

Gamification Quest: *
Design and Development of a gamification game



Luis Alisandra Senabre

Advisor: Dr. Raúl Montoliu Colás

Department of Engineering

Universitat Jaume I de Castellón

This dissertation is submitted for the bachelor's degree in Video Game Design and Development

Abstract

Nowadays video games are taking a more active role in our society. A field in which they have not yet highlighted but have great potential is in the education system.

In this document, Gamification Quest: * is presented. It forms part of a bigger educational project named Gamification Quest which pretends to join the education system with video games, in order to motivate the students and make them improve their academic performance. Gamification Quest: * (GQ*) assumes the playable part of the project as an RPG video game.

The intended target goes from high school to college students, in other words, people from 14 to 24 years which are in high school or college. Gamification Quest is a more complex education application but GQ* focuses on presenting a mechanics and a game system that uses the student's notes and deliveries, collected by the GQ application. Motivating him to improve his academic performance in order to progress in the game.

The development will target the Android platform which is the most extended one among smartphones and Unity3D is used as the game engine for the implementation of the game.

The core of the GQ* project is the Artificial Intelligence Techniques (AIT) that controls the Non-Playable Characters (NPC). They are implemented using the minimal Unity tools and focusing on their implementation from scratch. The techniques implemented are Steering Behaviors for characters' movement and Behaviors trees for controlling their actions and allowing them to respond in real-time to the player's actions.

Key Words:

Gamification, Artificial Intelligence, Game mechanics,
Squad's coordination, RPG, Behavior Trees

List of Contents

List of Contents	5
List of Tables	8
List of Figures	9
Chapter 1 Technical Proposal.....	10
1.1 Motivation.....	10
1.2 State of the Art.....	10
Gamification.....	10
RPG Games.....	11
AI Techniques.....	11
1.3 Game Overview.....	11
1.4 Objectives	12
1.5 Justification	13
1.6 Tools.....	13
Programming	13
3D Art.....	13
2D Art.....	13
Documentation	14
1.7 Project Plan	14
Section 1 –Design (D)	14
Section 2 –Implementation (I).....	14
Section 3 –Art (A)	15
Section 4 – Documentation (Doc)	16
1.8 Risk Management	17
Chapter 2 Gamification Context.....	19
Chapter 3 Game Design Document.....	20
3.1 Characters	20
Avatar.....	20
Familiars.....	20
Enemies.....	21
3.2 Game mechanics	21
Avatar.....	21
Familiars.....	22

Formations	22
Dungeons	23
Enemies.....	23
Levelling Up.....	23
Unlock new Dungeons	24
Getting new familiars	24
3.3 Player Controls	24
3.4 Gameplay	24
3.5 Game Visual Design.....	25
3.6 Game Music & Sound Design	25
Chapter 4 Project Development.....	26
4.1 Game User Interface (I21)	26
4.2 General Information Classes	27
4.3 Game Characters(I1)	28
GameObjects & Components.....	28
Scripts	28
4.4 Testing & Debugging	31
Chapter 5 AI Techniques	32
5.1 Steering Behaviors	32
5.2 Behavior Trees	34
Enemy behavior Tree	38
Boss behavior Tree.....	39
Familiar Behavior Trees.....	40
5.3 Actions & Action Manager	42
5.4 Blackboard	46
5.5 Putting all together	46
5.6 Testing & Debugging	47
Chapter 6 Game Art	48
6.1 Visual Identity	48
6.2 2D Elements	49
6.3 Characters	49
5.4 Environment (I2):	51
Chapter 7 Results	53
6.1 Technical Proposal	53
6.2 Game Document Design.....	53

6.3 Technical Report	53
6.4 Gamification Quest: *	53
6.5 Project Exhibition Video	55
6.6 Project Defense Presentation.....	55
Chapter 8 Conclusions.....	56
Project Objectives	56
Project Deviation.....	57
Project Risk	58
Future Work.....	59
Chapter 9 References.....	60
Appendix A Sound Design Document by Victor Avila	61

