Creation of a demo of a Tactical Role-Playing game with dynamic AI

by

Óscar Gil Ferrer

Degree: Video Game Design and Development
Course: Bachelor's Thesis
July 2, 2019

Supervised by:
Pedro José Sanz Valero

To everyone who believed in me

Acknowledgements

I want to thank my parents for being there through thick and thin. I want to thank my friends for encouraging me to finish the project. I want to thank Miguel Chover Selles because without him, this degree would not exist. And lastly, I want to thank Pedro José Sanz Valero for supervising and approving the development of this crazy project.

Abstract

This document details the Technical Report of a Bachelor's Thesis consisting in the development of a demo of a Tactical Role-playing game in 3D using Unity. The basis of the project is a simple turn-based strategy system together with a decision-based system previous to the combats. The efforts of the project are put mainly on the research and development of Artificial Intelligence techniques so as to provide the game of a dynamic AI.

This will be done by implementing three distinct systems: a Visual Novel-like system that will take into account the decisions made by the player to decide which scenario will be played right after; an Emotion Controller to adjust the behavior of the different computer-controlled units according to the situation of the battlefield; and an Error System that will take into account previous inefficient actions of the units to diminish their value during the decision-making portion of the AI.

Keywords: RPG, Strategy, AI, Personality, Learning

Contents

List of Figures

Chapter 1

Technical Proposal

1.1 Introduction

The Golden Age of Tactical Role-Playing Games (referred as TRPGs henceforth) has long come and gone. Recent TRPGs do nothing more than capitalize on older franchises without bringing much new to the table. There are exceptions to the norm, like the Firaxis reboot of the X-COM franchise [1], the The Banner Saga duology [2] or even the last non-remake entry of the Fire Emblem franchise [3].

It is true that these games can boast robust mechanics on combat, storytelling, management, unit customization and a long et caetera. But there is one thing that most TRPGs do not pay attention to even though it's one of the most important aspect in a "Tactical" game: its AI.

On most TRPGs, the enemy units limit themselves to attacking the nearest enemy without really thinking if attacking is even the best available option. If there is an enemy within its search area, it will invariably move towards it as long as it feels it's superficially secure to do so. And by superficially I mean that they look if they have enough ammunition, magic points, bullets, etc and if they have enough health to not be put in some kind of 'danger' state. It doesn't really matter if the enemy has some kind of counterattack that will surely obliterate the unit as soon as it tries to attack.

On the other hand, it seems logical that the unit does not really know what the enemy is capable of at first glance, and while this is true, the AI doesn't learn anything and keeps getting countered by the same unit over and over, while other units wait for their turn to be countered as well. This may seem like an extreme example, and the truth is that in general those games have enough of an AI to keep the players entertained.

But I have to ask: is this really enough? Is there nothing that can be done to better these AIs? That's where this project comes in. The goal of this project is to try to imbue the AIs of TRPGs with two crucial characteristics: basic learning, and personality.

The demo will be separated in two parts: The story part, and the combat part.

The story part will consist of text over static images relating the overarching story of the game and some choices that will affect the initial situation of the combat part. This choices will affect mainly on the "personality" part of the AIs. An option may demoralize the enemy so that in the combat part most enemies will try to flee while a small band will cover their retreat while another choice may involve the enemy having the jump on the player units, turning the tables on the aforementioned case in that now the enemy will try to cut any escape possibility for the player.

The combat part will take place in a grid-like environment with the camera offering an isometric perspective. The position of the enemy and ally units within the grid will depend on the choices

made in the story part. Once the units are in position, a unit will take its turn depending on its Agility stat, and following that order, from highest to lowest. On each turn the unit will be able to attack, use an ability if able, move, or wait. Each of these actions is measured in tiles, to know their range. This part will end once one of the two sides has completed their goal. If the enemy completes their goal first is a game over, while if the player completes its goal first, the demo will end on success. The demo will have a short story part and two different combats.

1.2 Related Courses

- VJ1227 Game Engines
- VJ1231 Artificial Intelligence
- VJ1218 Hyper Media Narrative and Video Game Analysis

1.3 Goals

- The creation of a malleable AI system that allows for basic strategies or changes depending on story choices and character personality.
- The creation of a seemingly logical behavior for enemy units when exposed to the opponent party's capabilities.
- The creation of a coherent storytelling structure.

1.4 Scheduling

1. Writing of the technical proposal 5hrs
2. Writing of the GDD 15hrs
3. Implementing the base combat gameplay 30hrs
4. Investigating AI techniques 45hrs
5. Implementing AI techniques 90hrs
6. Joining AI techniques with the storytelling 45hrs
7. Testing and revising 30hrs
8. Writing of the project's final report 40hrs

The tasks have been planned in general terms by approximating how many hours SHOULD be allocated on each task, and each task is dependent on the one before it. This is a one-man project, and as such every task will be accomplished by the author of the thesis. Next I will detail the tasks as much as possible.

Writing of the technical proposal The technical proposal is the pitch of the project to the supervisor, and as such should be short and concise, and while assessing the difficulty to make a viable project is an important and costly part of it, no more than 5 hours should be put on this task, as it is only the 'presentation' of the project.

Writing the GDD The GDD or Game Design Document is the document that shows all the details of the game you are trying to develop, and is a continuation of the technical proposal. The technical proposal is centered in the part that is the meat of the thesis (the innovative parts of whatever game you are developing), while the GDD is centered in what is the skeleton of the thesis (the genre, rules and components of the game around which you are developing the mechanics detailed in the technical proposal). This document should not be too verbose either, but should include every detail possible of how the game is formed (screens, how the player wins or loses, controls, objectives, narrative, etc), therefore, allotting it 15 hours, triple the time for the technical proposal, seemed appropriate.

Implementing the base combat gameplay As the base system of the thesis is from a tutorial, it is necessary to allocate time to follow each step of the tutorial and implement it myself so as to understand the ins and outs of the tutorial, rather than simply downloading the made project and start from there. The tutorial is quite lengthy, covering everything that needs to be known about the system, and showing the code little by little, as such, I allotted 30 hours, mostly due to having to understand the tutorial and not follow it blindly.

Investigating AI techniques The theoretical approach to the thesis is the first thing done, by explaining it in the technical proposal, however, concrete AI techniques have not yet been defined. Machine Learning is one of the techniques that is going to be used, but how? What kind of learning? Will the emotion system be implemented using a State Machine? Do I want to delve into more complex techniques or should I implement it in the simplest way possible? These questions need to be answered before even trying to enter into coding territory. 45 hours may seem like a lot, but there is much theory to be investigated.

Implementing the AI techniques Pretty straightforward. Most of the time will be dedicated to implementing the aforementioned techniques. Even so, this timeslot can be very variable depending on the complexity of the techniques, but even so, to ensure that everything works as it is supposed to, while taking into account any unforeseen circumstances, almost a third of the total time is more than a fair amount of hours.

Joining the AI techniques with the storytelling This is also one of the most important parts of the development. The final goal of the thesis is to see how a better AI can enrich the narrative and help to better immerse the players in your game. As such, the techniques need to make sense in the narrative. Not only that, but the implementation of the story part is also included in this slot. Fungus is pretty simple to use, so really, most of the time will be spent on preparing the scenarios, preparing the patterns for the AI, preparing the paths that can be taken and implementing the skills that the units can use. 45 hours may be a stretch but more or less hours can be taken or given depending on the time used on the previous slot.

Testing and revising This slot not only includes testing to make sure the game does not break at any point, but also testing with other players to get results for the conclusions of the thesis. That is, if the project accomplishes its objective to help with the narrative. As before, this slot is very variable due to unforeseen bugs and critical errors than can occur after the implementation is done.

Writing the project final report In the end, all the tasks, all the time allotted at the end of the project, all the research, explanations, etc, has to be compiled on a single document that has to contain every detail possible of the development of this thesis, and made to be understandable by anyone who tries to read it even if they have no idea of Artificial Intelligence. At most it has to occupy 50 pages, which is quite hefty, and also, after the preliminary evaluation, I will have to add whatever the tribunal deems is missing. As such, 40 hours seems like a fair amount of hours.

1.5 Tools

- Unity 3D: The Game Engine used to create the game. [4]
- Fungus: A Unity add-on to help create visual novel-style storytelling. [5]

1.6 Expected Results

By the end of the project, the demo will consist of an opening narrative part with two choices, and two different complete combats with opening narrative and different composition of enemy teams. If the player loses all their units or kills all the enemy units, the demo will end.

