomes [Roth et al., 2012]. Procedural Content Generation (PCG) is a popular choice to provide the so required in-game diversity (e.g, random levels, fauna, flora). PCG use has increased lately due to the skyrocketing production cost of AAA (triple A, big budget games)

games [McLaughlin, 2013], for which it is difficult to scale up asset production (e.g: 3D models or textures) [Hendrikx et al., 2013].

As surveyed by [Hendrikx et al., 2013], PCG can be divided into six classes shaped in a pyramid form where top classes are derived and based on bottom ones. Every class can be procedurally generated. Firstly they define what they call as Game Bits, which are elements which typically do not engage the user when considered independently [Hendrikx et al., 2013]. Bits can be Textures, Sound, Vegetation, Buildings, Behavior and Particle systems. On top of Game Bits, there is the Game Space which includes Maps (the space where all gameplay happens) and Bodies of Water. On top of the Game Space is Game Systems and Scenarios. Systems derive from the Space, forming Ecosystems, road networks, urban environments and entity behavior, while Scenarios derive from these Systems forming Puzzles, storyboards, the Story and Levels. On top of Scenarios, the Game Design appears with System and World Designs that can also be procedurally generated. The final class is called Derived Content and it incorporates everything that will make the player feel immersed in the game world such as News and Broadcasts and Leaderboards.

These six classes represent the possibility for PCG in todays video games with many AAA using some of those classes in their development. Examples such as Spore (a PCG evolution game created by Electronic Arts) which used PCG for Game Bits and Game Space creation or Elder Scrolls V: Skyrim (an action RPG created by Bethesda Game Studios) which used PCG to generate side-quests for the player (class Game Scenarios). The biggest example of PCG use in a game is No Man's Sky (a universe exploration game created by Hello Games) which uses PCG to generate not only an entire Universe (featuring over 18 quintillion of randomly generated living planets) but all fauna, flora and minerals of the game.

One of the genres that mostly use PCG for content generation is called Rogue-like. Examples such as The Binding of Isaac and Spelunky use PCG to generate their dungeons. An example of this dungeon generator is Donjon's Random Dungeon generator which uses Celullar Automata. Celullar Automata consists of an array of cells with discrete variables which evolve based on the values of neighbor variables. [Wolfram, 1983]. This behavior allows for the creation of cave-like structures and with extra processing (such as Donjon which processes rooms, corridors and doorways) is used to create dungeons in most rogue-like games.

Other way to increase replay value is by branching the narrative in non-linear ways. As compared in Figure 2.2, Half-Life [Valve, 1998] has a standard game narrative comprised of Beginning, Middle and End nodes, with no changes based on player input, whereas Mass Effect's story is comprised of multiple nodes where player input steers the story flow one way or the other making alternative endings possible. For this to happen, there must be more than one choice of action for the player to perform, and that action must somehow impact the story unfolding. Many games have exploited this ability widely, such as Façade [Mateas and Stern, 2003] (see Figure 2.3), Witcher 3, Undertale, Heavy Rain, and Detroit Become Human. These games require some narrative mediation system to ensure the player experiences a coherent story, by creating a path that the player travels according to one choices and the system automatically re-writes the path if the player decides to take a different direction.

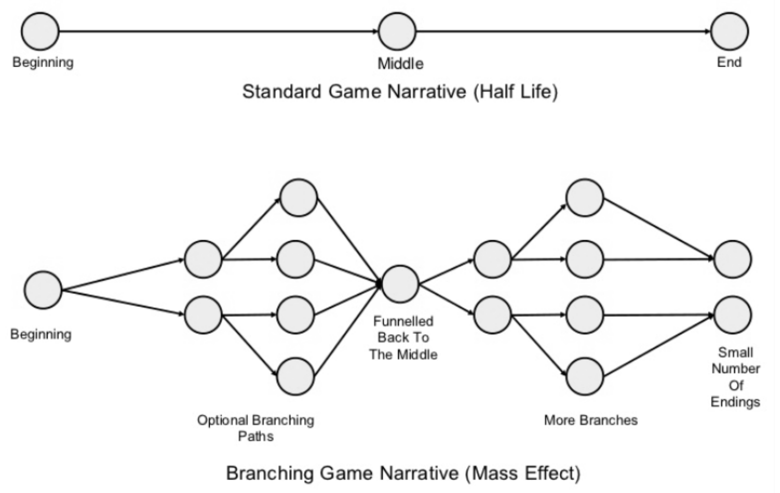

Beginning
Middle
End
Standard Game Narrative (Half Life)
Beginning
Optional Branching Paths
Funnelled Back To The Middle
More Branches
Small Number Of Endings
Branching Game Narrative (Mass Effect)

FIGURE 2.2: Story Branching in Half-Life and Mass Effect [Jack, 2011]. The difference between a linear and a non-linear game is the possibility of player choice in non-linear games (Like Mass Effect) where different actions create different paths.

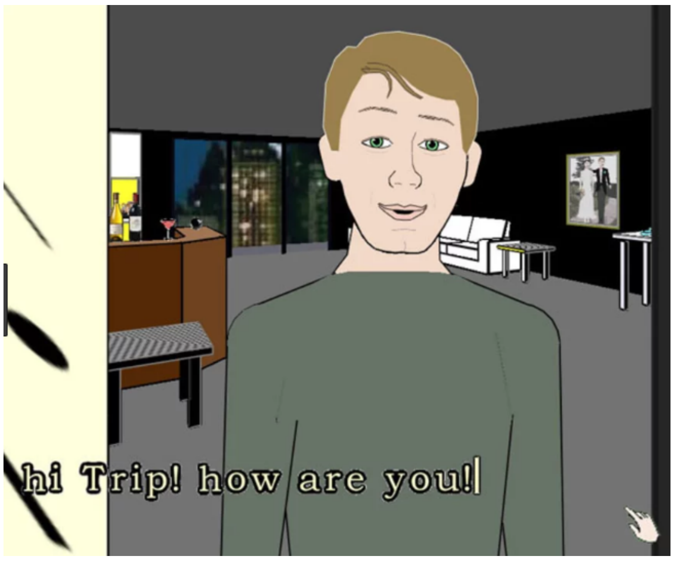


FIGURE 2.3: Façade Screenshot showing the interaction between the user and the NPC. The text present in the screenshot is written by the player to interact with the NPC.

A possible architectural plan for an interactive narrative is Mimesis that provides basis for the creation of interactive stories such as "conventional narrative media" [Young and Riedl, 2003]. Their work uses a controller called Mimesis Controller that acts as a Story server taking key role in the generation of a coherent story and experience for any user activity.

Drama managers like the Façade example (see Figure 2.4) are common in many interactive storytelling applications [Paul et al., 2009] [Paul et al., 2010] due to their ability to manage the unfolding story according to user input and are used to maintain a coherent story flow. The way Façade's Drama Manager works is by having a pool of around two hundred beats, which are a collection of behaviors that in sequence form a plot, and as the player interacts with the NPCs, beats are selected from the pool in a coherent way towards the desired value arc(s) thus forming a story flow.

Mist [Paul et al., 2010] is another system that uses a Drama Manager with various story elements and then assigns those elements to characters to develop a story. Those characters receive those elements and decompose them into smaller actions called primitives and then execute them. This has direct impact in the virtual world and can affect the player and vice-versa.

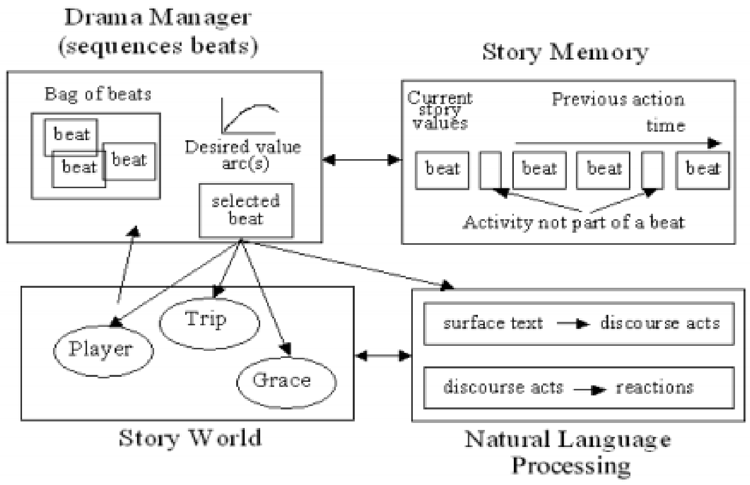


FIGURE 2.4: Façade Drama Manager. The story is composed of "Beats" and these are selected from a bag of possible beats according to the interaction between player and NPCs.

The common ground to all games with branched storytelling is the need for a Manager or Controller that has the power to create or change paths in a story. However there is not a standard on how this Controller should act, probably due to the amount of variation branched stories can have in different games.

2.4 Matchmaking

A matchmaking system is used in games to connect players in a given match. When a player wants to play online, its game instance (client) sends a request to the Matchmaking server, receiving a response with necessary information to establish a connection [Agarwal and Lorch, 2009]. A bad matchmaking service is known to be harmful to both skillful and casual players [Myślak and Deja, 2014].

One of the most popular types of matchmaking uses a numeric value, called a MatchMaking Rating (MMR), to match players according to their skill levels. The most popular method is the Elo system, created for chess competitions, in which the MMR is determined by the results of the player in past games

[Glickman and Jones, 1999]. Matchmaking usually picks players with close MMR values so as to ensure fair and, so, interesting, matches [Véron et al., 2014]. Match fairness and uniformity can be further enforced by ensuring that the players within a given team have an uniform MMR [Alman and McKay, 2017]. By using custom matchmakers, instead of general-purposed matchmaking systems as Elo, games can match players more accurately as a function of the game's mechanics.

Another known matchmaking algorithm is TrueSkill^{TM}. This system, developed by Microsoft and used in some of their online games uses a factor graph that uses player skill, performance, and the overall team performance to predict the outcome of a match[Herbrich et al., 2007]. In real tests with the game Halo 2, it showed good results when faced against the Elo system.

In some cases, the use of only one value for matchmaking can be lacking so the need to take in more data into account when matching players is real for some games, like Ghost Recon Online. In their work [Delalleau et al., 2012] they developed a Neural Network that uses player profiles (with embeddings and attributes) to predict the outcome of a match comprised of two teams. They compared their network to the TrueSkill algorithms with good results in outcome predictability.

2.5 Applicability

In this dissertation we will apply some of the knowledge of past literature and commercial applications. From a Game Design perspective, our framework should aid every game development pipeline agent by being able to be transferred seamlessly back and forth, similar to a Game Design Document, but exclusively for story related information.

Our work will also use similar ways of crossing players as seen in games like Absolver and Journey, where a player may initiate a story alone and be joined by other players that happen to cross that point in the campaign at the same time.

As game designers push the depth of their stories, managing and crossing all different types of players in a coherent and believable way may get complicated. Our work needs to assure that using PCG, a coherent story can be generated.

Finally, a multiplayer game where different players with different abilities and skill levels needs to be balanced. Elo will be paired with our framework, tested and compared to a custom solution in order to assure match quality and balance.

Chapter 3

The Framework

As identified in previous chapters, there is a gap in the industry for the creation of multiplayer campaign stories. Our goal is to eliminate this gap by creating a framework to offer a backbone structure for games to implement this link between SP and MP games by incorporating generic concepts that a multiplayer campaign game should have. We will provide an example on how the framework could be used in a released game on Section 3.2.

3.1 Framework Specification and Requirements

To be robust and generic, our framework has been arranged to be game genre independent. Shooters, Role playing, Strategy or other type of game that has a campaign story should be able to use our framework. Our framework is aimed at the full game development pipeline (see Fig. 3.1) and should be used by every participating member of this pipeline. By full game development pipeline, we mean every step along the development of a game, from the Concept Development phase where Game Designers pitch and formulate a basic idea of what the game will be, to the Design stage where Game Designers will detail the concept further by establishing how, where and when a game will be played out. After every idea is settled, the actual production begins with Artists and Engineers taking the ideas

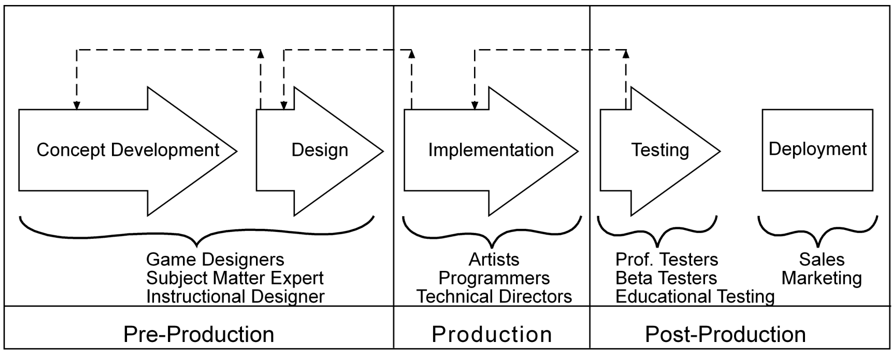


FIGURE 3.1: Full production pipeline for Game Development. Starting with the conceptualization of the idea with Designers and Experts, then the Design phase where the GDD is created, moving onto the Implementation part where the ideas come to reality and finally the testing and deployment phases.

defined by designers and Implement them into the game, like asset creation (including audio, 3D models, textures, sprites, etc.) and programming. After various cycles of implementation and idea reformulation (as Game Designers' ideas may not be final), the game enters its Testing phase to polish and enhance everything that was created. The final step of the pipeline is the Deployment of the game to the general public.

The MDA framework (see Section 2) defines those three steps of game design as Mechanics, Dynamics and Aesthetics. Our framework fits in the MDA framework definition, as it should be used to assist designers in the definition of Rules and Systems of their game (representing the Mechanics and Dynamics) while aesthetics, which are the emotional responses evoked in the player (not to be confused with graphical aesthetics which are the visual aesthetics of the game such as textures and models) should also be present in this definition, and should be described so that Artists and Engineers can create something close to the initial designer vision representing the Aesthetics of the MDA framework.