List of Tables

Table 1 Design Planning	14
Table 2 Implementation Planning	15
Table 3 Art Planning	16
Table 4 Documentation Planning	16
Table 5 Risks.....	18
Table 6 Dual Formations	23
Table 7 Triple Formations	23
Table 8 Sounds & Music List.....	25
Table 9 BT Checkers	35
Table 10 Actions.....	36
Table 11 BT Decorators	37
Table 12 BT Nodes Miniatures	38
Table 13 2D Game's Elements.....	49
Table 14 Environment Elements	52
Table 15 Project's Item Count	55
Table 16 Project's Schedule Deviation	57
Table 17 New Schedule tasks	58

List of Figures

Figure 1 Grant diagram	17
Figure 2 Mechanic Layers.....	21
Figure 3 GUI	26
Figure 4 Familiar's GUI	27
Figure 5 Characters' Interfaces	29
Figure 6 Familiars' Position in all the formation.....	31
Figure 7 SteeringOutput struct	33
Figure 8 Kinematic struct	33
Figure 9 SteeringObject interface	33
Figure 10 BT_Task Node.....	34
Figure 11 Enemy Behaviour Tree	38
Figure 12 Boss Behavior Tree.....	39
Figure 13 Familiar Movement s.....	40
Figure 14 Fighter Attack Subtree	41
Figure 15 Defender attack subtree	41
Figure 16 Magician attack subtree	42
Figure 17 Action Class	43
Figure 18 Enemy Seek Action	43
Figure 19 Action Manager Pseudo-code	45
Figure 20 Blackboard Interface	46
Figure 21 The complete AI Structure	47
Figure 22 GQ Main Menu Screenshot.....	48
Figure 23 Avatar's Designs	50
Figure 24 Avatar's modelling.....	50
Figure 25 Avatar's final model.....	51
Figure 26 Avatar's skinning	51
Figure 27 Dungeon Level.....	52
Figure 28 Download code	54
Figure 29 Formation Screenshots	54
Figure 30 Combat Screenshots	54
Figure 31 Boss Combat Screenshots	54

Chapter 1

Technical Proposal

1.1 Motivation

From my point of view, students lack in motivation during the school year, in general terms. They perceive their education more as an obligation than something that benefits them and the homework and exams turn into a real pain for them. I think that's something that can and should be changed.

From my previous experience working with children, I learned that one of the better ways to teach a kid something is by making they play and having a good time but, at the same time, making them aware that it is something serious and that they are learning. That, joining my background as video game developer and my interest in artificial intelligence techniques comes up in this project.

1.2 State of the Art

This section discusses the main aspects of the project, deepening on the current state of the main issues the project approaches.

Gamification

“Gamification is the application of game-design elements and game principles in non-game contexts. Gamification commonly employs game design elements which are used in non-game contexts to improve user engagement, organizational productivity, flow, learning, crowdsourcing, ...” [1]

In an educational context, most of gamification systems aim to improve the student's performance by proposing challenges and rewards, competitiveness and interaction between students. An example is the application *ClassDojo* [2] that helps the teacher improve the learning experience in the classroom in a simple and easy way, using achievements and rewards for the students in real time during the lessons.

RPG Games

A role-playing game (RPG) is a game where the player takes the role of a fictional character and take responsibility for his action in the world where the game develops. Normally there is a narrative in the game that guides the characters and leads them to different situations that they have to solve in order to continue the story.

The variant of RPG that concerns the project is the RPG video game type which is a video game genre where the player controls the actions of a character (and/or several party members) immersed in some well-defined world. This character normally adopts some role in the party group such as fighter, magician, healer ...

This project's mechanics are based on Final Fantasy XIII [3] paradigms system [4] where the player chooses a role for every character in several formations and changes them during the game.

AI Techniques

The project uses two main AIT with some variations to be implemented on Unity 3D:

Steering Behaviors: This technique aims to help autonomous characters move in a realistic manner, by using simple forces that are combined to produce lifelike, improvisational navigation around the characters' environment. They are not based on complex strategies involving path planning or global calculations, but instead use local information, such as neighbor's forces. This makes them simple to understand and implement, but still able to produce very complex movement patterns. This technique is the alternative to the NavMesh and NavAgent Unity Components.