Chapter 2

Game Design Document

2.1 Overview

2.1.1 Main Concept

This is a turn-based Tactical RPG with Visual Novel style narration in which the combat occurs in an isometric grid with several levels of height. Both the AI and the player will be able to move their units in turns while using specific abilities of each unit with the goal of completing the objective of the level, which depends on the decisions made in the story part previous to the combat.

2.1.2 Unique Selling Points

- Dynamic AI capable of simple learning and basic strategy patterns, discouraging individual actions
- Personality-based AI influenced by decisions made on the story part
- The ability to determine the approach to the battle before it even begins through decisions made on the story part

2.2 Gameplay

The gameplay is separated into to clearly distinct parts, a Visual Novel-style part, and the combat part.

2.2.1 Story Part

This is the part that takes place immediately after starting the game and will depict the backstory of the game, and offering the player some decisions that will have impact into the combat part of the game. The player will advance through the text by clicking anywhere on the screen and when the decisions appear on the screen, the player will be able to click on the desired option.

2.2.2 Combat Part

Once the story section finishes after advancing through all the text, the combat part begins. On this part, units are disposed on a grid, with its positioning depending on the choices made before, for example, if the player chooses to ambush the enemy, the player units will begin surrounding the enemy units, while if the player gets ambushed, the roles will be reversed. Once the units are put on the grid, the objective of the combat will appear on the screen. As before, the objective depends on the decisions made before. Ambushing means the objective to win the game would be to eliminate all opposing units, while being ambushed would mean the objective would be to break the enemy barricade and flee the scene to win.

After that, each unit's turn will alternate depending on their speed stat. On each turn the player will be able to:

- Move the unit: Once selected, colored tiles on the grid will mark where the unit is able to move. Selecting a colored tile will make the unit move to that location.
- Attack with the unit: Same as with moving, colored tiles on the grid will mark the attacks range. Once a valid enemy is selected, the unit will deal damage to the other unit.
- Use a unit's special ability: Almost the same as attacks, except they may be able to target enemies, allies or the unit itself depending on whether it is an offensive, or support ability.
- Defend with the unit: Ends the turn and the unit takes less damage until its turn rolls around again.

Different kind of units will have different kinds of move and attack ranges, as well a different skills. If the player moves the unit first, he will only be able to use the rest of the actions, while if the player attacks or uses an ability first, he will only be able to move afterwards.

After all this, the player will be able to choose the orientation the unit will have until its next turn. Instead of a 360º wheel, the player will only be able to orientate the unit north, south, east or west. This is important as attacking a unit from the sides or behind will affect their aim and damage, with attacking from behind being a guaranteed hit with a 120 percent damage multiplier.

All of these actions will be performed by pressing Z to accept, X to cancel, and the arrow keys to navigate the menus or to select tiles on the grid.

Units

Melee Units

Melee units are only able to attack tiles adjacent to them, with only one tile of range. Melee units' attacks have a height tolerance of 2, meaning they can only attack units that are at most, at 2 height levels higher than the unit itself. All damage dealt by melee units is considered physical.

- Paladins: Paladins are defensively focused and therefore their offensive capabilities are reduced, but with their ability Holy Cross, they can combine their magical stats with their physical stats to deliver a devastating blow, but with limited usage. Be careful though, as lots of enemies can overwhelm the Paladin even with its powerful defense.
- Scouts: Units focused on mobility and offense, their skill, Blind, allows for harassing tactics by reducing the enemy's ability to aim.
- Soldiers: It is a middle of the road class with average offense a defense and whose skill Air Blast is a weak ranged attack.

Ranged Units

Ranged units specialize in ranged attacks but not necessarily have a basic attack range of more than one, as seen with mages. Rangers damage is considered physical while mages' skill damage is considered magical.

- Rangers: Normal ranged units with a range of basic attack of 3 to 4 tiles around them. As such, Rangers cannot attack at melee range. Their skill Arrow Rain allows them to attack enemies in an area.
- Mages: Mages have the same attack range as melee units but reduced melee offense and defense. As such, they have to be protected against melee units. They have 3 kinds of fire magic with different ranges and will use them according to their strategy. They have a long range but weak Fire Dart, a medium range medium power Fire Ball and a close range but very powerful Fire Hands.

Stats

All units have the following stats:

- HP: Short for Health Points. How much damage a unit can survive before dying.
- PATK: Short for Physical Attack. How much melee damage can a unit deal.
- PDEF: Short for Physical Defense. How much can a unit mitigate physical damage.
- MATK: Short for Magical Attack. How much magical damage can a unit deal. As only mages can deal magical damage, this stat is irrelevant for all others.
- MDEF: Short for Magical Defense. How much can a unit mitigate magical damage.
- RES: The capacity of the unit to resist status ailments.
- DODGE: The capacity of the unit to be able to dodge another unit's attack.
- SPD: Short for Speed. How early will the unit's turn come around.
- MOV: The movement range of the unit.
- JUMP: The height tolerance for the unit's movement.

2.3 Story

2.3.1 Setting

The game will be set on a medieval fantastic era. The world has recently suffered a cruel war and is living its aftermath. The player will take control of an strategist that tries to bring its unit back to its country through enemy lines.

2.3.2 Core Mechanics

The meat of the project comes for the AI behavior of the enemy units. Given the different roles of each of the units, each one will try to act according to the circumstances of both ally and enemy units. Other TRPGs have individual strategies but don't dwell much on group strategies or the

fact that the enemy can counter its tactics. As such, enemy units will try to maintain formations according to their remaining forces, not repeat mistakes, such as that if a Ranger attacks an immune Scout, nearby Rangers will not waste their attacks on that Scout anymore.

On the other hand, each unit will also have Emotions. If the battle is going well for them, they may try to act more reckless while still being logical and adhering to their strategy, and be willing to take more risks. When ambushed, though, the enemy units will try to flee, and not pay that much attention to strategy for the most part while more level-headed units will try to cover the retreat of their allies.

Relation to Story

These mechanics tie to the story in that being more confident, more fearful, or try to flee will be affected by the decisions of the player on the Story Part of the game. If the player chooses to attack from the front, the enemy will act confidently, knowing that they can take on a little unit with no problem, but may start to act more cautiously once the player starts taking out on enemy units. The enemy units will take into account how many and of which type of units their unit is composed to form their strategy the same way a real person would.

2.4 HUD

2.4.1 Story Part

On the story part the player will see a background and plain text over it in a textbox that will mark who is talking and a conceptual image of the character talking (or none if it is narration)

2.4.2 Combat Part

On the combat part the player will see the battlefield from an isometric perspective. The battlefield will be divided in tiles in a grid pattern and both ally and enemy units will be spread on the grid. When attacking, a little textbox will appear with data about the enemy unit, such as how much damage will the unit deal to the other, and the percentage of success of the attack.

During enemy turns, little messages at the top of the screen will show what kind of action they have selected, and during allied turns, the player will be able to navigate a menu on the bottom right of the screen that shows the available actions for that unit on that turn. Unavailable options will be greyed out.

2.5 Screens

The game will only have 2 screens:

- Story Part Screen 2.1
- Combat Part Screen 2.2

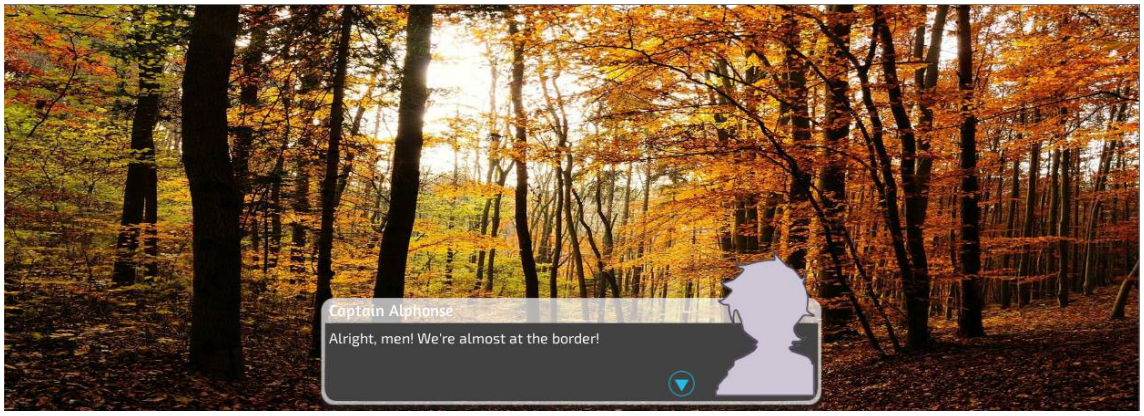

Figure 2.1: Screenshot of the Story Part

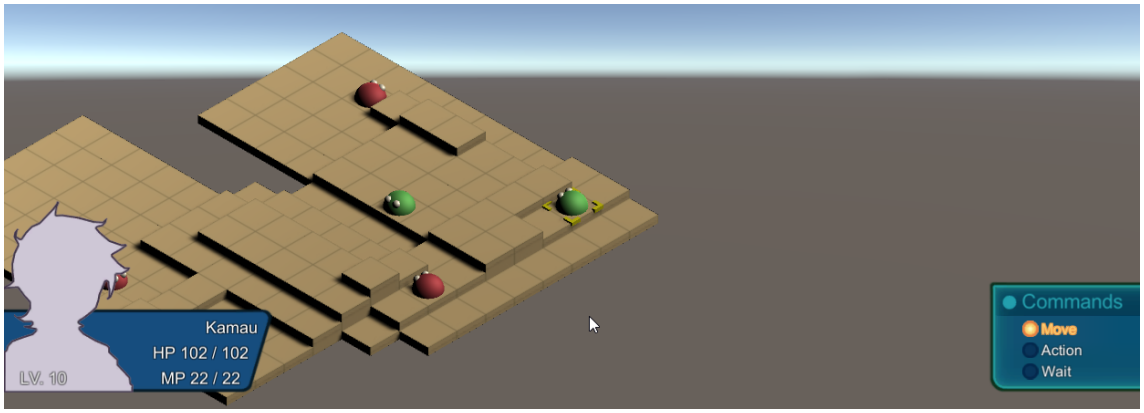

Figure 2.2: Screenshot of the Combat Part

Chapter 3

Basic Concepts for the Thesis

3.1 What is a Tactical Role-Playing Game?

First of all, the most important thing would be to define the genre of game that is the whole core around which the thesis will be build.

A Tactical Role-Playing Game or TRPG for short is a sub-genre on Role-Playing Games whose main focus is the combat, being usually turn-based and taking place on a grid. This system is inherited from pen and paper RPGs where maps are divided in grids for ease of movement options and positioning. As the name of the genre suggests, the games are mostly focused on small-scale skirmishes where tactics shine instead of more high-level strategies, like in grand strategy games like Europa Universalis [6] or Real-Time Strategy games like Age of Empires [7].

While normal RPGs try to focus on world exploration, TRPGs follow a more closed type of mapping (if any), like a graph, that serve as a background for the combat. Like normal RPGs, though, the combat rules are kept mostly the same, with the characters having stats, equipment, and abilities.

TRPGs mostly rose to prominence on Japan, where the most famous series of TRPG, like Fire Emblem [3] being born there. As such, most conventions from Japanese RPGs are also inherited with the combat actions being the same as on those games (using an action/attacking or defending), plus the ability to move and having the actions subordinated to its range. On the other hand, modern western TRPGs took the same approach as the Japanese ones, while its precursors were more free-form.

3.2 "Academic" AI vs Video Game AI

The term Artificial Intelligence is used to refer to the ability of computer programs to solve problems, as programmed by a human. However, this term is too broad. AI can be used for a myriad of things, from complex tasks like operating cars safely, to menial tasks like playing chess. As such, it is necessary to make a clear distinction between what would be considered as "academic" AI, and what we will be seeing in this thesis, Video Game AI.

"Academic" AI goals is for the program to make human-like decisions, being able to interact with the real world, learning through outside stimulus, automation of day-to-day tasks, on an optimal manner, even surpassing human capabilities. These things, however, are NOT the goal of Video Game AI.

The goal of Video Game AI is to develop companions and opponents to the player that are

able to think and decide, yet with all of the human flaws implemented. Players feel cheated if they have to confront an AI that has faster reaction speed that is humanly possible, so the general idea is to balance challenge and easiness so that, and this is the imperative word, the player feels **entertained**.

3.3 The Three Layers of Video Game AI

Millington and Funge [8] defined three layers on Video Game AI that define the process that the AI goes through to get the desired results. These three layers are:

Strategy. This layer concerns the algorithms that rule over the group behavior of the AI. Whether the opponents will adopt a more aggressive attack, a more defensive approach, try to take control of the terrain, etc

Decision-making. This layer concerns the individual actions taken by each of the computer-controlled actors by themselves, like attacking another actor, healing another actor, building, remain passive, etc

Movement. This layer concerns the positioning of the computer-controlled actors, with behaviors such as patrolling, running, chasing, remaining static, etc

In broad terms, the Strategy determines the mindset of the group, which will define the goal to be achieved by the actors. However, each actor has its own actions or may not be able to the same thing as another actor, as such, the Decision-Making arranges which of the actions possible is the one that helps in completing the overall goal, while Movement will dictate the best positioning and movement-related actions to accompany the actions of the Decision-Making.

3.4 Decision-Making: Reinforcement Learning

AIs are usually limited to whatever behaviour the developer has programmed, however, through some algorithms it is possible for an AI to learn how to solve a problem by itself through the use of examples. This is called Reinforcement Learning and is one of the approaches of Machine Learning.

This approach is the basis of the Error System of the thesis. By receiving feedback in the form of damage results, the unit bases its attacking options on which one is the best for the defending unit.

As can be seen, on turn 1, all abilities are initialized with a max value so as to have the AI try every single ability before settling on the best option. The AI uses Fire Dart on the Player Unit and it receives feedback, that is the damage done to the unit, and stores it in a pair (ability, target) as the key. Once the AI has received feedback on all the possible abilities, it will be able to decide which one is really the best ability to use on that target. This is a brute-force learning algorithm that differs on the fact that the AI does not predict the results and achieves the best option immediately, but rather slowly learns based on actual results, like a human would do.

3.5 Strategy: State Machine

A finite-state machine or State Machine for short, is an abstract machine that can only be one state at a time. In terms of AI, a State Machine can make an actor behave one way or another depending on the active state. For example, during Calm state, a guard may be patrolling on the same route over and over, but as soon as the state changes to Alert, the guard may start to chase the player and attacking it while in range.