When the designer first pitches the idea and starts building a concept, he/she creates a Game Design Document (GDD) where the game is described as detailed as possible and where the idea starts to come to life. This document should contain all details regarding Story, Characters, Environment/Level, Gameplay, Art description, Sound, User Interface (UI) and Controls [Andrade, 2013]. When the GDD is well developed and the game idea/concept is ready for the next stage, the designer should translate the story related parts of the game design document onto our framework concepts. This translation will aid the middle and late stages of development as it will not be just abstract concepts written in a document, but actual components that together form a game. The designer could entirely drop GDD usage for story related definitions and use our Framework for this purpose, maintaining GDD for details such as platforms, monetization, and other non-story related details. The exclusive use of our Framework for story related work would facilitate the exchange and communication between different areas of the pipeline, such as Designers to Programmers or Designers to Artists.

The proposed framework should be easily implemented into the current engines (e.g. Unity, Unreal Engine) and should be modifiable for any specific and non-planned usage, by adding other components and connecting to current ones. As the framework components are conceptual, they can be implemented in a variety of programming languages and environments, while maintaining the backbone structure to aid the game development stage.

3.2 Framework Overview

The proposed framework is composed by several components such as nodes, player profiles, etc. (see more details in Section 3.3) that should be used, first by designers and then by programmers and artists, to define and create the game (see Figure 3.2). To introduce our framework, we will now present a practical example of a possible implementation of our framework on a published and acclaimed game, Battlefield 1 (BF1), which is a first person shooter with both SP and MP modes. Our framework is aimed at multiplayer campaigns with the possibility of story crossing and branching, and BF1 has only five isolated single player campaign stories. The SP mode has well written stories where players advance through the world completing simple objectives like capturing points, eliminating hostile enemies and so on. The MP is very repetitive in gameplay like the average multi-player shooter games, meaning that objectives and possible situations or outcomes do not change with player input or time. Our framework could be used to allow the SP campaign modes to have MP interaction, thus becoming more interesting due to the confrontation with real human players instead of artificially controlled enemies, while each player would develop one's own story.

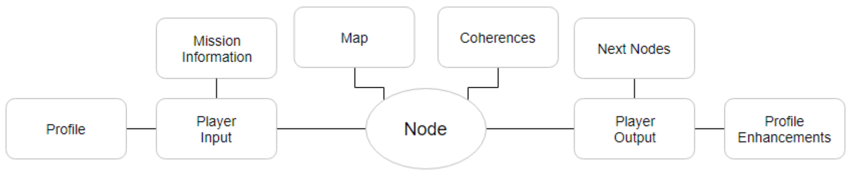


FIGURE 3.2: Framework Components Overview. The central component is called a Node, which has information about the Players that go into the Node and how they leave the Node, as information of the location of gameplay (Map) and that Node Coherence list.

As an example of the game, in one of the campaign stories, the player is a tank driver with the main goal of advancing through enemy lines along other NPC tanks and infantrymen while battling enemy NPCs. Our framework could be used to allow different roles in this campaign, allowing players to be either a tank operator (gunner, driver, commander etc.), an infantrymen or any other battle role, each with its own back stories leading to that particular story moment. In our framework each story moment (or level) is seen as a Node by our framework, and a player's story can be seen as a graph. In the aforementioned tank battle, one of the tank drivers could be there for any reason: maybe his last tank crew got wounded in a previous node and he was the only one capable of moving on to the next node or maybe he was assigned to that node as his first battle in the war. Our framework covers partial success as well, any mission can have multiple levels of success, each one leading to a specific node, coherent with that success value. If in the tank battle, the player was a tank operator and his tank was heavily damaged, the player could still move onto the next node as an infantryman, implicating a partial success. This way stories can be non-linear and more mesmerizing. Each node can handle various player roles (defined by designers), and the way to determine if a player fits any of those roles is to use players' descriptors (Stats in our framework) such as player health, level, experience, and any information defining that player.

Our framework was created to allow the possibility for players' individual stories to cross each other at story intersections. In those intersections (hereafter story nodes), players can be on the same or opposite sides. This is possible due to the existence of a list of player roles for each story node, and the framework uses this to allocate players to nodes in a coherent way. Additionally, a node-based stories approach allows the possibility of the materialization of multiple paths in a pool of nodes while giving designers complete power to design these nodes individually in as much detail as possible.

3.3 Components

This section describes in detail the core components of the proposed framework that should be used to define a multiplayer campaign story.

3.3.1 Node

The base of our framework is the Node and it represents a point in a campaign story. A Node contains all the information regarding that point in the story, which can be where multiple types of players cross paths and face each other. Each Node contains a Map, Player Input and Output lists (capable of storing multiple input and outputs) and Coherence lists. Recalling the aforementioned example, the battle where tanks and infantrymen faced each other is represented by a node. In that node, the player inputs are the slots that should be filled with players for each specific role using their Stats for matching. The player outputs contain information for progression in the story such as the next node. Coherence examples in that node could be the physical location of the battle or the appearance of the map.

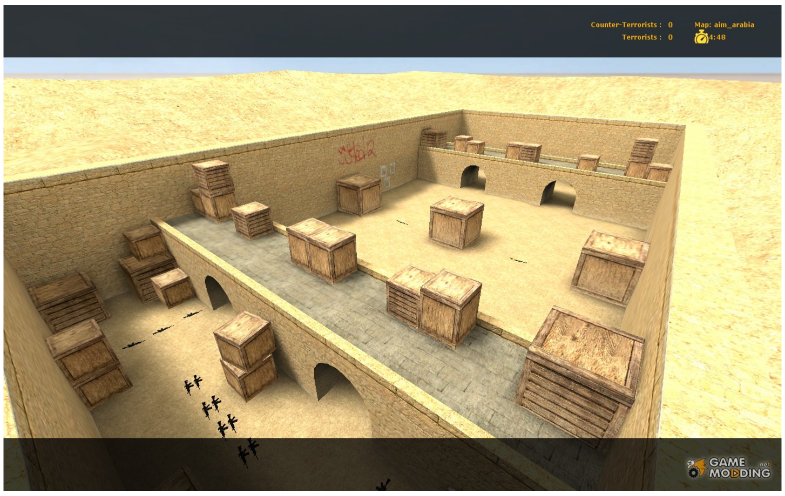


FIGURE 3.3: An Example of a Map in the Counter Strike Game. It contains all assets needed for that map to be played out such as walls, boxes, guns etc.

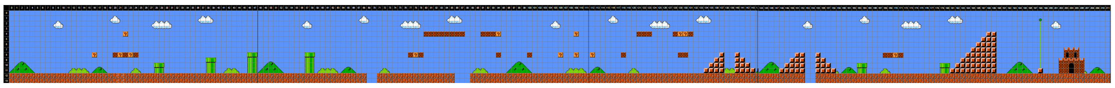


FIGURE 3.4: An Example of a Map in Super Mario Game. The player goes from the left to the right jumping over platforms, collecting coins and defeating the enemies until he/she reaches the end of the map.

3.3.1.1 Map

The map is the game engine's representation of game-play physical location (in the Unity Engine this is represented by a Scene object[1]). This is the 2D/3D space where the node is played out, containing all assets and game-play mechanics ready for players to use. In 3D we have the example of the popular game Counter Strike, where a map is the physical location where players face each other in combat (see Figure 3.3). In 2D we have the example of the classic Super Mario map where the player must jump through platforms to reach the end of the level (see Figure 3.4).

3.3.1.2 Player Input

The information that defines the type of players that will play that node and what they will do in that node is stored in the player input list. Each player

[1]Taken from Unity's Official Documentation: Scenes contain the environments and menus of your game. Think of each unique Scene file as a unique level. In each Scene, you place your environments, obstacles, and decorations, essentially designing and building your game in pieces. - https://docs.unity3d.com/Manual/CreatingScenes.html

input has information about the player profile that fits that Role (defined by the designer) and its mission information like Mission Name, Description, Role, Objective Boundness and Success. The role represents the role of the player in that mission (ex: In the aforementioned BF1 example, Tank Driver, Infantrymen etc). Objective Boundness represents the amount of times a player can die (i.e fail the mission) in the current node without actually failing and having to start the node over. For instance, a Objective Boundness of zero means that if the player dies, she/he fails and has to restart the node from the beginning, whereas with a value of three means that the player can die and re-spawn up to three times before failing the node. This setting is there to allow re-spawn mechanics as seen in many modern multiplayer games. Success is a value that should be used to keep track of the success in the current mission by that player, a value of zero means the player is failing in the mission and any value above that represents the success node and profile that the player will achieve in the Player Output. With the aforementioned BF1 example, the success for an infantryman would be the amount of enemy outposts captured, the value of zero would fail the node whereas a value of one or more would mean a success. Different success values above zero allow the branching of the story.

In the previous BF1 example, a possible Player Input for any tank gunner battle node could be:

Player Input #1

Profile:

Stat#1: Class

Value#1: Tank Gunner

Mission Information:

Mission Name: Destroy the Enemy Base Building

Mission Description: Use your tank and the help of your squad to destroy the enemy base building located in the city X.

Role: Tank Gunner

Objective Boundness: 0

Current Success: 0.

This player input would be filled with tank gunners that were joining that specific Node at that time. After the destruction of the enemy base building, this player input's current success value would increase to one.

3.3.1.3 Player Output

Associated with each player input in a given node, there is a player output. The Output contains information about how a player leaves the node after its completion (success or fail). Player output has a list of next nodes, which the game will compare with the current success of the respective player input and pick the node where to send the player next. It also includes a list of profile enhancements, such as experience or in-game currency, that will be picked in the same way as the next node to increase or decrease any player Stat. The list of next nodes in the typical game would have two nodes, a win node and a fail node. But with our framework, this list can be used to branch the story in a coherent way, with partial successes. An example for this is a mission of a fighter pilot in a war, in which the mission is to attack some targets in the ground. The player attacks some of the targets, but gets shot down by anti-aerial weapons. In the proposed framework, the player can be sent to another node where one's plane was shot down but he is still alive, hence the partial success of the previous node, whereas if the player attacks every target the success is maximum and the player is sent to a different node. These partial successes can be defined by designers in any way they see fit to the node in question, a node can have above two types of success. Profile enhancements can be used to manage player Stats after the Node is played. If the player completed the mission with 100% success, the player can be rewarded with some experience points or any other stat boost. It can also be used to deteriorate any player stats, in case the player failed a specific objective inside the mission.

Following the previous **Player Input** tank gunner example, this is an example on how that Player Output would look like:

Player Output #1

Next node List:

1. **Next node#0:**Tank Battle Node (failed, replay)
2. **Next node#1:**Infantry Battle (partial success, tank got destroyed, proceed on foot)
3. **Next node#2:**Rendez-vous point Node (success, move on)

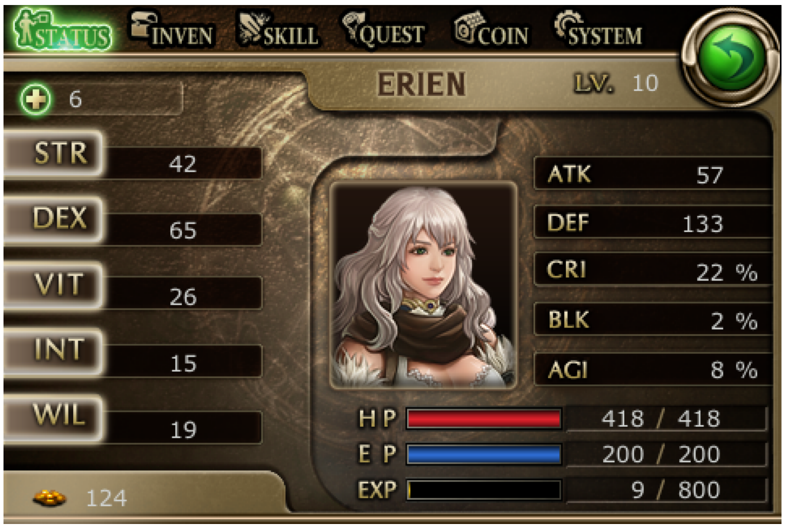
Profile Enhancements list:

1. **Enhancement#0:**None
2. **Enhancement#1:Stat:** Experience **Value**:+100
3. **Enhancement#2:Stat:** Experience **Value**:+300

As the previous example shows, the next node list has three nodes: the first one for $currentsuccess = 0$ (node failed), the second one for $currentsuccess = 1$ for partial node success, and the third one for $currentsuccess = 2$ for full node success. The same principle applies to the profile enhancements list: if the player fails the node, no profile changes are made, but if the player partially succeeds, then it's profile Stat *Experience* will gain a boost of 100 in value, whereas a full node success will reward the player with three hundred *Experience* points.

3.3.1.4 Profile

Associated with every player, is a Profile which is a set of Stats. Each Stat is a numeric value that is used to store player related values and match players in our matchmaking algorithm. Example uses of Stats are, Player Health, Experience, Ranking, or the last node they played on (see Fig.3.5). Any Stat can be inserted in a profile, and the matchmaking system will look for every common Stat between two profiles or previously chosen Stats by designers for the matchmaking (using weights). By assigning weights to each Stat, designers can give higher importance to a group of Stats and less importance to another group. Let us imagine a designer who wants to give 90% importance to the Ranking Stat, and 10% importance to the class of the player, thus matching players with different classes but in a close ranking position.