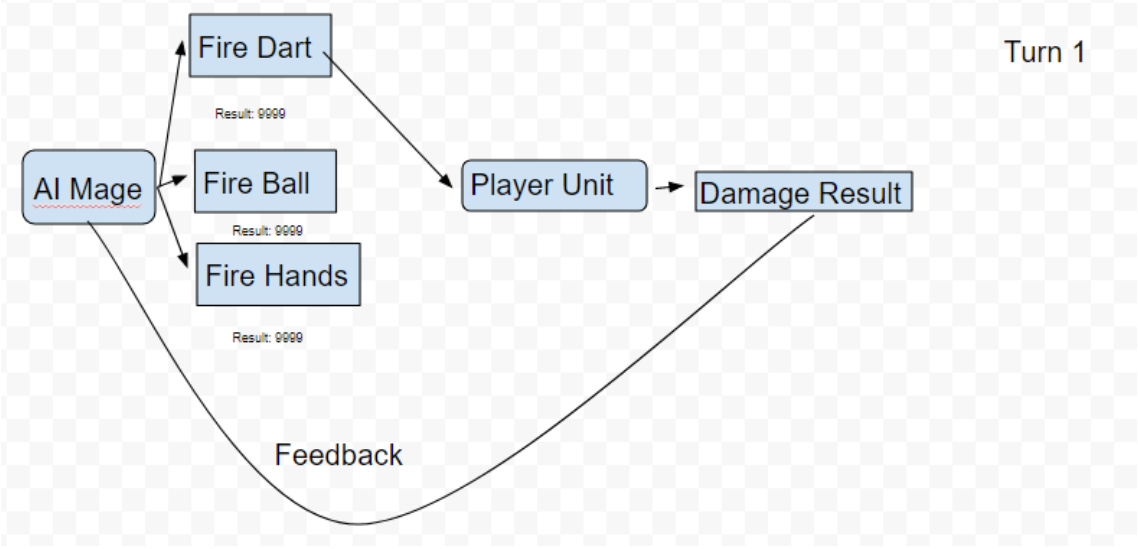

Figure 3.1: Feedback loop on the first turn of an AI Mage

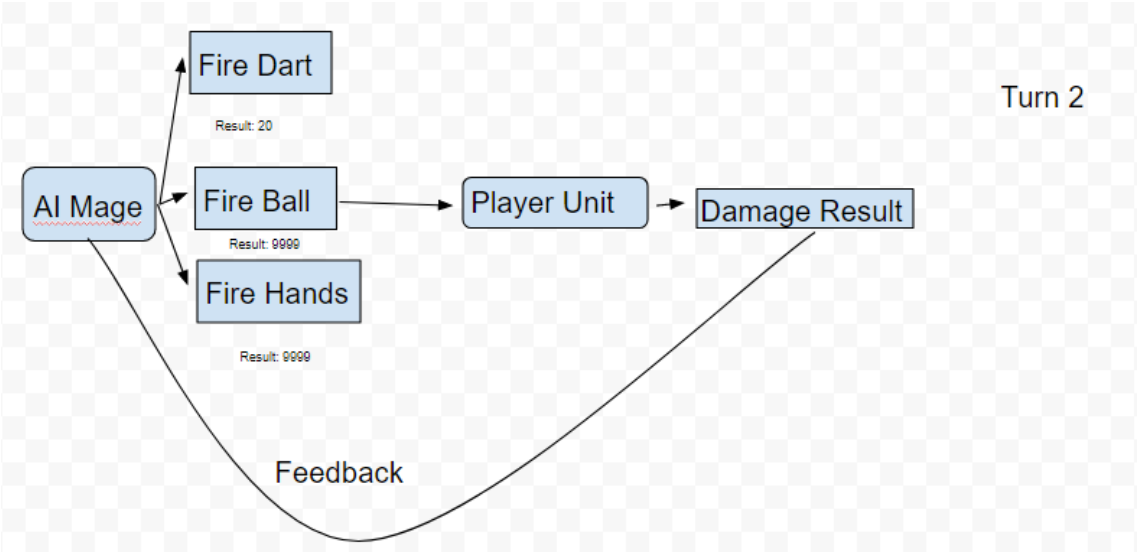

Figure 3.2: Feedback loop on the second turn of an AI Mage

In Video Game AI the State Machine has been deprecated a long time ago by the use of less error-prone systems, as real time may have an AI reacting to multiple stimuli at the same time, producing an overlap of states, or making it set a state that is not the correct one by some combination of stimuli that coincides with the trigger for an unrelated state.

However, in the ambit of this thesis, the turn-based system works in our favor for implementing a state machine that cannot possibly fail. In turn-based environments we can control which stimuli affects our AI actors and in which order making a State Machine a simple and viable solution.

As seen in the figure 3.3, this is how the Emotion System will work. At the beginning of each turn, the state machine will evaluate if its state should change, and if it does, the active attack pattern of every unit will change according to whatever emotion (state) is active at the time. In this example, the default attack pattern will make use of the Error System to learn as stated in the previous section, while the shaken attack pattern will make the AI keep learning but instead forgoing Fire Hands, recharging its Magic Points every three turns and even forcing the AI to use Fire Darts on some turns. The emboldened attack pattern uses a similar pattern, but instead forgoing the weaker attack and even recharging its Magic Points.

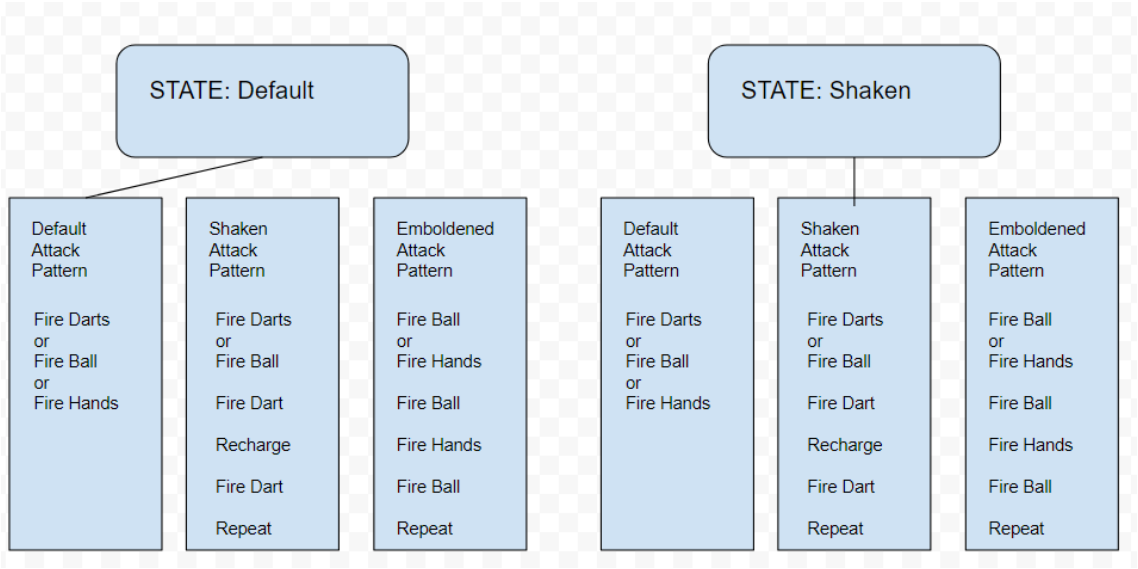

Figure 3.3: Two different States of the State Machine

The state check is done each AI turn due to the fact that the units get their turns according to their speed instead of each side getting to move every unit at the same time. As such, it is possible that from one turn to another, the state has changed.

Chapter 4

Narrative Development

For the development of the Story Part, we will be using an add-on to Unity called Fungus [5]. With this system, the objective is to tie the gameplay with plot and player decisions. At the start of the game, the player will be given background information through visual novel-style screens and at the end of that, will be given a choice, that will determine which map will be the one the player will play in. This is important because the player may choose to partake, following examples featured in the thesis, in an all-out attack versus an invasion force with a relatively small platoon, being outnumbered from the start, which thanks to the Emotion System detailed earlier, would make the enemy units feel emboldened as soon as the battle begins. On the other hand, the player may choose to cross through a less defended post, which will result in the ensuing battle being more fair.

4.1 About Fungus

First of all, since Fungus is not a default library for Unity, it would be appropriate to explain the basics of it so that the approaches can be better understood.

Fungus uses a flowchart system composed of blocks of commands. This commands include things like making a character say a block of text, showing a menu to select choices, change a character's sprite, change the background, change the music, and even call methods of a class. If a choice is used, a block can fork into other blocks, which can subsequently fork into other blocks, or the simply can advance into another block for ease of separation of commands.

These commands are easy to use with no need to program except for integration purposes, and can manage everything from a database of characters, places, etc for ease of reference, to the backgrounds, textboxes, sprites, etc. It is completely possible to make a functional visual novel game without having to code anything.

4.2 Technical Approach

This part needed no theoretical research, therefore we will jump directly to the technical approach.

The thesis, will consist of a single screen for the story part, with a menu with two choices that will go to two different scenarios on the gameplay part. As such, we will make use of three different blocks on the flowchart.

The Start block will have all the Say commands that show two characters, Captain Alphonse and Officer Roderick, speaking about the aforementioned background information. Once the characters

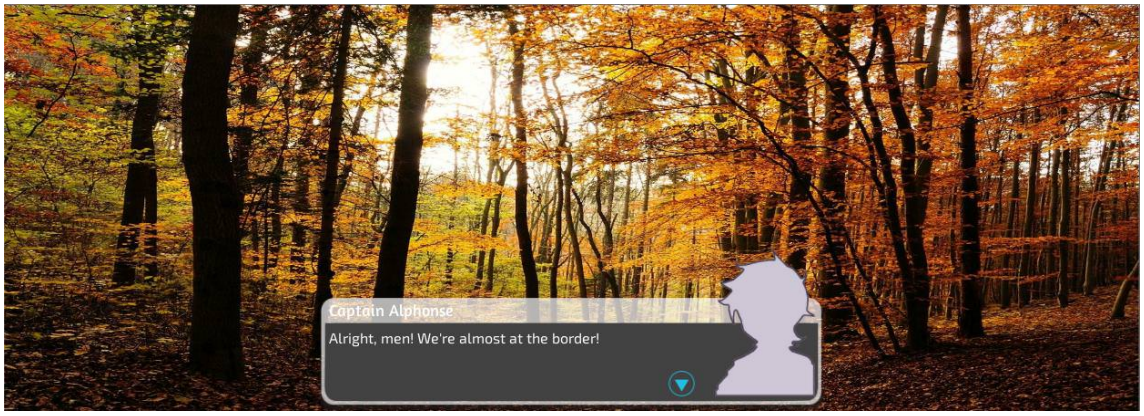

Figure 4.1: Example of a Fungus-made novel scene.

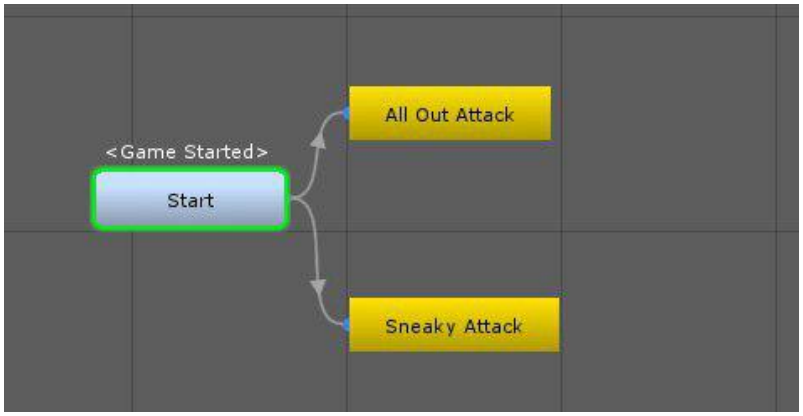

Figure 4.2: Flowchart of the thesis.

stop talking, two Menu commands will show the player two options: either participate in an all-out attack, or decide on a more sneaky approach.

On the Menu commands we can see at the end that the two other blocks of the flowchart are mentioned. This is to indicate which option will branch to each block. On each of those blocks is a single Call command that will call the method of a class called SceneTransition whose only job is to execute a LoadScene command on the two different scenarios.

4.3 Narrative Impact

It may not seem like much, but this bit of story keeps the players invested in the battles they have to fight after the narration, as it provides motivation for the survival of characters the player cares about.

This demo does not intend to come up with that much depth to their characters, but is more focused in showing the fact that the choices do affect the battles after that, as one of the battles is more difficult than the other.

This fact also ties with the AI, as each scene has a different configuration of enemies that thanks to the Emotion Controller depicted on upcoming sections will determine different personalities for the units involved in the battle.

The two choices involve ambushing a less defended point to cross the border so that the units can return to their country or try to attack the main battalion that wants to invade to weaken it while trying to cross the border. The first choice involves a combat with fewer units while the second

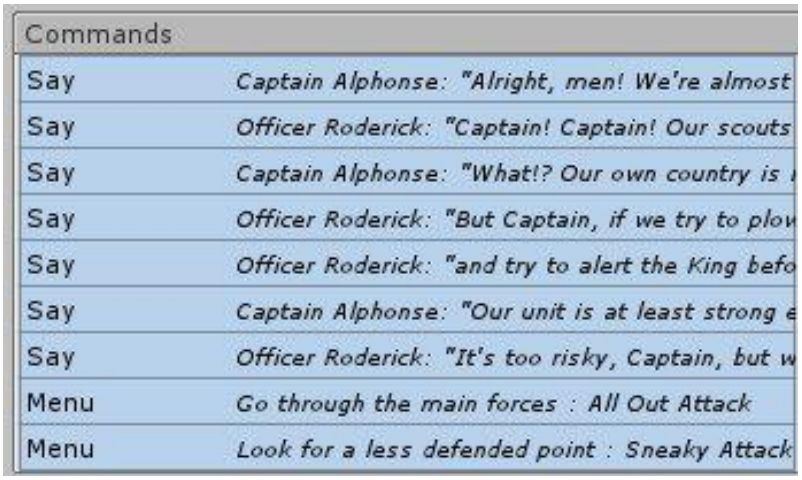

Figure 4.3: Commands on the Start block.

choice will involve a more difficult combat with more units and a different board composition.

Chapter 5

Tactics RPG System

5.1 Base of the system

This project has made use of a tutorial [9] to implement a robust Tactics RPG system based in the game Final Fantasy Tactics Advance [10]. This system is very customizable and makes use of very modular parts so that the user can add classes, abilities and the like without much trouble.

5.1.1 Capabilities of the engine

First of all, it would be necessary to explain what the provided engine in the tutorial can and cannot do, so as to ascertain the work done by the student. The following lists will itemize terms related to the thesis work and not everything the engine can do as it is not important to the work done.

5.1.2 What the engine CAN do

- Generate custom maps
- Generate character classes and abilities
- Interchange control between AI and player between turns
- Control communication between classes with a "Notification Center"
- Make the AI follow a strict attack pattern
- Calculate best target of an attack based on hit percentages

5.1.3 What the engine CANNOT do

- Have menus outside of the main game (no title screen, map selector, etc)
- Have the AI make its own decisions based on the state of the game
- Have different kinds of patterns for the AI that can be changed on the fly
- Have the AI evaluate its decisions once made and store the results
- Have a narrative component intertwined with the gameplay

5.1.4 Conversation Manager

This system also includes a Conversation Manager so that the developer is able to insert a conversation as an intro to the battle, as a defeat conversation, and as a victory conversation. Unfortunately, it does not support choices, and it is loaded after the board and units, so is not able to influence the disposition of the board and units, therefore the need to use Fungus to be able to implement choices.

Nevertheless it is very useful for adding more flavor to the battle.

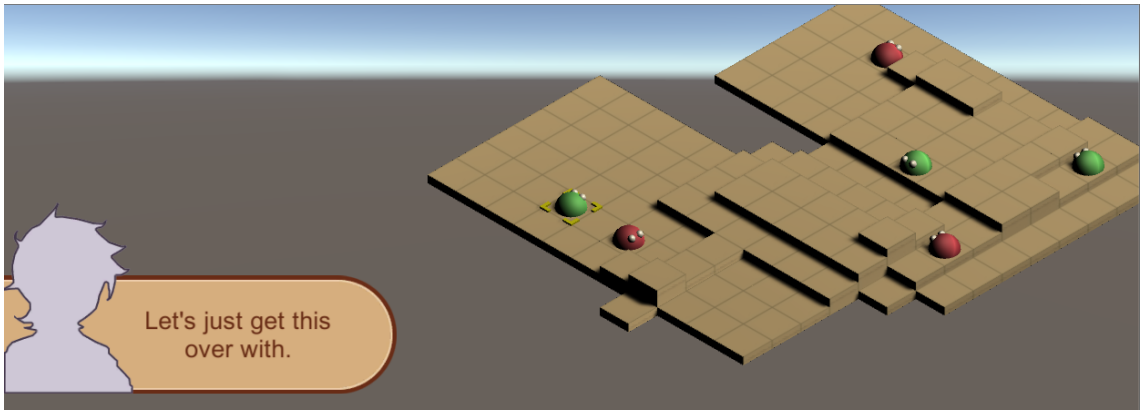

Figure 5.1: Screenshot of a sample conversation using the Conversation Manager

5.1.5 AI in the Base System

The creator of the tutorial wanted to make a kind of "dumb" AI, but not dumb in the sense that it makes dumb decisions, but in the sense that it cannot foresee what the player may do by simulating future turns, but rather, it plays by the actual turn, without much regard for consequences as long as it has the best present result. As such, it may take actions that may be seen as "dumb" in the long run.

I am going into detail of two important parts of the AI system that I tinkered with to implement the Emotion Controller and Error System.

Ability Pickers

The system makes use of Ability Pickers to determine which abilities the AI can use. This objects have as children which abilities the unit will use in a determined order, and it has two types of use, a fixed picker, where the unit uses only the determined ability, and a random picker, that uses the abilities assigned into that picker a random. For example, let us use a mage's pattern:

- Fixed: Fire Ball
- Fixed: Fire Hands
- Random: Attack or Fire Hands

According to this set of pickers, on the first turn, it will use a Fire Ball, the second turn it will use the Fire Hands, on the third turn it may use a normal Attack or the Fire Hands again, on the fourth it will return to the beginning, using Fire Ball, etc

Plan of Attack

The Ability Pickers determine "what" the unit will do, but the Plan of Attack will determine "how" will it use this ability to maximize its effect. As different abilities have different ranges and areas of effect, and may even be angle dependent, an algorithm will determine the best targets depending on distance, angle, number or enemies caught in the area, etc and will fill the Plan of Attack with the information relative to the tile it will select as target in the end (as there may be multiple equally valid targets, they are selected randomly).

5.2 Implementations for the project

5.2.1 Abilities

The abilities that came with the tutorial are mostly copies from the abilities from Final Fantasy Tactics Advance, and I had to implement my own to use in the project. The combats of the thesis, for ease of demonstration, will feature only Soldiers, Scouts, and Mages. Most of the used skills are already implemented except the three levels of fire magics for the Mage.

5.2.2 Minimum Range

I had to make two searches, one from the unit to the minimum range, and the other from the unit to the maximum range, and then, return the difference of both searches, thus having a list with only tiles from the minimum range to the maximum range.

5.2.3 Conversations

Obviously, the generic conversation included in the tutorial was not appropriate for the setting and atmosphere of the project, so I had to modify all three (intro, defeat and victory) conversations. This was very easy as the scripts modify the editor to create ConversationData, that are assets that contain the conversation you want to be shown.