ERIEN LV. 10
6
STR 42
DEX 65
VIT 26
INT 15
WIL 19
124
ATK 57
DEF 133
CRI 22 %
BLK 2 %
AGI 8 %
H P 418 / 418
E P 200 / 200
EXP 9 / 800

FIGURE 3.5: An Example of player stats. This example has 15 possible stats such as: STR (Strength), INT (Intelligence), and Gold. Image from Fantastic Knight (iOS/Universal)[Minoraxis, 2011]

3.3.1.5 Coherences

Our framework defines Coherences as values in a one dimensional spectrum (see Figure 3.6). These values define the Node in any desired dimension. We defined some example Coherence types, such as:

1. **Spacial Coherence:**. Physical space of the node. For example: City or Desert.
2. **Temporal Coherence:**. Time and date of the node. For example: 1914 or 1918. (see Figure 3.6)
3. **Story Coherence:**. Position of the node in a plot. For example: Beginning or End.

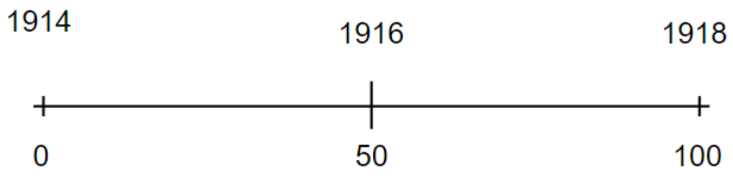


FIGURE 3.6: An example of the possible values for a Temporal Coherence from the aforementioned BF1 example. World War 1 was fought from 1914 to 1918 and each node will have a coherence value according to that node's date. For Nodes that took place in 1914, the Temporal Coherence values should be close to 0, whereas for Nodes that took place in 1918 should have a Temporal Coherence values should be close to 100.

These can be added to any node and defined with a scalar value from 0 to 100. For example, the Spacial Coherence value of 0 could determine a Metropolitan City physical space, whereas a value of 100 would be a Desert village. Coherences are used to generate procedural stories, based on players previous nodes. Each Coherence i has a procedural story generation weight w_i that is used when finding for possible next nodes using a weighted Euclidean Distance. Concretely, if the last node the player played is $\mathbf{N_1}$, the cost of going to node $\mathbf{N_2}$ is

$$\Psi(\mathbf{N_1}, \mathbf{N_2}) = \sqrt{\sum_i w^i (N_2^i - N_1^i)^2}, \tag{3.1}$$

where $\mathbf{N_k} = (N_k^1, ..., N_k^z)$ corresponds to the z-dimensional coherence vector of node k, and w^i corresponds to the weight of the i-th component of the coherence vector. Weights w_i can be used by the game designer to generate a procedural story by calculating the node with the least cost from the entire node pool. Given that we played node N_i, we pick, among the set L of the node pool, the node with lowest coherence distance:

$$l = \arg\min_{l^* \in L} \Psi(\mathbf{N_i}, \mathbf{N}_{l^*}). \tag{3.2}$$

This assures designers that a player will always be directed to a Node that is coherent with their previous one. Let us assume that the player is currently on **Node A**. This Node is a part of the story that is played in New York City, a huge metropolitan city. In this example, this node has two coherences, **Spacial Coherence:** 80 and **Appearance Coherence:** 100. These values are defined by the designer and are totally subjective. For this example **Spacial Coherence** defines how North (Geographically) a Node is, with 0 being South, and 100 being North and **Appearance Coherence** defines the humanization of the Node, with 0 being for untouched landscapes and zones like Deserts and 100 being for completely human filled places like huge metropolitan cities. Then we have **Node B** and **Node C**. Both these Nodes have the same **Spacial Coherence:** 85, meaning they are a little further North than our **Node A**, but **Node B** is a Glacier in Canada, untouched by humankind for years while **Node C** is the city of Toronto. With only these coherences and their values, the most coherent move for a player after New York City, would be Toronto because that would be the least sudden change in coherence possible.

3.4 SP and MP crossing

The way our framework crosses the two game genres, SP and MP, is due to its architecture for MP campaigns. Designers can use our framework to create tailored linear or non-linear stories for strict player types or create a pool (see Figure 3.7 (A)) of unconnected nodes and let the procedural story generator create the story for each individual player as it plays.

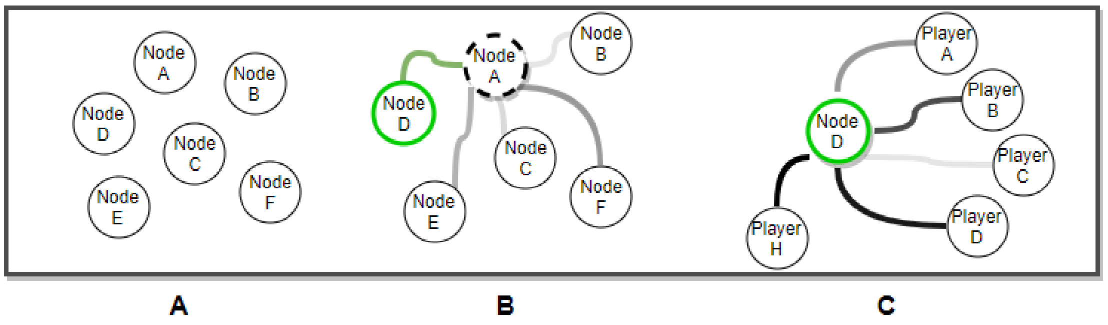


FIGURE 3.7: Flow of a game using the framework. (A)-Node Pool containing every game node created by designers. (B)-Node Matching, node A matches with every Node in the Node Pool, but the minimum cost is Node D. (C)-Player Matching, Player H is hosting a session on Node D where other players will join after they match Player H profile and any open Player Input spots.

To create a procedural story, our framework uses previously defined Coherences to get the closest possible node to a player's current node. Once a close Node has been picked from the node pool (see Figure 3.7 (B)), the player will move to that Node and the matchmaking process begins. The system now has to either find players that have a close and similar profile to our player and are set to play in the picked Node or to find any open sessions (a match already on-going) for this player to join (see Figure 3.7 (C)). For both options, player profiles are compared and matched against each other to allow for balanced matches. In case of failure to find any open sessions, the player starts a game session on the current Node and waits for other players to join later.

SP and MP crossing can still be achieved without a procedural story generation. Designers have full power to decide each player's path based on his/her profile using the next node list on the player outputs for this purpose.

3.5 Matchmaking

The most common method of matchmaking in MP games is any system that uses the Matchmaking Ranking (MMR) value of each player. This numeric value represents the skill of a player. A player with MMR=1000 is less skilled and thus less likely to win against a player with MMR=2000. Each game has its own matchmaking algorithm that is usually based on the Elo System [Elo, 1978]. The Elo system is a rating system created to be used in Chess matches to determine the expected result of two players with unknown strength in the form of a MMR. In the Elo System (see Figure 3.8), the expected outcome of Player A in a match against Player B, with Ra and Rb meaning their respective MMR, is given by [Glickman and Jones, 1999]

$$Ea = \frac{1}{1 + 10^{(Rb-Ra)/400}}. \tag{3.3}$$

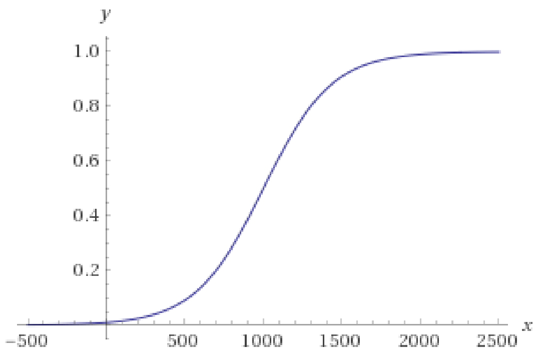


FIGURE 3.8: Plot of the Elo System for the Player B Rating (Rb) = 1000 shows that the lower Player A rating the lower his/her chances of winning the match.

Although the Elo ranking system is commonly used in competitive scenarios, such as E-Sports games like Overwatch, League of Legends or Counter Strike, it may not be truly appropriate to match players by only one criterion in more complex games where player profiles have more than on defining Stat. Instead, we propose to match players based on their various profile's stats using a weighted euclidean distance.

Game designers can associate a weight to each Stat, that will be used in the matchmaking algorithm to generate better matches. For instance, a designer can use a 70% weight on the player MMR, 10% on their experience points, and the remaining 20% on their strength stat. This will match players with close MMR, experience and strength stat. Our proposed player matching formula calculates the distance between two player profiles' **P1** and **P2** using a weighted euclidean based system and is defined by:

$$d(\mathbf{P_1}, \mathbf{P_2}) = \sqrt{\sum_i w^i (P_2^i - P_1^i)^2}, \tag{3.4}$$

where $\mathbf{P_k} = (P_k^1, ..., P_k^z)$ corresponds to the z-dimensional Profile vector containing z Stats and w^i coincide to the weight of the i-th Stat. Then, this distance is used to obtain the cost of matching the player and the node requested profile by using a formula that by taking into account the time in which a player is waiting for a match, favors those matches. This formula is defined by:

$$\Phi(\mathbf{P_p}, \mathbf{P_n}) = \alpha \cdot d(\mathbf{P_p}, \mathbf{P_n}) + (1 - \alpha) \cdot \gamma e^{-\beta t}, \tag{3.5}$$

where α, β and γ are empirically defined scalars the game designer can tune to favor, or not, players that are waiting longer for a match, and t represents time (see Figure 3.9).

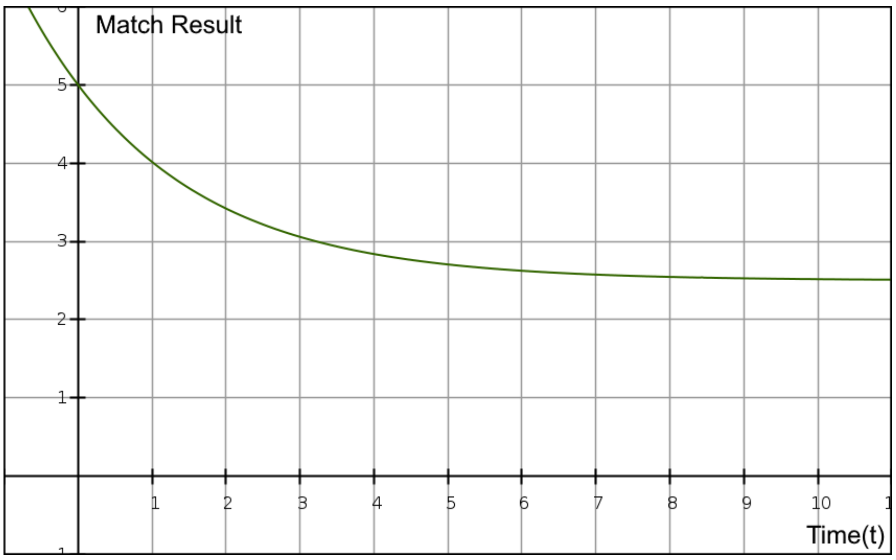


FIGURE 3.9: Plot of our weighted euclidean example for $\alpha = 0.5$, $\gamma = 5$, $\beta = 0.5$ and a euclidean match $d(\mathbf{P_p}, \mathbf{P_n}) = 5$. When the match time increases, the match value will decrease until a minimum of $\alpha \cdot d(\mathbf{P_p}, \mathbf{P_n})$. This will favor ongoing matches and attempt to prevent skipping older matches.

Then, the system computes the cost of matching player p with the set of all available players in open sessions N for that player node n to find the session in which the player best fits, that is, the one with least cost:

$$n = \arg\min_{n^* \in N} \{\Phi(\mathbf{P}_p, \mathbf{P}_{n^*}), \Phi(\mathbf{P}_p, \mathbf{P}_{n^*}) < \sigma\}, \tag{3.6}$$

where σ represents a minimum threshold that the matching needs to fit in order to be considered for matching.

The flow of the matchmaking system begins with a player defined by its profile P and its next node N. The player sends that information to the matchmaking server which upon receiving the request searches for any open matches for node N where the player with profile P can fit in any open slot. While the player waits for this matching, she/he plays a time fill mission previously defined by designers for node N. If a good match is found by the server, the player will join that match and proceeds the game. If no match is found, the server creates a listing for that node and profile and waits for other requests and informs the player to start playing with NPCs. After waiting a previously defined amount of seconds (matchmaking timeout, defined by designers) the server informs the player that there are no available matches and that he/she will have to play against NPCs.

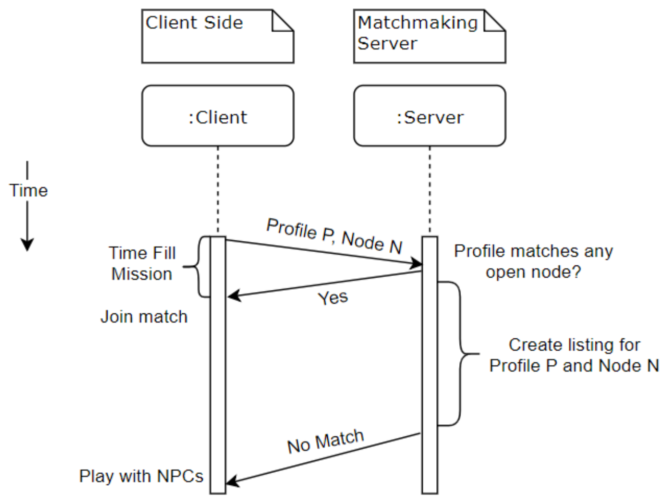


FIGURE 3.10: Flow of the matchmaking initiated by a player sending her/his profile and current node. If no match is found then the player plays against NPCs.