5.2.4 Units

The generic units provided by the tutorial are one of each of a rogue, warrior and mage, but these did not comply with the idea of units that I had in mind, therefore I had to compile all the implemented abilities, stats and movement types into the appropriate units that are stated in the Game Design Document (referred henceforth as GDD). This was also trivial to do as it did not require any script tinkering, only creating Unit Recipes (which is the way the system has to create units to put in-game).

5.2.5 AI

The meat of the project, the AI was the main focus of my programming efforts to implement ways of giving personality to the units and have them "learn" without having to resort to complex algorithms like Machine Learning, and when studying the tutorial I found that the system lent itself to be modified for my two main ideas.

Error System

The first system is giving the AI the ability to evaluate the effectiveness of their attacks after the action is done.

Theoretical Approach The theoretical approach to this system is pretty simple. When it is the AI turn to attack, the AI will have stored the results of past interactions of their attacks on the target.

Ex.: Let us say a Paladin attacks a Mage with a magic-based attack. The Mage, having higher magical defense, will receive little damage, which will be reflected on a result table. On the Paladin's next turn, it will check the table and notice that the magic-based attack has a lower score than the other attacks, so he will chose another attack instead of attacking with the same one again.

This way the AI "learns" by combining elements of dumb AI (following a pattern without questioning whether or not is the best course of action) and smart AI (simulating thousands of games and always choosing the best course of action). Smart AIs can be considered too unfair, as it will never make a mistake, while dumb AIs may be considered too easy as they will never break out of their established pattern. Combining both AIs makes for an AI that starts dumb but becomes smart the more the game advances, making for more interesting match-ups.

Technical Approach The technical approach to the system, however, is not so simple. The AI provided by the tutorial can evaluate which target is the best AFTER the attack has been chosen, but after careful consideration, due to stipulations of the thesis, it has been deemed unnecessary to do those calculations, as the classes presented in the thesis are not able to target more than one foe/ally or have fixed areas of effect.

However, the calculations of the tutorial include the code to check if any target is in range before and after moving, which would be impossible to calculate if the attack has not been chosen yet.

Therefore, the best solution would be the following:

The tutorial uses a class called Ability Picker to choose an ability to use in that turn. The class in itself is pretty simple. It gets an ability name and looks for it in the unit, and if it does not find anything, it selects the first option of the abilities list, which is a melee attack.

This class, however, is only an abstract class, and it has a method that has yet to be implemented, the Pick method. To show how it works and how it should be implemented to work on the thesis, let us have a look at one of the already implemented Ability Picker concrete classes.

The Fixed Ability Picker, simply selects the Ability that has been passed as a parameter and looks for it in the unit in question, and then, adds it to the PlanOfAttack.

While this class may seem unnecessary, it adds a new parameter to the PlanOfAttack, the target. As not all the abilities target enemies (such as healing or buffs, or even abilities that can only target the unit itself), and leaves us room in the Pick method so that we can choose which abilities we want to pick for that turn, which brings us to how are we going to implement the Pick method to select which ability we want to use based on previous experiences, but first, we need to look at another method to understand the two-part approach to the problem.

The Evaluate method on the ComputerPlayer class is the method that evaluates the best targets for the ability once it has been chosen. As said earlier, this evaluations are mostly redundant due to the lack of abilities in the thesis that make use of those calculations, but is necessary due to the fact that it can calculate where the ability can hit before and after moving the unit without having to actually do so.

Further down is the most important to our calculations, PlanDirectionDependent, as it is the one that searches every possible target tile and looks for all of the possible targets of the ability. The RateFireLocation method just adds a mark to the location if it contains a valid target for the ability so that the PickBestOption does not have to brute-force again. This method evaluates the HitRate for each valid mark and puts on a list the ones with the higher attack rate and in the end selects one of the targets at random from the final list, since all of the targets on the list share the same HitRate.

Now that we know how the engine works, its time to show how are we going to modify it so that it adds another layer of evaluation.

First things first, every Unit object should be able to store pairings of Ability-Target-Value, therefore, we will add a List of structs to every Unit, which will be initialized as empty. As it is not known whether a Unit will ever use an Ability over a certain Target, it is best to initialize the structs as they are used.

Next, we will create the class ConditionalAbilityPicker. This Picker will create a PlanOfAttack for each of the abilities included in the Picker and once completed, will compare the struct of each one and retain the one with the highest damage value. In case of a draw, it will pick the last one.

On the comparison part, the method will look for the structs on the caster's list, and if a struct of the combination is not found, it will add one to the list, with an initial damage value of infinity. This is made so that the unit tries its other abilities instead of always picking the first one tried, as it would do if the damage value started at 0.

The difficulty in implementing this part comes from the fact that this new Picker has to change parts of other methods that could start a chain reaction where the other type of Pickers stop being compatible with the new code, which is not what we are looking for. On the contrary, as stated in the next section, we make use of all of the available Pickers.

Final Approach In the end, as seen in the Pick method the final result did not diverge that much from the Technical Approach, except for the fact that Pair is now a class, and implemented an Equals method that only compared its ability and target and not its damage. This is done because due to the generality of the ArrayList, one cannot modify the members of the list, therefore, we need to create a new pair with the updated damage value, remove the previous one, and add the new one to the list.

Emotion System

This system will make the AI units change patterns to adapt to the circumstances of the battlefield.

Theoretical Approach This system adds an upper layer to the previous system. While the "error system" only depends on the own results of each unit by itself, the "emotion system" evaluates each turn the state of the battlefield (in the case of the thesis, number of units per side), to choose one of the three attack patterns available to each AI unit.

This has been done to apply more flexibility to the AI of the game. Now it will be able to change attacks based on results, and change entire patterns based on the state of the battlefield. The "emotion" conditions will be the same for every Unit, but the strategy of each attack pattern will depend on the type of unit.

For example, if a mage sees that it is being outnumbered, it will resort to skills with a larger range even if they do less damage. In the specific case of the thesis, the new attack pattern will forgo the lesser ranged skills such as Fire Hands (2-tile range) and will use Fire Ball less frequently (5-tile range), while making Fire Darts its main source of damage (7-tile range).

On the other hand, a warrior will make use of more aggressive tactics in its pattern as a last-ditch effort to turn the tide of the battle. Following the previous example, in the thesis that would mean that while the default attack pattern will use some of its turns to defend or recover with a spell, the new attack pattern will make sure that every turn is dedicated to either attacking physically or using an offensive skill.

Technical Approach On the previous system we saw that on the Evaluate method, the first thing done is assigning the AttackPattern of the unit to a variable. As it is now, the method will pick the only AttackPattern available in the prefab. When we add the other two attack patterns we will need a test to see which should be the active pattern.

To do so, first we need to name the patterns appropriately, therefore, the three patterns will be called "default Attack Pattern", "shaken Attack Pattern" and "emboldened Attack Pattern". The next step would be to call the check before assigning the pattern to a variable. This check will traverse the unit list checking the Alliance field of each one and counting how many of each Alliance there are. If there are double or more player units than enemy units, the check will return the string "shaken", if there are double or more enemy units than player units, the resulting string will be "emboldened" and in any other case, the method will return a null.

This string will now be used twice. First, the method will show a message on the screen that says "[UnitName] feels [emotion]" to notify the player of the change in pattern for the unit. Then, after that, it will be used to check for the attack pattern by name, by using it as an addendum to the transform.find() method. That is, unit.transform.find(string + " Attack Pattern"). As a failsafe, the method will look directly for the default attack pattern and show "[UnitName] feels normal" instead of "[UnitName] feels default" if for any reason the string is a null. Also, the unit will store its last state and will check if the new state is the same as the last state and in case of being the same, will not show the message to the player, but if it is not the same, aside from showing the message, the method will reset the position of the index in the AttackPattern, making it start again on the new pattern.

Final Approach In the end, this part was also kept mostly the same as stated in the Technical Approach. Every turn, before the Evaluate method selects an Attack Pattern, the method EvaluateState is called to look how many of each Alliance is present on the board, and when done, assigns a state. The extra step that had to be taken was changing the UnitRecipe class so as to accommodate all three Attack Patterns, and changing the UnitFactory class to instantiate all the patterns without duplicating the Driver component on the unit object. Also, due to the Attack Patterns being so simple, the index is not reset, as it would not have virtually any effect on the game.

5.3 Narrative Impact

As stated in the Technical Proposal, most other TRPGs do have some semblance of personality, but it does not vary as the game progresses, leading to some dumb decisions, and are not able to adapt their Emotions to the current setting of the battle.

Also, most AIs fall on two camps. Too smart to commit errors, or too dumb to learn from them. Too smart AIs feel unreal, as they should not be able to predict how much damage they are able to do before doing it (on a side-note, this is something the players need, though, to not feel like their efforts are wasted testing on who they are able to do more damage), while too dumb AIs feel easily exploitable and not much of a challenge.

These changes could be completely internal and not show any visual cues, making the player none the wiser, but I feel that the feedback helps the player feel like their actions have consequences on the overall battle other than simply diminishing the opponents capabilities by maiming or killing them, and also that the AI is not that easily exploitable, making them rethink their strategies.

The central idea of the Emotion and Error Systems is to provide the AI units with a stronger personality, being able to respond to their own actions, evolving from dumb to smart given enough time, and having human-like feelings that forces them to make decisions based on whether they are alone or in a group.

These systems may have no bearing in the story itself, but they do not need to. Turning the AI units into something more than targets to kill is the real objective of the systems, thus helping with the emergent narrative and strengthening the overall impact of the story on the player, making them question their own decisions more than just trying to reach the ending.

5.4 Flow of the AI

So, with the basis of the thesis and the in-depth explanation out of the way, and as a summary of everything explained, let us see how the flow of the AI of the thesis works with in-game examples.

In this first test scenario, we will have a Player Unit (called Alaois) and an AI-controlled Mage with their starting positions one next to the other, and Fire Darts will have a full 7-tile range instead of 5-7. Also, the attack pattern for the mage will only include the ability the use either Fire Dart, Ball or Hands depending on the feedback.

To begin, once the AI gets its turn, a call to the Evaluate method will be made, to know what actions are present on the Attack Pattern of the active unit. Once the method retrieves the Attack Pattern, it will find that the action to be taken is to either use Fire Dart, Fire Ball or Fire Hands, governed by the Conditional Ability Picker.

The Conditional Ability Picker works by selecting a target and movement position for each of the abilities selected in the picker. For each of these abilities, it will compare it with the actual maximum value (initialized at 0), and replace its maximum with the value of the ability if it is higher or equal. Once all, in this case three, abilities have been tested, the one with the highest value will be the one executed.

In this example, all three abilities will have an initial value of 9999, so the last skill tested will be the one executed. Here, it is Fire Hands. Once selected, the unit will move to a random tile from where it is able to use the ability (in normal cases it would move to the tile that gives the unit the best hit rate, but since magic has a 100% hit rate, the tile selected is random, as all of them have the same hit rate), and then use it.

Once this is done, the OnApply method on the DamageAbilityEffect will be called. This class is the one that calculates the damage done, and also stores that damage into the unit's paired list (the list that stores pairings of ability, target, and damage done on the last use). In this case, 11 damage has been done, and the pairing "Alaois - Fire Hands - 9999" will be updated to "Alaois - Fire Hands - 11". Then, the unit positions itself facing Alaois, and its turn ends.

On its next turn, the comparison will boil down to using only Fire Ball or Fire Darts, since Fire Hands has a value of 11 for that target, while the other two still have the initial value of 9999. This time Fire Dart gets selected and the same process applies, moving to a random tile and using the ability. Fire Dart's pairing value will be 4, as it is the damage done to Alaois this turn.

Finally, on its third turn, the mage will use the only untested ability yet, Fire Ball, which deals 8 damage. With this all 3 abilities have been tested and have proper values. As such, in its fourth turn it will compare the three values obtained on the previous turns: Fire Hands' 11, Fire Dart's 4 and Fire Ball's 8. The winner is Fire Hands value of 11, and that means that the AI unit has learned that the most optimal ability to use on Alaois from now on will be Fire Hands.

There are exceptions to this flow, those being, not having range for some or all of the abilities, and having multiple targets. By not having range on one or more skills, the Ability Picker will ignore those abilities for the value comparison, as moving to be in range is only a last option if no other ability is in range. This can be seen in the Pick method, where if by the end finalPlan has

no ability, this plan becomes the optionalPlan, moving. This makes sense because from a purely numerical value, attacking gives more value than moving even if by moving the unit could access more powerful abilities.

On the other hand, if there are multiple targets within range, the Conditional Ability Picker will test each ability on each of the different reachable targets. In the previous example, if another player-controller unit was next to Alaois, the AI unit would have to test all three abilities again with the other target, and once tested, the AI would not only compare the abilities with themselves, but also with the other targets. We already know that Fire Hands is the most damaging ability on the mage's repertoire, but if for some reason the other target had returned a higher value for Fire Ball than using Fire Hands on Alaois, the mage would begin using Fire Ball on the other target until the ranges and/or targets changed.

The aforementioned scenario is shown in-game in one of the videos in the appendices.

Lastly, we will have another test scenario, this time with 10 AI units, and 5 player-controlled units.

The first unit to act is an AI-controlled one, therefore, EvaluateState is called. There, the result of the evaluation will be "Emboldened", as there are 10 Enemy units, and only 5 Hero units, fulfilling the condition of having double or more Enemy units than Hero units. The units initial state is always Default even if it is going to change immediately in the first turn. As such, since lastState is not the same as the actual state, the BattleMessageController will show a message on-screen that shows that the actual unit feels emboldened.

Then, once the state has been decided, the pattern is assigned using a FindChild method with the structure "state + job of the unit + Attack Pattern". After this, we enter on the Decision-making layer described in the previous scenario.

This last scenario is the one depicted if the player chooses a frontal attack instead of trying to ambush.As a hypothetical of this scenario, if the player were to defeat an enemy unit without suffering any loss themselves, the next AI units would have its state changed to "Default" due to the new check returning a result of 9 Enemy units against 5 Hero units, therefore not fulfilling any of the two conditions that mark either "Shaken" or "Emboldened".

Chapter 6

Conclusions

6.1 Tester Opinion

To better understand the project, I asked a group of 5 people with experience on TRPGs but no previous knowledge of the thesis if they saw any noticeable difference between the AI of the project and the usual AI of the TRPGs they had played. The set of questions was the following:

- What is your degree of skill on TRPGs?
- How many different TRPGs have you played?
- Did any of them have an AI outside of the norm?
- After playing, did you notice any difference with a normal AI from other TRPGs?
- Did you find the game difficult?
- Did you feel any more invested in the story?

Tester 1 was a seasoned veteran of the genre and as such cruised through the demo with no problem at all. They commented that the game was pretty easy and while the messages made the Emotion System apparent, he did not find any change at all over the course of the game. As such, they did not feel invested at all on the story, as the messages felt tacked on in an effort to make the NPCs more alive than they really were.

Testers 2 and 3 were more of a middle-of-the-road fans of the genre, liking it a lot but not having time to invest on many games or prioritising other kind of games. Both of them had similar experiences with the game, comparing the system and the GUI with Final Fantasy Tactics Advance, while making no comment on the AI. Through the questions it could be learned that they did not pay much attention to it but were pleasantly surprised when the message showed that the enemies had changed emotion, but did not feel any more invested in the story at all. The difficulty seemed normal but they felt it was too easy to miss with the skills.

Tester 4 had played the least of the group, with only beginning to enjoy the genre recently, and as such had more problems at the time of testing. They felt that the difficulty was increasing as the game progressed, and felt that by the end the enemies were spamming the same type of ability until their emotion changed. While not to the point of feeling cheated, they felt the AI had some kind of unfair advantage, but it did not last long because it happened near the end of the game. The narrative aspect fell flat on them because they were focused solely on ending the game as fast as possible.

Tester 5 was kind of an experiment, as they had no previous experience with TRPGs, but were interested in narrative-focused games. The aspect of having to choose strategies before the battle

even began was seen as a good idea, and while the Emotion System was perceived as innovative, the lack of conversation during the game made the NPCs feel as flat as faceless grunts.

6.2 Personal Opinion

Putting aside what has been learned though the process of the thesis, it has been made clear that not only the systems did not have as much narrative impact as initially thought, but also that a system that learns as slowly as this one does, does not make sense in the context of brief skirmishes or with units that can be defeated in as little as three turns, leaving no possibility for the system to be really put into effect.

The Emotion System, while simpler, had a better effect on the minds of the testers due to the fact that it affected the whole enemy side, but also fell flat due to the fact that the changes in patterns were not drastic enough, focusing mainly on the frequency of attacks rather than deeper strategies.

6.3 Learning as a developer

This project made me able to adapt myself to the work of others to implement my own ideas making use of the given systems and complementing them instead of working against the system, therefore boosting my teamwork capabilities even if I had to work on the project by myself.