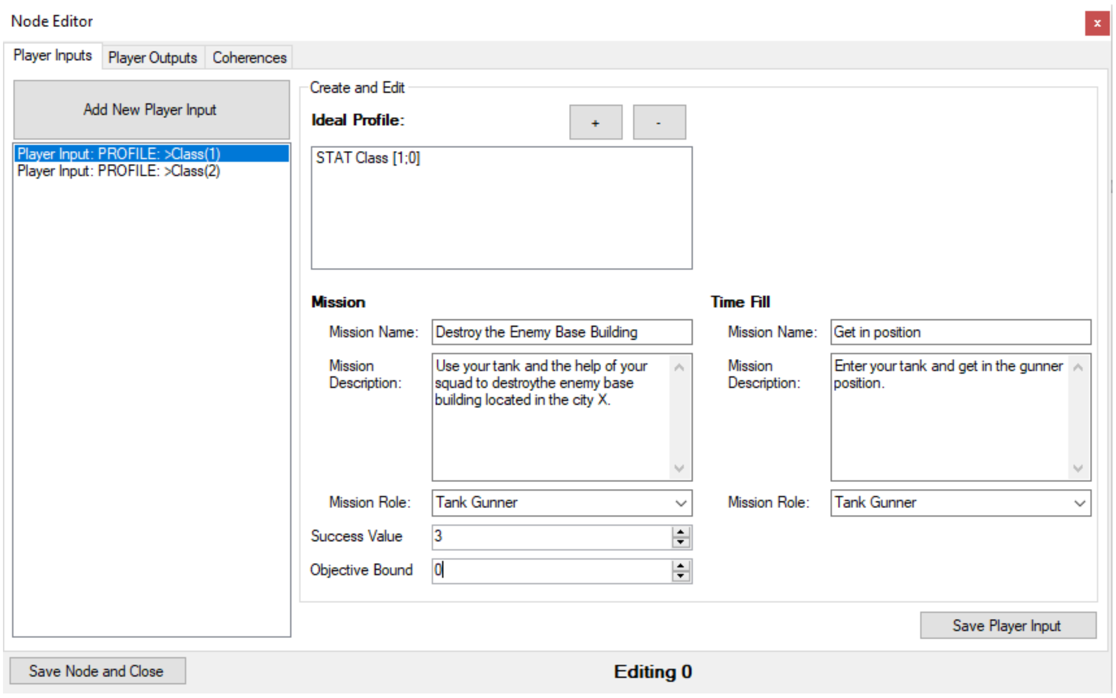
3.6 Designers GUI

Alongside our framework proposal, we have created a prototype of a Graphical User Interface (GUI) Application to aid designers to conceptualize, define and create the game based on our framework components. This application has a core implementation of every component and can be used to create multiplayer campaign stories. As the proposed framework is extendable, and may not cover every designers' need, the framework and GUI Application can be expanded with new components that they see fit. The Application uses JSON [Crockford, 2006] data format to store the campaign story so that data can be exchanged further in the development pipeline for the game's implementation.

In this application (see Figure 3.11) the designer creates every Node for her/his campaign story. In every Node the designer must define: the map where the players will play with a description so that Artists can create the needed assets such as 3D Models; player inputs, outputs and coherences.

For every player input there is a corresponding player output. In the inputs, designers must define the profile for each slot with the corresponding stats that will then be used to match players to each specific input. Then, they define the Mission Information, which is the objective of that specific player in that Node. It can be for example a Protect mission where the player A needs to protect player B or a specific item in the map. Then, the objective bound variable specifies how many times the player can re-spawn after dying and the success value represents how many partial success there are. The designer can also define a Time fill mission which will be played in the time it takes the matchmaking system to match with other players.

Regarding player outputs, the designer defines the output for each specific player input, starting by determining the next nodes in a list sized exactly as the value of the success in the input. Input success value of two means there will be two possible next nodes. The output also can have a time fill mission, which defines what happens after the player finishes his/her mission in that Node, and has to do something to move on to the next Node. Let us imagine a mission to protect a fortress during five minutes; when the time passes and the player is relieved of duty the output's time fill mission could be to reach the commander for the payment or to reach the next city where the next node will start. Finally in the output the designer can define profile enhancements for the player. In the end of a mission, for a complete success the player could be rewarded with 100 experience points, or in a fail situation the player could be weaken and have its strength stat decreased. Each profile in the enhanced profiles list is directly linked to the player's current success. This value is present in the Input variable of "Current Success" which should be keep track of player progress in that node.



Player Input: PROFILE: >Class(1)
Player Input: PROFILE: >Class(2)

Ideal Profile:
STAT Class [1;0]

Mission
Mission Name: Destroy the Enemy Base Building
Mission Description: Use your tank and the help of your squad to destroythe enemy base building located in the city X.
Mission Role: Tank Gunner
Success Value: 3
Objective Bound: 0

Time Fill
Mission Name: Get in position
Mission Description: Enter your tank and get in the gunner position.
Mission Role: Tank Gunner

Editing 0

FIGURE 3.11: Framework Editor for Designers to Manage a story. In this screen-shot the aforementioned example of Player input is used to show how designers can fill the information in the application.

After defining every Node and its individual information using this application and the proposed framework, the designer exports the campaign. The application generates a JSON file containing all information about the story. The designer then passes it onto Artists and Engineers. Artists are the ones focused on the aesthetics part of the game development phase, such as the creation of Textures, 3D Models, 2D Sprites and any other needed asset. Engineers on the other hand, more concretely Software Engineers are the ones that implement the gameplay mechanics so the game becomes playable. Other types of engineers such as Audio Engineers can also use the exported campaign to develop the Audio assets needed for the visioned game. Artists and Engineers then implement their work into the game.

3.7 Case Study

To test the implementation of the proposed framework in a real game development scenario, we designed and developed a game so we could use it as a practical testing area. It was decided that a Role Playing Game (RPG) would be the genre that would use the framework to its full capacity as this game genre usually focus on individual player stories and centers the game on the player and her/his actions.

A plot for the game was created about the tale of two bordering nations that needed a rare mineral present in their border. From there, 19 Nodes were created describing a journey across the border, from north to south, with different missions opposing both countries (see Figure 3.12). With the help of PCG, the design process was streamlined. We used Donjon's Random Dungeon Generator [Donjon, 2009b] to generate the maps of the game, Donjon's Random Adventure Generator [Donjon, 2009a] to generate the plot and objective for each node, and finally, we used Fantasy Name Generators [Emily, 2012] website to generate random names for the game's cities.

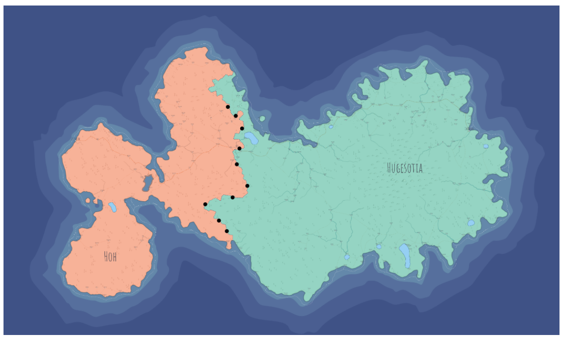
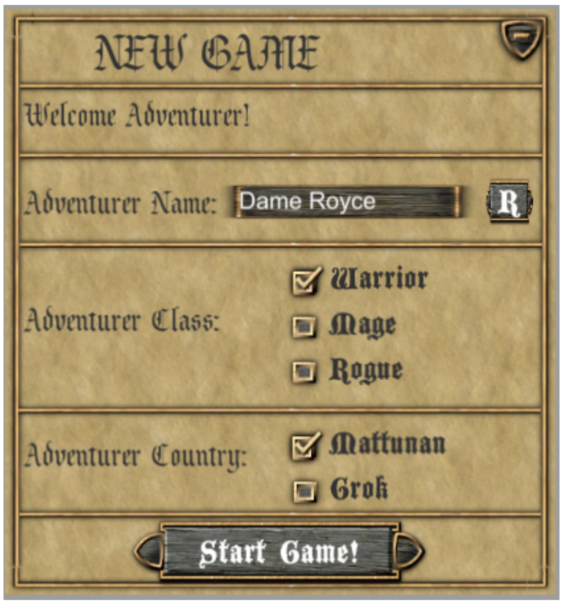


Figure 3.12: Case Study game map procedurally generated using Axgaar Generator [Azgaar, 2018]. Each black dot represents a Node's location.

Nodes can be played in a linear fashion like most games, but we can also create procedural stories by using Node Coherence components to stochastically assign the player's next node. These Nodes were defined by three types of Coherences: *Location*, *Story* and *Appearance*; for instance, a Node that is physically located in the North of the world would have a *Location* value of 0, while a Node physically located in the South would have the value of 1. The game flow starts as players enter their desired nick name and choose a character class from three possibilities (see Figure 3.13). This character class will be the criterion used in the matchmaking system. The matchmaking algorithm will favor to match two players in a given node, with the same class, before matching players with different classes (see equation 3.6). In this game, even if two players with different classes are matched against each other, the match will not be unbalanced for one of them as character damage and health are the same for any class.



Adventurer Name: Dame Royce

Adventurer Class: [✓ Warrior] Mage Rogue

Adventurer Country: [✓ Mattunan] Grok

Figure 3.13: Unity Game Menu with a randomized character name generator, class and country selection.

After choosing a character class and her/his nickname, the game begins and every player starts by playing a tutorial node against NPCs to get familiar with the game. With this game we wanted to test the implementation of the framework alongside the game development pipeline. Then, we wanted to test whether the game (built on top of the framework) plays well with our matchmaking system and if players are indeed matched together in MP combat.

We implemented two versions of our game: a web and an Unity version. We describe both below.

3.7.1 Web Version

We decided to create a web version of the game because it is essentially a Dedicated server architecture (see Figure 3.14) but with no upfront costs given that there are free web hosts that can be used to act as the game's server. This game would be used to validate the framework implementation and the matchmaking system.

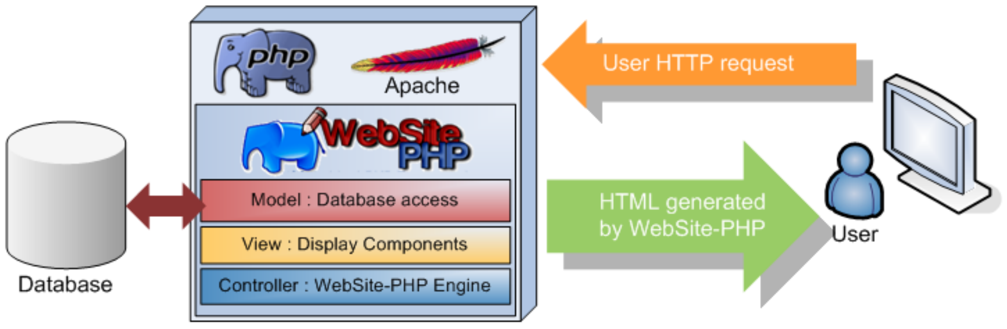

php
Apache
WebSite PHP
Model : Database access
View : Display Components
Controller : WebSite-PHP Engine
Database
User HTTP request
HTML generated by WebSite-PHP
User

Figure 3.14: Web Architecture for our game web version. From the User, an HTTP request (AJAX - Asynchronous JavaScript And XML) is sent to the server side. In the server side, a PHP script receives the AJAX request, processes it, accesses the Database and returns either HTML or JSON which the User receives and uses.

Our web version was implemented using HTML, CSS, Javascript, JQuery, PHP and MySQL. Javascript was used on the client side to control the game logic and player input whereas PHP was used on the server side for matchmaking requests and game logic application. For example, whenever a player requested a match, a request was made to the PHP side of the game as shown in the matchmaking flow (see Figure 3.10). MySQL was used to store player information, matchmaking requests, inputs and node information according to the data model present in Figure 3.15. The Nodes table held information regarding the playable Nodes, wehreas the Player table held information such as players' profiles and current node. For matchmaking purposes the Matchmaking table was used to create and consult available listings and when player were matched the battle table held all the damage given and received by each player.

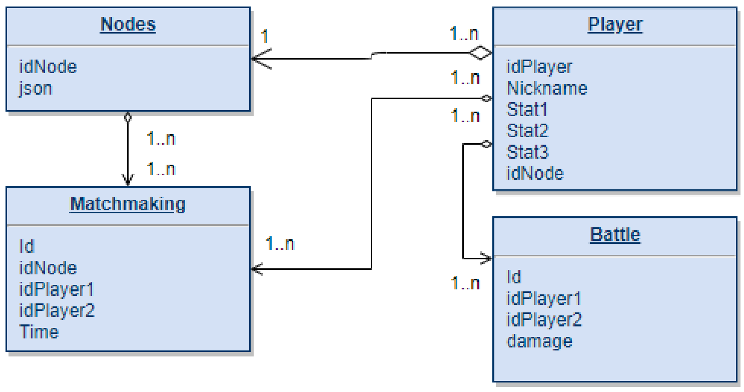
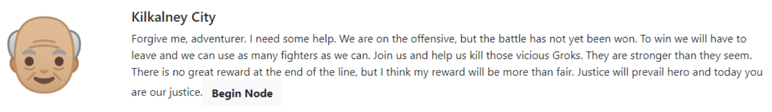


FIGURE 3.15: The Database model for our game featuring four tables. The Nodes table uses JSON to store the information regarding each node.

The game can be played with just a mouse where players would decide when to start a node. The name and description of the node is presented to the player like shown in Figure 3.16. In this example the node represents a City that is under attack by the rival nation and the player must defeat her/his opponent.



Kilkalney City

Forgive me, adventurer. I need some help. We are on the offensive, but the battle has not yet been won. To win we will have to leave and we can use as many fighters as we can. Join us and help us kill those vicious Groks. They are stronger than they seem. There is no great reward at the end of the line, but I think my reward will be more than fair. Justice will prevail hero and today you are our justice.

FIGURE 3.16: Screenshot of a player just before starting a Node, where details about that node are presented to her/him.

In case of battle, the player had the opportunity to decide from three types of strategies: Attack (which dealt 10 damage), Counter-Attack (which dealt a random value between 5 and 15) or Defense (which blocked 50% damage dealt by the opponent). In case of failure, i.e death, the player would have to replay the same Node until successfully winning the battle and moving on with the story.