Also, I found myself thinking about what other features I could add to the system to better implement the main ideas instead of simply dumping all necessary work into the implementation of the idea.

And finally, it helped me think critically on which features I could realistically implement against the deadline, which were more important on the overall intention of the project and which ones were simply secondary, helping me avoid feature creep.

6.4 Learning as a designer

This project made me able to think of features that helped improve the overall project, even if at first they seemed to not really add anything of interest.

Also, by following the tutorial I was able to see how other people think about what they feel is important to add as features, like its approach to AI made me understand that “perfect” AIs are boring and even unfair, and that giving them a certain “dumbness” would make more satisfying to defeat.

And finally, it also helped me think about how the features I added could help into making a more deep narrative, instead of thinking them as mechanics for the sake of mechanics.

6.5 Constraints

6.5.1 Technical Limitations

Due to how the placement of the units works in the original system, I was not able to place the units using variables, and therefore unit position will be hardcoded for each of the scenarios.

6.5.2 Scheduling Problems

At first, it seemed like what would occupy most of the time of the project would be the programming, yet at the end of it, I realized most of the time was dedicated to the research on how to implement the ideas before even touching any code. As stated in the "approaches" sections, seemingly simple tasks can be made much harder when trying to adapt them into another, bigger system, as the one in the tutorial.

In fact, those sections are the result of the research, and while the final implementation followed quite well the research, some changes were needed to be made. As a result of this, the end schedule went far different from the initial plan.

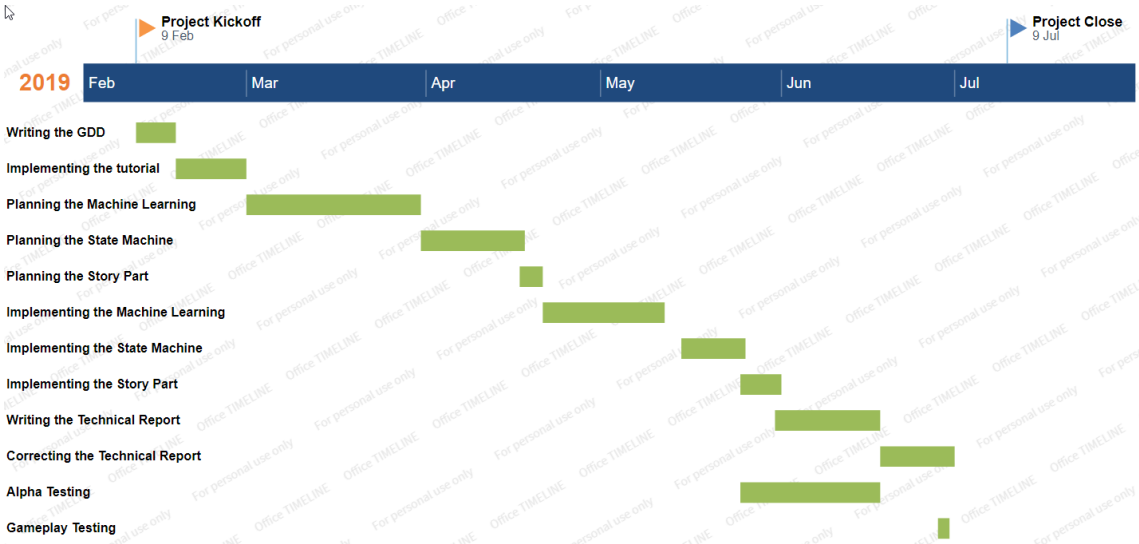

Figure 6.1: Gantt Chart of the final time allotted

While it may seem disproportionate to spend so much time on theory, it was necessary due to the need to understand all the ins and outs of the system provided by the tutorial, as starting to code with only a superficial understanding of the flow of the algorithms provided would have caused major problems. This also shortened the coding time considerably, therefore, while the initial schedule diverged greatly from the final schedule, the total hours did not vary as much.

6.6 Final Conclusions

While it has been a learning process that made me appreciate more the well-made and balanced AIs of commercial TRPGs, it has also made me realize that ideas that look good on paper may not seem all that good or even outright bad once implemented and tested, which is kind of frustrating but also puts an emphasis to the importance of testers, not only to know if the game works as intended, but also to note the inherent problems of the project itself.

On a more specific note about the AI, the idea of making a "dumb" AI that turned into a "smart" AI by learning, only punished those less experienced with the genre instead of enriching the experience for everyone, as intended, and did not have any effect on the narrative as a whole, at least from a player perspective.

In the end, it is sad that the project ended like this, but not every project can be a victory.

6.7 Future Lines of Work

I initially thought of this idea because it seemed like a simple way to enhance the narrative experience in an indirect kind of way, and as an aspiring narrative designer, it is always interesting to try new approaches instead of the traditional narrative, or try working in aspects that do not seem directly related to the narrative yet could help with immersion.

But the process of making this project made me realize that I may be more suited to more traditional narratives and may need to practice on that route before trying again on more obscure ways to improve the narrative.

Bibliography

[1] X-COM. https://en.wikipedia.org/wiki/X-COM, Feb 2018.

[2] The Banner Saga. https://en.wikipedia.org/wiki/The_Banner_Saga, Feb 2018.

[3] Fire Emblem. https://en.wikipedia.org/wiki/Fire_Emblem, Feb 2018.

[4] Unity 3D. http://www.unity.com, Feb 2018.

[5] Fungus. http://fungusgames.com, Feb 2018.

[6] Europa Universalis. https://en.wikipedia.org/wiki/Europa_Universalis, Feb 2018.

[7] Age of Empires. https://en.wikipedia.org/wiki/Age_of_Empires, Feb 2018.

[8] John Funge Ian Millington. Artificial Intelligence for Games (2nd edition). Elsevier, 2009.

[9] Jon Parham. Liquid Fire. http://theliquidfire.com/2015/05/04/tactics-rpg-series-intro/, May 2015.

[10] Final Fantasy Tactics Advance. https://en.wikipedia.org/wiki/Final_Fantasy_Tactics_Advance, Feb 2018.

Chapter 7

Appendixes

7.1 Links

- Link to the repository of the project
- Video demonstrating the Error System
- Video of general gameplay of the demo

Addendum: For the project to work on Unity, one has to download Fungus and also select Parse Jobs on the pre-production contextual menu. The project uses a json file to get the stats of every unit, so the file has to be parsed with that option. This only needs to be done once.

7.2 Algorithms

7.2.1 Machine Learning Algorithm

```
public override void Pick(PlanOfAttack finalPlan)
    {
        PlanOfAttack optionalPlan = new PlanOfAttack();
        for(int i = 0; i < pickers.Count; i++)
        {
            PlanOfAttack plan = new PlanOfAttack();
            BaseAbilityPicker p = pickers[i];
            p.Pick(plan);
            cp.Evaluate(plan);
            if (plan.ability == null)
            {
                optionalPlan = plan;
                continue;
            }
            string unit = plan.unit.name;
            string ability = plan.ability.name;
            Unit.Pair pair = new Unit.Pair(unit, ability);
            if (owner.pairings.Contains(pair))
            {
                pair = (Unit.Pair) owner.pairings[owner.pairings.IndexOf(pair)];
            }
            else
            {
                owner.pairings.Add(pair);
                max = pair.damage;
            }
            if(pair.damage >= max)
            {
                max = pair.damage;
                finalPlan.ability = plan.ability;
                finalPlan.target = plan.target;
                finalPlan.moveLocation = plan.moveLocation;
                finalPlan.fireLocation = plan.fireLocation;
                finalPlan.attackDirection = plan.attackDirection;
                finalPlan.unit = plan.unit;
            }
            bmc.Display("Current max: " + max + " by " + ability);
        }
        max = 0;
        if(finalPlan.ability == null)
        {
            finalPlan.ability = optionalPlan.ability;
            finalPlan.target = optionalPlan.target;
            finalPlan.moveLocation = optionalPlan.moveLocation;
            finalPlan.fireLocation = optionalPlan.fireLocation;
            finalPlan.attackDirection = optionalPlan.attackDirection;
            finalPlan.unit = optionalPlan.unit;
        }
        finalPlan.complete = true;
    }
```

7.2.2 State Machine Algorithm

```
string EvaluateState()
   {
       string state = "";
       int heroes = 0;
       int foes = 0;
       for(int i = 0; i < bc.units.Count; i++)
       {
           if (bc.units[i].GetComponent<Alliance>().type == Alliances.Hero) heroes++;
           else foes++;
       }
       if (foes >= heroes * 2) state = "Emboldened";
       else if (heroes >= foes * 2) state = "Shaken";
       else state = "Default";
       return state;
   }
```