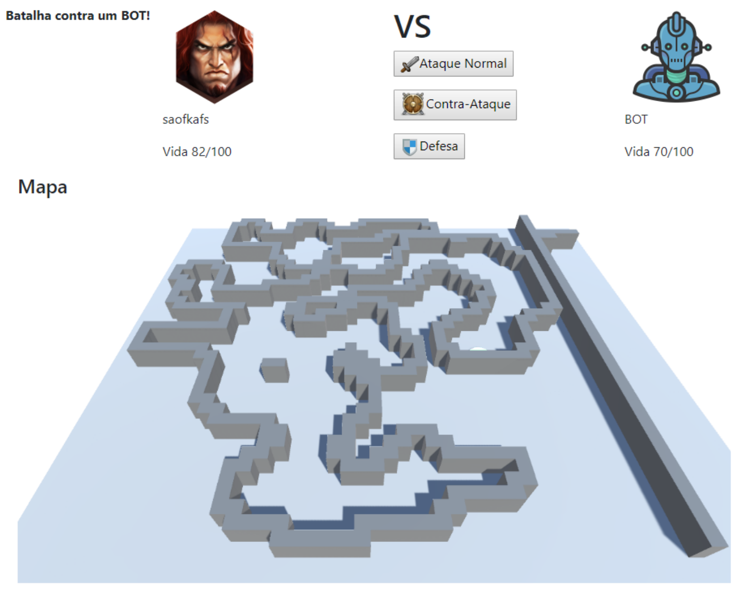


FIGURE 3.17: Web Game Screenshot showing a battle between a player and NPC due to failed matching.

3.7.2 Unity Version

After implementing and validating the web version, we decided to create a 3D Unity version to validate the extendability of the proposed framework. The Unity Engine was chosen due to its rising popularity over the years, cost (free for both personal and commercial use), and community support.

We used the same design from the web version, i.e node information and game plot. We downloaded every visual asset from the internet. Character models 3.18 were downloaded from OpenGameArt [Art, 2009] and animated using Mixamo's auto-rigger and animator [Mixamo, 2008]. Other 3D models such as weapons were downloaded from TheFree3D [Free3D, 2014].

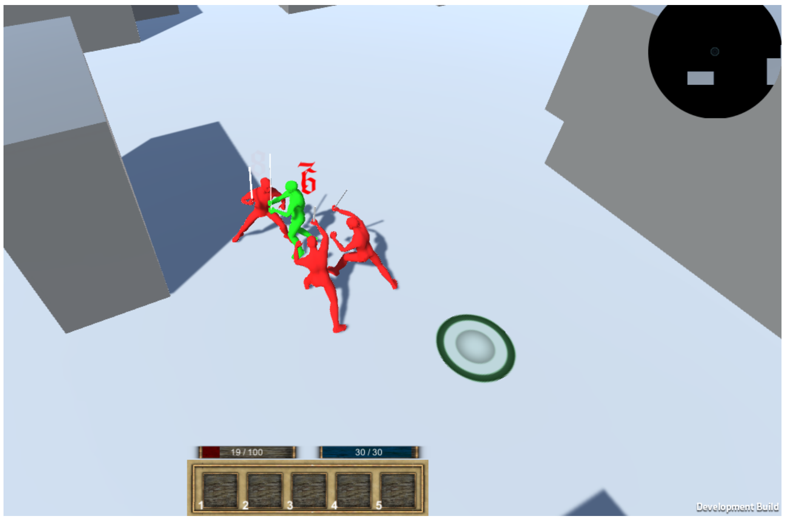
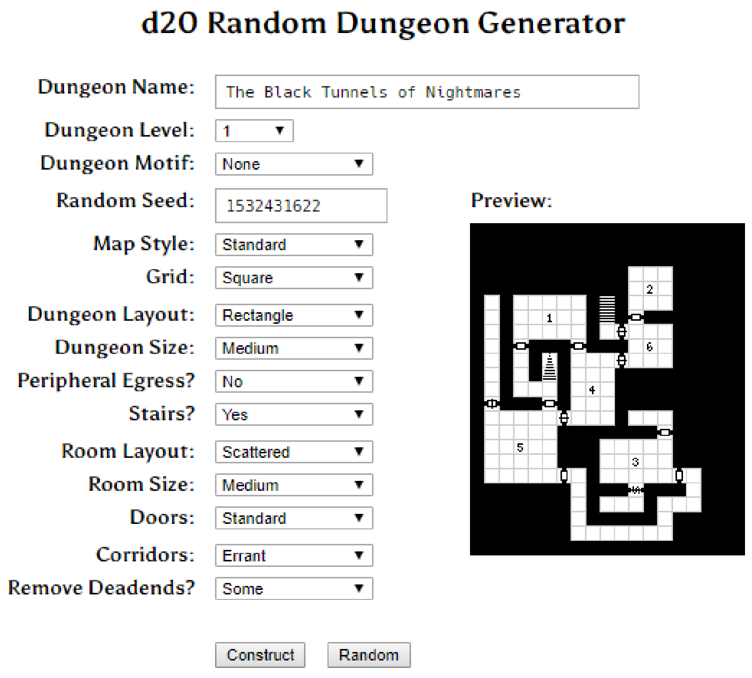


FIGURE 3.18: Unity Game Screenshot with a player in a generated map fighting three opponents.

The maps for each node were randomly generated using Donjon's Random Dungeon Generator (see Figure 3.19) which generates a 2D matrix that can then be used inside Unity to draw the 3D map. Unity allows exportation to a wide variety of platforms such as PC, Consoles and phones. Our test builds were all PC versions since those were the platforms we used for development.



d20 Random Dungeon Generator

Dungeon Name:	The Black Tunnels of Nightmares
Dungeon Level:	1
Dungeon Motif:	None
Random Seed:	1532431622
Map Style:	Standard
Grid:	Square
Dungeon Layout:	Rectangle
Dungeon Size:	Medium
Peripheral Egress?	No
Stairs?	Yes
Room Layout:	Scattered
Room Size:	Medium
Doors:	Standard
Corridors:	Errant
Remove Deadends?	Some

FIGURE 3.19: Donjon's Dungeon Generator where it is possible to create procedural dungeons with rules and constraints such as Layout, style and size.

Similar to the web version, this version of the game can also be played with a mouse or with a gamepad. Here, the flow of gameplay is a bit different: the player initiates the game and is presented a menu to start playing. Immediately the player's current node is loaded and the matchmaking flow is initiated (see Figure 3.10). Then the player spawns in the world and can interact with an NPC where that node details are presented to the player such as node name and description. If the matchmaking found a match, the world will be populated with other human players, if not, it will be populated with enemy NPCs (see Figure 3.20). If the player reaches the end of the dungeon she/he wins the node and moves on in the story, but in case the player dies she/he has to return to restart the node.

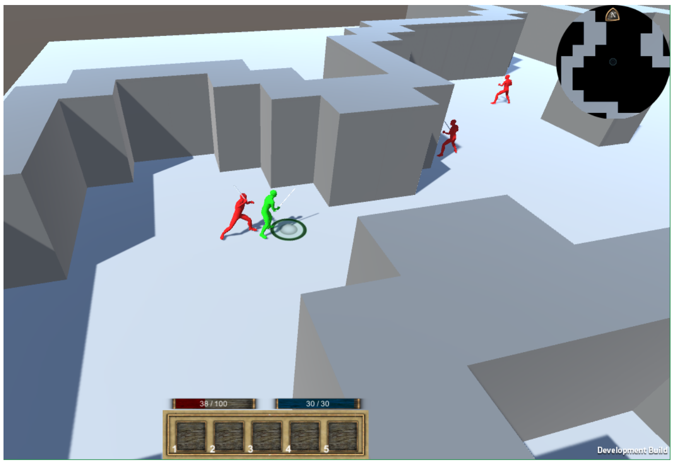


FIGURE 3.20: Unity Game Screenshot with a player in a generated map with NPCs.

3.7.2.1 Networking

Networking is an important part since the proposed framework is for multiplayer interaction. We decided to use Listen Servers (see Figure 3.22) rather than Dedicated Servers (see Figure 3.21) for economic purposes.

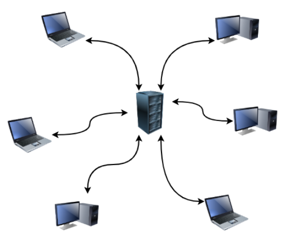


Figure 3.21: Dedicated Server Architecture where players connect to a server that is running online 24/7. This architecture costs upfront money for developers.

Dedicated servers act as a centralized authority in a multiplayer session and players' clients connect to that server in order to play with other players, whereas Listen servers act as dedicated servers while being able to act as a client as well, thus playing and serving at the same time. For small games and indie companies that have little or no budget for dedicated servers (that have to run 24h/7days and there needs to be a high quantity of server instances proportional to the player base), listen servers are the best choice as any player can, essentially be the server. In Unity we can easily implement both of these solutions, so we decided to go with Listen Server, so that a player that could not find a match right away in a matchmaking request could host one's own session and be joined by others.

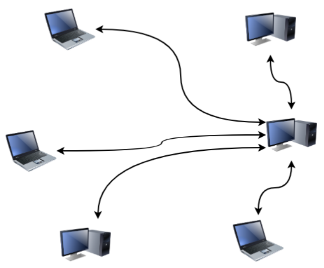


Figure 3.22: Listen Server Architecture where a player is both Client and Server. This architecture saves developers money because each player uses its client and bandwidth to host other players.

For a Listen Server to succeed hosting other players, it must be Port Forwarded. Port Forwarding is a method to open ports in a router to redirect connections to a specific IP address of that router's network. This method only works with specific router permissions which may not be available in certain settings.

Chapter 4

Testing and Discussion

4.1 Implementation

Our case study game was built on top of the framework. We implemented two versions of the game to validate the framework on two different contexts. One the two, a 2D web version, was used to test implementation, matchmaking and real life situation tests, whereas the 3D version was used to extend the implementation validation.

The web version implementation was all done in PHP, MySQL, Javascript and HTML. The proposed framework's components were all implemented as designed previously.

The 3D version was implemented in C# using the Unity Engine. It was important to validate the proposed framework with a widely used commercial engine as Unity is to certify that the framework is easily integrated in such engines.

4.2 Procedural Story Generation

To test our Coherence system, the values that define each Node in various dimensions such as Physical Space or Time, we ran simulations for various sets of weights. We implemented the Coherences at the Node scope, meaning that each Node from the node pool had a set of Coherence values that defined it on Appearance, Location and Story. The Appearance type describes the visual appearance of a level, the Location type describes the physical location of a level and the Story type describes the place in a overall story that node fits in.

Global to every node and coherence are the specific weights for each coherence type. The greater the weight, the more important that coherence will be in the matchmaking algorithm. If a designer wants to give a heavy focus on the story, the weight for the Story coherence should be higher than any other weight, of course, if this value is too high, the story will be more linear. To give some unpredictability to the results, the Random coherence weight was set to 10%.

We used our proposed procedural story generator for four different weight values,
Story heavy:
($w_{story} = 0.8$, $w_{location} = 0.05$, $w_{appearance} = 0.05$, $w_{random} = 0.1$),
Appearance heavy:
($w_{story} = 0.05$, $w_{location} = 0.05$, $w_{appearance} = 0.8$, $w_{random} = 0.1$),
Location heavy:
($w_{story} = 0.05$, $w_{location} = 0.8$, $w_{appearance} = 0.05$, $w_{random} = 0.1$) and
Balanced weights:
($w_{story} = 0.3$, $w_{location} = 0.3$, $w_{appearance} = 0.3$, $w_{random} = 0.1$).

These simulations used our case study game's Node pool, where each node has its own set of coherence values. We also stored the nodes that the player already played so that there are no loops in a story. With each story generation we also calculated the total sum of each coherence value contribution of each node to the final result.

Shown below are the different stories generated by the system. The numbers represent the node ID and $\Rightarrow$ represents the flow of the story as the player progresses in the nodes.

Example 4.1. *Story Heavy*

1⇒ 14 ⇒ 2 ⇒ 12 ⇒ 13 ⇒ 15 ⇒ 4 ⇒ 16 ⇒ 6 ⇒ 5

Total: $Story = 0.44, Appearance = 0.074, Location = 0.225$

Example 4.2. *Appearance Heavy*

1⇒ 17⇒ 15⇒ 10⇒7⇒ 8⇒ 4⇒ 3⇒ 19⇒ 5

Total: $Story = 0.03, Appearance = 0.7, Location = 0.235$

Example 4.3. *Location Heavy*

1⇒11⇒9⇒2⇒14⇒3⇒7⇒15⇒5⇒13

Total: $Story = 0.0275, Appearance = 0.05, Location = 0.36$

Example 4.4. *Balanced Weights*

1⇒2⇒14⇒3⇒15⇒13⇒4⇒7⇒8⇒10

Total: $Story = 0.3957, Appearance = 0.624, Location = 0.528$

Node 7

Story: 0.1

Location: 0.4

Appearance: 0.7

Node 15

Story: 0.2

Location: 0.3

Appearance: 0.6

Node 8

Story: 0.4

Location: 0.8

Appearance: 0.2

As we can see the Node matching generates a procedural story composed on Nodes that are expected to be coherent to one another. For example, in the Location heavy generation, the player would go from Node 7 to Node 15. Node 7's coherence value for Location is 0.4 while Node 15's coherence value for Location is 0.3 meaning that these nodes are located closely in physical space (in a realistic example, the same way Brooklyn and Manhattan are close physically), thus being coherent in a Location perspective.

4.3 Matchmaking

We ran game simulations with NPCs playing against each other to test our matchmaking system and compare it to an Elo system approach. The case study game and all its nodes, presented in Chapter 3, were used for these simulations. We ran simulations for four different numbers of concurrent players $N = \{10, 50, 200\}$. The methodology was to have N concurrent players simulating game-play and matchmaking requests. We tested each player having one, two and three defining stats, which were randomly generated with values between 0 and 10. We also tested restricting or relaxing the matching threshold (see Equation 3.6) to observe how it affects matching. The simulation flow is as follows:

1. Player is created with 1,2, or 3 Random Stats with values ranging from 0 to 10 (exclusive).
2. Player requests a match in the first Node
3. Player waits for timeout or match details
4. Player simulates game play by generating a 50% chance of winning.
5. If Player wins it requests the story generator a new node to progress to, if not, it stays in the same node.

Each player plays exactly 10 matches.

Progression means that players request a new node to our Story Generator which will be set with three Coherences, Story ($w_{story} = 0.8$), Location ($w_{location} = 0.05$), Appearance($w_{appearance} = 0.05$) and Random($w_{random} = 0.1$), which generates some branching but not too much. For our matchmaking equation we used an $\alpha = 1$ to give 100% weight to the euclidean distance (see Equation 3.5) part of the equation, and 0% to the time part of the equation. This is due to the simulation taking a short time to complete, so a time factor is insignificant. We also used a timeout of 5 seconds, meaning a player will wait five seconds for a match if no match is found right away, after those five seconds, the player will play against NPCs.

4.3.1 Proposed Non-weighted Matchmaking Method

We ran tests with the proposed euclidean distance based algorithm (see Section 3.5). Our framework is capable of handling various stats and some of those might be direct inputs to the matchmaking system. For that reason we needed a system that would take in account more than one value, in comparison with Elo based systems that use only the team's rating (MMR).

For one Stat (see Figure 4.1) and $\sigma = 3$ (see Equation 3.6), meaning that the system would only match players with a stat difference of 3 points, our matchmaking algorithm failed to find a match for 26% of the requests with $N = 10$. That percentage dropped to 10% for $N = 50$ and with $N = 200$ it hits the 2.4% of failed match. With a more strict $\sigma = 1$, the results are unfavorable with 56% of failed match for $N = 10$ and the lowest percentage of 3.8% for $N = 200$. Overall, the results show a decrease of failed matchmaking with the increase of player base.

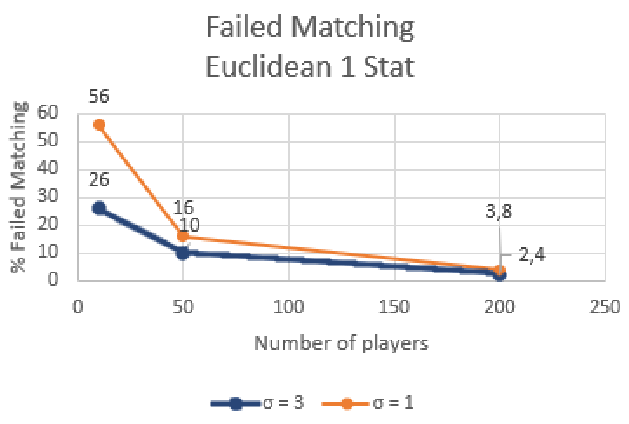


FIGURE 4.1: Results of Euclidean 1 Stat Simulations.

For two Stats (see Figure 4.2) and $\sigma = 3$, the failed matching percentage for $N = 10$ almost doubles to 52.6% and 5.8% for $N = 200$. With $\sigma = 1$ the system hits 70% failed matching for $N = 10$ and with $N = 200$ it manages to keep the unsuccessful matching at 22%.

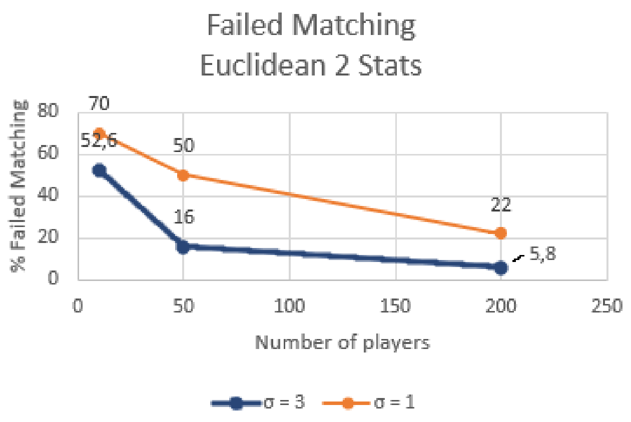


FIGURE 4.2: Results of Euclidean 2 Stat Simulations.

For three Stats (see Figure 4.3) and $\sigma = 3$ the results are very high for small player bases. Around 70% failed matching for $N = 10$ and as low as 14% for $N = 200$ and $\sigma = 3$. The combination of three stats, procedural story generation and 10 possible values for each stat makes it very unlikely that players find a match in lower player bases. We tested with $\sigma = 1$ and we hit a high of 74% for $N = 10$ and a low of 58% for $N = 200$.

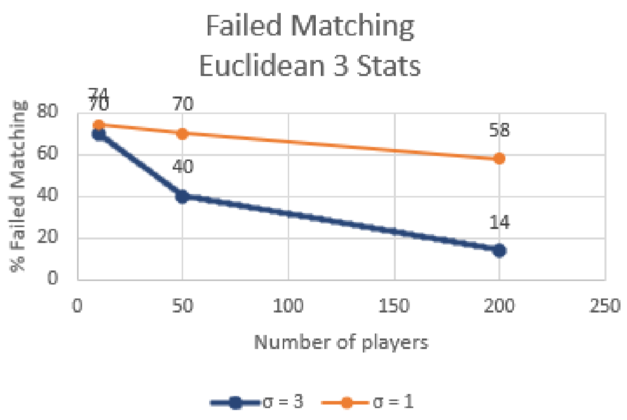


FIGURE 4.3: Results of Euclidean 3 Stat Simulations.

Overall these are good results as a player base of 200 concurrent players is very small for high profile commercial games. Designers must have a clear idea about the branching scale that they will introduce in their game, and the more defining stats and their possible values will affect more the possible matches.

4.3.2 Proposed Weighted Matchmaking Method

Our euclidean based system provides a good matching because it can consider more than one criterion when matching two players. However, the more criterions used, the harder it is to get a good match, specially when the value range is wide. To improve the results for more than one criterion, a weighted euclidean distance system can be used instead. We tested this by using two different sets of weights for the three criterions (stats).

The first set of weights was 80% for Stat 1, 10% for Stat 2 and 10% for Stat 3. This means that when matching two players, there will be significant more importance given to the first Stat difference and the other two stats will have small effect on the final matching value. With this set of weights (see Figure 4.4) the system is more capable of matching with only 3.1% failed matching at $N = 200$ for $\sigma = 3$ and 25% for $\sigma = 1$.

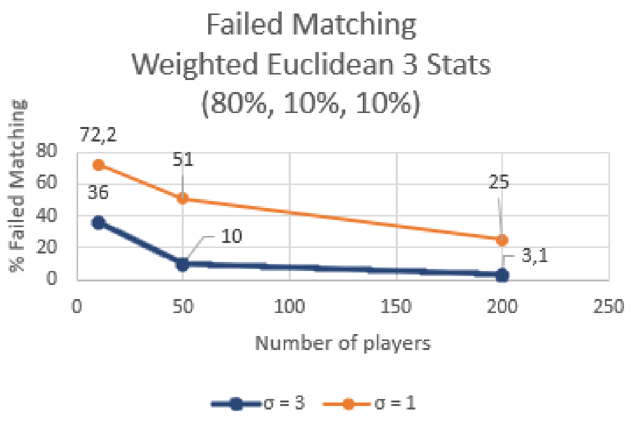


FIGURE 4.4: Results of Weighted Euclidean 3 Stat Simulations with 80%, 10% and 10% weights.

The other set of weights used, was 50%, 30% and 20% (see Figure 4.5), which meant to show extra equilibrium in stat importance. The results show that it also beats the non-weighted euclidean distance based system hitting a low of 4.4% with $N = 200$ and $\sigma = 3$ and 37% for $N = 200$ and $\sigma = 1$.

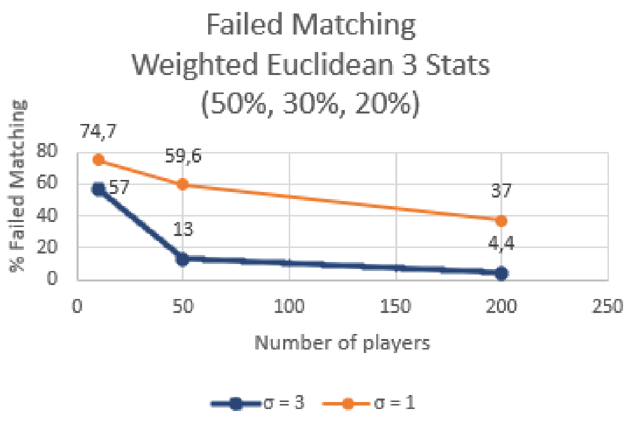


FIGURE 4.5: Results of Weighted Euclidean 3 Stat Simulations with 50%, 30% and 20% weights.

4.3.3 Elo based

We implemented a version of the Elo version with the same attributes as our matchmaking tests to compare both directly. In the case where our Elo system used more than one Stat, the average of every stat was used as the MMR value for the elo equation. We used the generic Elo equation with Ra and Rb being the average of the stats for Player A and B respectively. The perfect match of this equation is 0.5, meaning that either player has 50% chance of winning, so we used

$$m = \arg\min_{Ra,Rb}\{|0.5 - \frac{1}{1+10^{(Rb-Ra)/10}}|\}, \tag{4.1}$$

to find the two players with the fairest match. For example a value $m = 0$ means the match is perfectly balanced. We tested the elo system with $\sigma = 0.033$ and $\sigma = 0.0984$ as we determined that these were the exact values that matched the ones used in the euclidean tests ($\sigma = 1 and \sigma = 3$).

For one Stat (see Figure 4.6) and with both $\sigma = 0.0984$ and $\sigma = 0.033$ hitting a high of 55% and 69% for $N = 10$, respectively, and a low of 3.6% and 13% for $N = 200$, respectively. As previous tests, the failed matching percentage decreases as the player base increases.

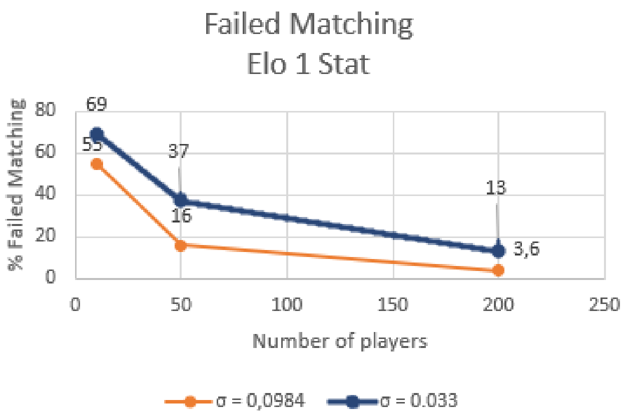


FIGURE 4.6: Results of Elo System 1 Stat Simulations.

For two Stats (see Figure 4.7) and $\sigma = 0.033$ the elo system hit a high of 63% for $N = 10$ and a low of 6.8% for $N = 200$ whereas $\sigma = 0.0984$ hit a high of 42.7% for $N = 10$ and a low of 2.8% for $N = 200$.

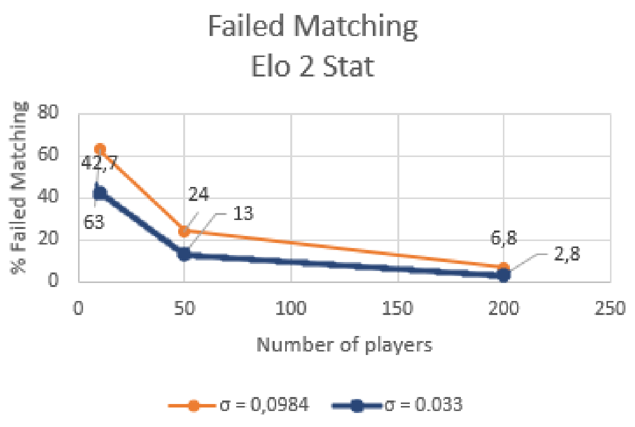


FIGURE 4.7: Results of Elo System 2 Stat Simulations.

For three Stats (see Figure 4.8), for $\sigma = 0.033$ and $\sigma = 0.0984$ the elo system achieved a high of 56% and 38% for $N = 10$, respectively, and a low of 10% and 2.5% for $N = 200$, respectively.

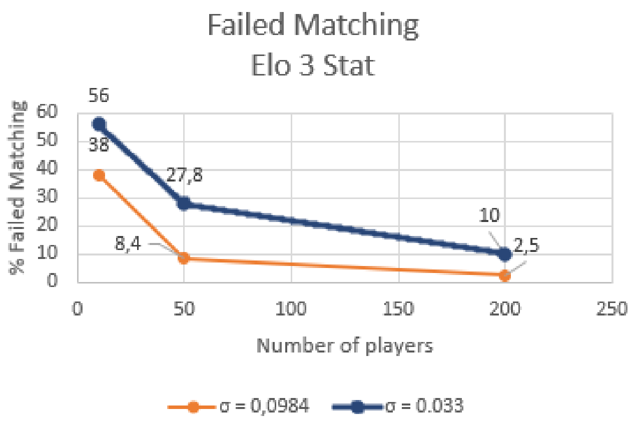


FIGURE 4.8: Results of Elo System 3 Stat Simulations.

4.3.4 Analysis

Let us now compare the proposed methods (weighted and non-weighted) and the elo based matchmaking systems according to their performance for each threshold σ according to the previous shown results. For elo based $\sigma = 0.0984$ is related to $\sigma = 3$ for euclidean distance based and $\sigma = 0.033$ is related to $\sigma = 1$.

For one Stat profile matching, both systems do a good similar job. The non-weighted euclidean based system manages to win with lower percentages for lower σ but overall both systems show good percentages for $N = 200$.

For two Stats, the euclidean system fails 22% at a strict σ value, whereas the elo based system fails at 6.8%. For both relaxed σ and higher $N = 200$ the two systems are separated by 3% making the elo system the winner with 2.8%.

For three Stats, the euclidean system hits 14% failed matching with $\sigma = 3$, whereas the elo based system handles the stricter $\sigma = 0.033$ with the lowest 2.5%. However with weights, the euclidean system hits 3.1% with 80/10/10 and 4.4% with 50/30/20.

In terms of failed percentage the elo appears to be able to match players more easily in diverse situations, however this does not prove its total efficacy. Let us imagine an example of a three criterion matching with Stat A,B, and C. Each player can have one of the stats (A,B,C) with value ten and the rest of the values are 0. Player X would have $A = 10, B = 0, C = 0$ and player Y would have $A = 0, B = 10, C = 0$. Now let us try and match Player X and Y. Following the euclidean distance (non-weighted), the system would give a matching value of $m = 14.1$, showing that the players are indeed different. But when we switch over to the elo system, as it only handles one criterion, we have to average the stats giving us a total match value of $m = 0.5$ because the average of the three stats is the same 10/3.

The elo works as proven by games such as Rocket League, Overwatch, CS:GO, League of Legends and others, but is limited when given the possibility to match with more than one criterion than just skill, and as demonstrated in the previous example, it can match players but it does not guarantee a balanced match in every situation. However, our framework is not bound to the euclidean based systems, thus being possible to use and implement elo based systems or any more suited system that designers see fit for their game.

4.4 Designers GUI

To test our Designers GUI, we ran usage tests ($N = 14$) to assess the usability of our GUI application to design and create multiplayer campaigns. We started by creating a small introduction presentation about our framework (see Figure 4.11 and Appendix B). This presentation explains the framework overview and how it can favor the development process. Our tests were conducted with 14 people, 12 of which were male and 2 females, with ages comprised between 20-27 years of age. All the males were familiar with video-games while the two female subjects were not that familiar, only playing on their phones but very little time per week. From all the 14 users, only 3 have any experience with the game creation or design process.

After a brief introduction, we initialized these tests by asking users to give an example of a Node in their favorite games. The results of this query are present in Figure 4.9. Overall the understanding of the concept of Node in our framework was understood, with only three subjects failing to present an example of such concept in their favourite video games. For example, one of the answers from one of the successful subjects was "A semi-final match between two teams in a FIFA video game tournament", indeed the journey of a team in a FIFA tournament could be represented as a set of Nodes, from the Eighth-finals all the way to the final.

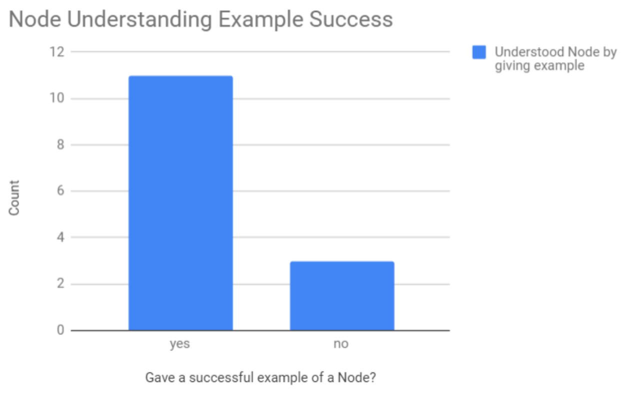


FIGURE 4.9: A query during our tests where users were asked to give a practical example of a Node in their favorite games showing a good understanding of what a Node is.

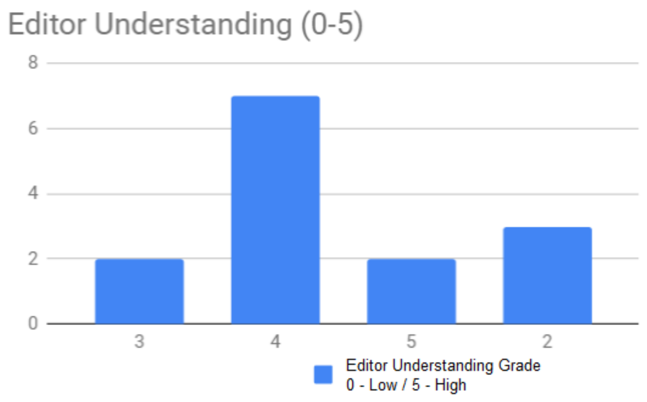
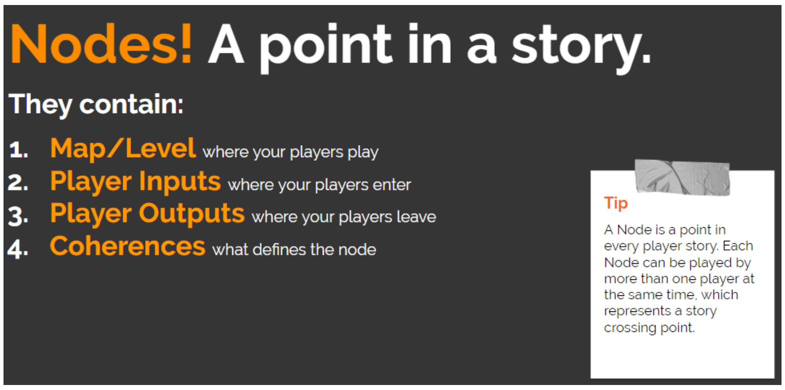


FIGURE 4.10: Our framework editor understanding results were positive. With an average of 3.57.

We then presented the GUI Application to the users and requested the creation of two missions with different player profiles and connection between one another. We presented these tests in the form of a guide (see Appendix A), but before starting we made it clear that our framework was the subject of evaluation and testing and not the person doing the testing, so that the users would not feel judged or pressured to get things right. The goal of these tests was to assess whether our framework concepts were easily recognizable in the editor and correctly used by new users. The results (see Figure 4.10) were positive with an average of 3.57/5 of editor understanding. This understanding was rated based on both self and our evaluation of each subject's ability to follow the directions given by our guide and time consumed doing so. Only three subjects showed problems in following these directions, due to the lack of understanding of how the framework components connected to one another. This is partially explained by their lack of video game play experience, since other subjects that are daily players understood these concepts. Other factor that could possibly explain a bad understanding of the editor application is the lack of polish for the User Interface (UI) and User Experience (UX) while developing the application. We also did not see a factor in tech-background in our results since formation background for our subjects ranged from sports and health, to no higher education (for example: Fireman).



Nodes! A point in a story.

They contain:

1. Map/Level where your players play
2. Player Inputs where your players enter
3. Player Outputs where your players leave
4. Coherences what defines the node

Tip

A Node is a point in every player story. Each Node can be played by more than one player at the same time, which represents a story crossing point.

FIGURE 4.11: Framework Presentation Slide for framework introduction. See the full presentation in Appendix B

4.5 Real-life tests

We tested our case study game, fully implemented with the framework in a real controlled environment. Before beginning these tests, we setup ten computers in a room with the game running. Testing heavily with lots of users was not possible so instead we tackled some possibilities and compared them to our simulations (see Figure 4.12). We initiated the register of ten players with random classes from a possible of 3 values (Warrior, Mage and Rogue), each player has its own computer. These tests were made using a non strict euclidean threshold of $\sigma = 3$, which would made possible the matching of different classes. The first methodology used was a random picking of two players and having them requesting a match. Then, after five rounds we picked two players that would stop playing and continued with the others. Then we picked four players that would advance as much as they could in the game while the others took a break. When these four ended the game, we resumed the same random picking methodology used before, this time including the two players that left the game early, but now picking one other that would stop playing. Soon enough there were players which completed the game, and we picked two players at random that would drop the game completely. The results for $N = 10$ and $\sigma = 3$ were of 55% failed matching out of 65 total requests.

Our simulations for $N = 10$ and $\sigma = 3$ with one stat showed that the matchmaking system failed on 26% of the requests. We believe the reason for the difference is due to the fact that simulations did not take in account any player drop out

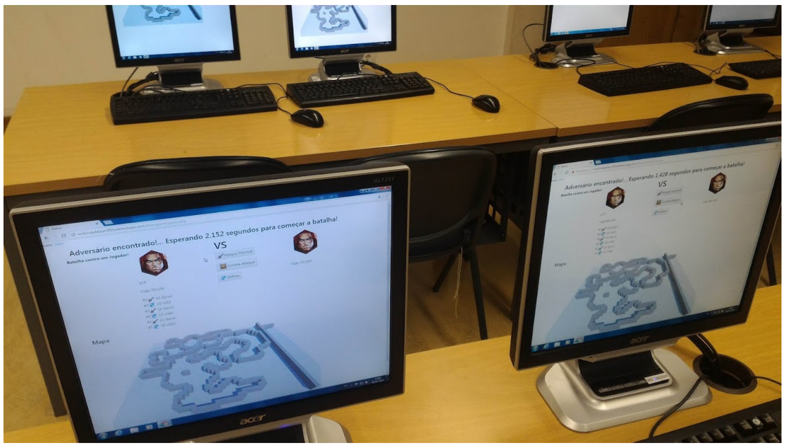

Adversario encontrado!... Esperando 2.152 segundos para começar a batalha!
VS
Mapa

FIGURE 4.12: Real Life tests showing two players being matched.

rate as they were sequential and fully randomized. Overall, results were positive since $N = 10$ is a very small player base for any game and there were players who found a match in the majority of cases, i.e there was no player that only played against NPCs.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

The video game industry is full of high quality games. Both single-player and multiplayer are heavily played all around the world during many hours per week by millions of players. However, they have flaws that are inherited by their setting. Single player games quality is tied to their story and NPC quality, whereas multiplayer games quality is tied to their interaction and game-play. By nature, the video game industry is very commercial, meaning that proprietary solutions are paramount and not discussed academically, so there is not a single solution to aid the design of multiplayer campaigns such as Dark Souls invasions mode, Absolver or Journey video games.

As a contribution to this gap in the industry, we developed a framework capable of integrating popular game design methods such as the MDA framework and GDD usage. Our framework was built to take from SP games the well written and tailored stories to a MP interaction setting where the gameplay does not feel dull or repetitive. We tested this framework in different stages of the game development pipeline.

For the design phase, designers can use our Framework Editor Application to create and manage a multiplayer campaign story where they define all the levels in the game and the types of players that will play those same levels. Once the design phase is complete, the game needs to be implemented by programmers and artists. Programmers use the design to create the gameplay mechanics and the inner workings of the game, while Artists (Sound, Visual, etc.) focus on creating the needed assets.

We tested the programmers usage of the framework by implementing the framework data defined previously by a designer in our case study game. Our case study game, and several implementations of it, are evidence that the framework and its components are capable of handling the core of any story related game genre's development.

Inside our case study game we tested a proposal for a matchmaking system that we expected to be more effective than the popular Elo system, since it uses more than one stat for the matching of players. We started by comparing our non-weighted euclidean based system with the elo based system. For one stat both systems are more successful in matching players as the player base increases. With two and three stats the euclidean based system efficiency decreases, as there are less successful matches, but with weights, the euclidean based system manages to keep low failed percentages. On the other hand the Elo system maintains close results either for one, two or three stats, however this does not mean the matches are balanced like shown. The Elo is capable of matchmaking but not in a balanced way since it needs to average the number of criterions used and an average of multiple stats is not representative of balance in some cases, whereas an euclidean based system can differentiate each individual stat value and give an accurate difference of two different players. Our framework is not bound to any specific matchmaking system, so each developer can use whatever system suits their specific needs.

Our framework showed to be capable of not only handling a multiplayer campaign with as much branching as the designers desire, making games feel unique to any type of player, but also of aiding in the development of such games by streamlining the process of conceptualization, design, and implementation providing tools for developers to create, manage and share their work back and forth.

5.2 Future Work

This section details some of the possible extensions of our contribution to the video game industry. It analyzes each individual contribution separately in each sub-section.

5.2.1 Framework

The framework provides enough components for most known commercial game genres, such as Shooters, Strategy and Action/Drama games. However it is possible that a particular type of game either by being innovative or having features that are not very common may need additional components. In cases like this, it is possible to override and add new components that interact or not with the core components proposed by this dissertation.

5.2.2 Framework Editor

Our framework editor albeit just a prototype, could be improved in various ways. With the aforementioned framework modifications, the editor too could be modified in its core to provide the addition of new components that any particular game types may need. Other feature that would significantly help the designers in the design process would be a real time simulation of the nodes and the possible paths. This simulation would be built in the editor so that after the creation of several nodes, designers could see a simulation of what emergent paths would be created and tune in variables in real time. This would help since the branching increases the complexity of the possible paths undoubtedly.

5.2.3 In Engine

The editor and the framework are built as standalone products, but instead they could be added as plugins or addons to existing Engines such as Unity or Unreal Engine. There, the creation, management, and implementation process could be simplified by having all components already built onto the engine, thus not needing an extra implementation process. This would also help designers that do not have enough programming background to mess around with the implementation to create their stories in a drag-and-drop/visual programming environment and then the plugin/addon would translate that automatically into objects and instances in-game.

5.2.4 Scale

The conducted tests were based on a small scale compared to the thousands of daily players on current commercial video games. Our proposed framework and matchmaking systems could use additional validation on a bigger scale.

5.3 Contributions

The work done in this dissertation was featured in an article accepted for publication at the ArtsIT 2018 Conference in Braga, Portugal.

Appendices

Appendix A

Designers GUI Test Guide

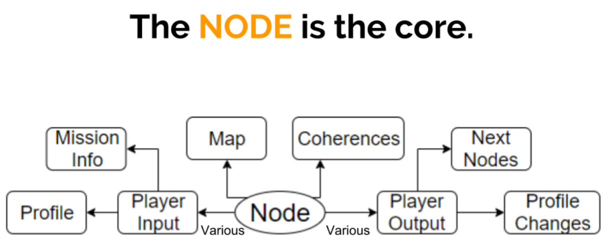
The guide that was given to user subjects to follow and use the Designers GUI application.

1. You are a Game Designer and you just got introduced to a new Framework for Multiplayer Campaign Games.
2. You want to create a game in medieval times. There will be two missions, one tutorial and one huge battle.
3. First create a tutorial mission
   (a) This mission doesn't discriminate individual players
4. Now create the battle mission
   (a) This mission will have 2 types of players, team A and team B
   (b) Each side either wins or loses.
5. Now export your campaign and send it to programmers.

Appendix B

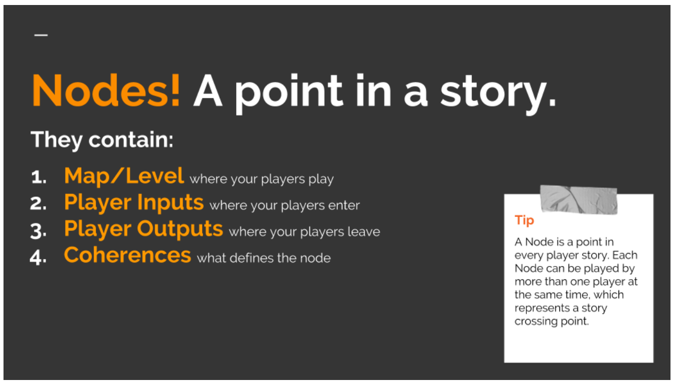
Framework Presentation Slides

The framework slides of the presentation given to Designer GUI application test subjects.



The NODE is the core.

FIGURE B.1: Slide 1 of the Framework Presentation.



–

Nodes! A point in a story.

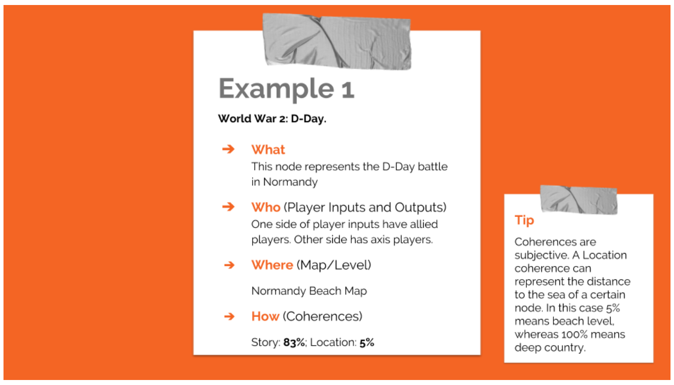
They contain:

1. **Map/Level** where your players play
2. **Player Inputs** where your players enter
3. **Player Outputs** where your players leave
4. **Coherences** what defines the node

Tip

A Node is a point in every player story. Each Node can be played by more than one player at the same time, which represents a story crossing point.

FIGURE B.2: Slide 1 of the Framework Presentation.



Example 1

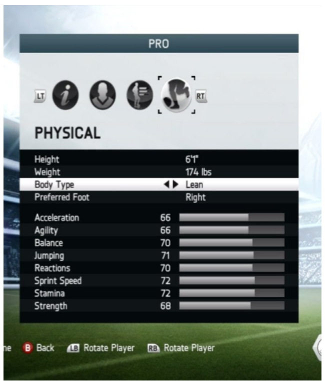
World War 2: D-Day.

- **What**
  This node represents the D-Day battle in Normandy
- **Who** (Player Inputs and Outputs)
  One side of player inputs have allied players. Other side has axis players.
- **Where** (Map/Level)
  Normandy Beach Map
- **How** (Coherences)
  Story: **83%**; Location: **5%**

Tip

Coherences are subjective. A Location coherence can represent the distance to the sea of a certain node. In this case 5% means beach level, whereas 100% means deep country.

FIGURE B.3: Slide 1 of the Framework Presentation.



PRO

PHYSICAL

```
Height            6'1"
Weight            174 lbs
Body Type         ◀ ▶ Lean
Preferred Foot    Right

Acceleration      66
Agility           66
Balance           70
Jumping           71
Reactions         70
Sprint Speed      72
Stamina           72
Strength          68
```

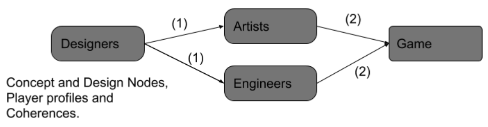
Player Profiles

A Profile is a set of Stats that define a player.

These can be used in Player and Node matching.

FIGURE B.4: Slide 1 of the Framework Presentation.

The Flow



1. **Designers** pass their design to **Artists** and **Engineers**
   a. **Artists** create the visual assets such as 2D Sprites or 3D Models.
   b. **Engineers** create the game mechanics
2. **Artists** and **Engineers** create the **Game** itself.

FIGURE B.5: Slide 1 of the Framework Presentation.

Bibliography

[Agarwal and Lorch, 2009] Agarwal, S. and Lorch, J. R. (2009). Matchmaking for online games and other latency-sensitive p2p systems. In *Proceedings of the ACM SIGCOMM Computer Communication Review*, volume 39, pages 315–326. ACM.

[Alman and McKay, 2017] Alman, J. and McKay, D. (2017). Theoretical foundations of team matchmaking. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1073–1081. International Foundation for Autonomous Agents and Multiagent Systems.

[Andrade, 2013] Andrade, A. (2013). Gdd examples. http://seriousgamesnet.eu/assets/view/238. Accessed: 2018-04-26.

[Art, 2009] Art, O. G. (2009). Open game art. http://www.opengameart.org. Accessed: 2018-04-26.

[Azgaar, 2018] Azgaar (2018). Fantasy map generator. https://azgaar.github.io/Fantasy-Map-Generator/.

[Barrett, 2017] Barrett, B. (2017). Overwatch just reached 35 million players. https://www.pcgamesn.com/overwatch/overwatch-sales-numbers. Accessed: 2018-04-26.

[Bertolo, 2014] Bertolo, M. (2014). Game design document.

[Brew, 2015] Brew, S. (2015). Until dawn, the interactive movie, and storytelling. http://www.denofgeek.com/games/until-dawn/37259/until-dawn-the-interactive-movie-and-storytelling.

[Crawford, 1984] Crawford, C. (1984). The art of computer game design.

[Crockford, 2006] Crockford, D. (2006). The application/json media type for javascript object notation (json). Technical report.

[Delalleau et al., 2012] Delalleau, O., Contal, E., Thibodeau-Laufer, E., Ferrari, R. C., Bengio, Y., and Zhang, F. (2012). Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):167–177.

[DIGITAL, 2017] DIGITAL, S. . D. (2017). Absolver video game. https://www.absolvergame.com/. Accessed: 2018-04-26.

[Donjon, 2009a] Donjon (2009a). Donjon random adventure generator. https://donjon.bin.sh/fantasy/adventure/. Accessed: 2018-04-26.

[Donjon, 2009b] Donjon (2009b). Donjon random dungeon generator. https://donjon.bin.sh/fantasy/dungeon/. Accessed: 2018-04-26.

[Elo, 1978] Elo, A. E. (1978). *The rating of chessplayers, past and present.* Arco Pub.

[Emily, 2012] Emily (2012). Fantasy names generator. http://www.fantasynamegenerators.com/. Accessed: 2018-04-26.

[Free3D, 2014] Free3D (2014). Free3d. https://free3d.com/. Accessed: 2018-04-26.

[Glickman and Jones, 1999] Glickman, M. E. and Jones, A. C. (1999). Rating the chess rating system. *CHANCE-BERLIN THEN NEW YORK-*, 12:21–28.

[Grubb, 2017] Grubb, J. (2017). Ashen expands on journey's passive multiplayer by turning real strangers into companions. https://venturebeat.com/2017/06/13/ashen-expands-on-journeys-passive-multiplayer-by-turning-real-strangers-into-npcs/. Accessed: 2018-04-26.

[Hamilton, 1995] Hamilton, K. (1995). Gta gdd. http://www.mikaelsegedi.se/gdd/Grand-Theft-Auto-Design-Document.pdf. Accessed: 2018-04-26.

[Hendrikx et al., 2013] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1.

[Herbrich et al., 2007] Herbrich, R., Minka, T., and Graepel, T. (2007). Trueskill^{TM}: a bayesian skill rating system. In *Proceedings of the Advances in neural information processing systems*, pages 569–576.

[Hunicke et al., 2004] Hunicke, R., LeBlanc, M., and Zubek, R. (2004). Mda: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1722.

[Inc., 1994] Inc., C. (1994). Diablo gdd. http://mikaelsegedi.se/gdd/diablo$_{p}itch.pdf.Accessed:$ $2018-04-26$.

[Jack, 2011] Jack, A. (2011). Emergent systems as a narrative device. https://www.slideshare.net/alanjack/emergent-systems-as-a-narrative-device. Accessed: 2018-04-26.

[Kwak and Blackburn, 2014] Kwak, H. and Blackburn, J. (2014). Linguistic analysis of toxic behavior in an online video game. In *Proceedings of the International Conference on Social Informatics*, pages 209–217. Springer.

[Lazzaro, 2004] Lazzaro, N. (2004). Why we play games: Four keys to more emotion without story.

[Lim and Reeves, 2010] Lim, S. and Reeves, B. (2010). Computer agents versus avatars: Responses to interactive game characters controlled by a computer or other player. *International Journal of Human-Computer Studies*, 68(1):57 – 68.

[Mateas and Stern, 2003] Mateas, M. and Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama. In *Proceedings of the Game developers conference*, volume 2, pages 4–8.

[McLaughlin, 2013] McLaughlin, M. (2013). New gta v release tipped to rake in £1bn in sales. https://www.scotsman.com/lifestyle/gadgets-gaming/new-gta-v-release-tipped-to-rake-in-1bn-in-sales-1-3081943.

[Minoraxis, 2011] Minoraxis (2011). Fantastic knight video game. https://itunes.apple.com/gb/app/fantastic-knight/id429852137?mt=8. Accessed: 2018-04-26.

[Mixamo, 2008] Mixamo (2008). Mixamo. http://www.mixamo.com. Accessed: 2018-04-26.

[Myślak and Deja, 2014] Myślak, M. and Deja, D. (2014). Developing game-structure sensitive matchmaking system for massive-multiplayer online games. In *Proceedings of the International Conference on Social Informatics*, pages 200–208. Springer.

[Märtens et al., 2015] Märtens, M., Shen, S., Iosup, A., and Kuipers, F. (2015). Toxicity detection in multiplayer online games. In *Proceedings of the 2015 International Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6.

[Paul et al., 2010] Paul, R., Charles, D., McNeill, M., and McSherry, D. (2010). Mist: An interactive storytelling system with variable character behavior. In *Proceedings of the Joint International Conference on Interactive Digital Storytelling*, pages 4–15. Springer.

[Paul et al., 2009] Paul, R., McNeill, M., Charles, D., McSherry, D., and Morrow, P. (2009). Real-time planning for interactive storytelling. In *Proceedings of the 9th Irish Workshop on Computer Graphics*, pages 89–94.

[Poulter, 2009] Poulter, S. (2009). Smarter games, dumber children. http://www.news.com.au/technology/smarter-games-dumber-children/news-story/a4e43cabc1 e5805e56eb59c666bf2e39.

[Roth et al., 2012] Roth, C., Vermeulen, I., Vorderer, P., and Klimmt, C. (2012). Exploring replay value: shifts and continuities in user experiences between first and second exposure to an interactive story. *Cyberpsychology, Behavior, and Social Networking*, 15(7):378–381.

[Rouse and Illustrator-Ogden, 2000] Rouse, R. and Illustrator-Ogden, S. (2000). *Game design theory and practice*. Wordware Publishing Inc.

[Schell, 2014] Schell, J. (2014). *The Art of Game Design: A book of lenses*. AK Peters/CRC Press.

[Steam, 2018] Steam (2018). Steam stats. http://store.steampowered.com/stats/. Accessed: 2018-04-26.

[Tassi, 2016] Tassi, P. (2016). Gta online's $500m in microtransactions could mean a very different 'gta 6'. https://www.forbes.com/sites/insertcoin/2016/04/14/gta-onlines-500m-in-microtransactions-could-mean-a-very-different-gta-6.

[ThatGameCompany, 2012] ThatGameCompany (2012). Journey video game. http://thatgamecompany.com/journey/. Accessed: 2018-04-26.

[Tomlison, 2013] Tomlison, P. (2013). Game design document.

[Valve, 1998] Valve (1998). Half-life video game. https://store.steampowered.com/app/70/HalfLife/. Accessed: 2018-04-26.

[Véron et al., 2014] Véron, M., Marin, O., and Monnet, S. (2014). Matchmaking in multi-player on-line games: studying user traces to improve the user experience. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 7. ACM.

[VGChartz, 2017] VGChartz (2017). Global yearly chart. http://www.vgchartz.com/yearly/2017/Global/. Accessed: 2018-04-26.

[Volk, 2016] Volk, P. (2016). League of legends now boasts over 100 million monthly active players worldwide. https://www.riftherald.com/2016/9/13/12865314/monthly-lol-players-2016-active-worldwide. Accessed: 2018-04-26.

[Wolfram, 1983] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601.

[Yee, 2006] Yee, N. (2006). Motivations for play in online games. *CyberPsychology & behavior*, 9(6):772–775.

[Young and Riedl, 2003] Young, R. M. and Riedl, M. (2003). Towards an architecture for intelligent control of narrative in interactive virtual worlds. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 310–312. ACM.