Behavior trees: A behavior tree (BT) is a mathematical model of plan execution used in computer science, robotics, control systems and video games. They describe switchings between a finite set of tasks in a modular fashion. Their strength comes from their ability to create very complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. BTS present some similarities to hierarchical state machines with the key difference that the main building block of a behavior is a task rather than a state. Its ease of human understanding makes BTs less error prone and very popular in the game developer community. BTs have shown to generalize several other control architectures.

1.3 Game Overview

GQ* makes up the interactive part of the GQ application: besides the academic performance, the student need to replay several times the different levels of the game in order to power up his account. In the levels the player has to complete a dungeon, traveling across its halls to find different objects and facing the monsters who live in it to gain experience points (EP). He has the help of his familiar spirits (or simply familiars), NPCs with its own characteristics, controlled by a set of AI techniques that make them adapt their behaviors in real-time to the player's commands. The familiars have five basic stats that determine their role and power: VP (Vitality points), Attack (Physical damage), Magic Attack (Magical damage), Defense (Resistance to physical damage) and Magical Defense (Resistance to magical damage).

Before entering the dungeon, the player has to choose three of his familiars and put them in three different formations (a formation is a set of three familiars with one role each). Changing the role of the familiars, the player can create different formations focusing on the attack or on the defense according to the situation. Once inside the dungeon the player can swap between the three preconfigured formations and the familiars will react adapting their behavior to their new role in each formation. To fulfill the different formations steering behaviors AI techniques will be used, implemented separately and combining them to create more complex squadron behaviors.

In total, each familiar can play three different roles in a formation. Each of them powers the stats of the familiar according to his function:

- **Fighter:** Is placed on the front line and takes care of inflicting physical damage to the enemy party. Power Stats: Attack and VP.
- **Wizard:** Is placed in the rear of the formation and cast powerful spells to inflict magical damage to the enemies. Power Stats: Magic Attack.
- **Defender:** Is the spearhead of the formation and his duty is to receive as most damage as possible from the enemies in order to protect his weaker partners. Power Stats: VP, Defense, and Magical Defense.

The enemies, just like the familiars, have the five basic stats named before but, unlike the familiars, they cannot change roles, each type of enemy will have a predetermined role and will always be that.

At the time of combat, the player can select an enemy as the target and attack him with either physical or magical damage. His familiars will focus their attacks on him according to the criteria of its actual role.

Due to the gamification context in which the game is found it is necessary that the levels are replayable and that they are not boring. Because that the enemies are spawn at different points each time and has a strength proportional to the player's level.

With the game concept designed around gamification, the player sees his efforts rewarded within the game, with an increase in his stats if he has worked correctly in the course. Thus, the reward is linked directly with the actions carried out, encouraging the activity of students within the gamificated subject by the GQ app.

1.4 Objectives

This project has two types of objectives to accomplish. The first one are the Subject's own objectives, listed on the Docent Guide of the subject [5]. The other ones are the objectives set up explicitly to this project and are listed next:

- O1 - Implement a video game which takes advantage of the Gamification Quest app.
- O2 - Minimize the use of Unity tools to implement the AIT used on the game, creating them from scratch whenever possible
- O3 - Design and implement fun game mechanics that integrate the use of complex AIT.
- O4 - Implement the familiars so that they respond in real-time to the player's orders.

1.5 Justification

In this section, the relation between the proposed project and the courses taught in the pertinent bachelor's degree will be stated. Since Gamification Quest: * is a complete video game almost every subject attended during these years could be mentioned. However, only the most related ones are listed.

As mentioned in section 1.4, the main objectives of the project are creating complex AIT from scratch and taking advantage of the GQ app, so the most related subjects from the bachelors are VJ1231 Artificial Intelligence and VJ1222 Video Game Conceptual Design. The first one, for the knowledge about Artificial Intelligence Techniques such as Steering Behaviors and Behaviors Trees in order to accomplish O2, O3, and O4. The Second One to fulfill O1 and be able to integrate the game mechanics with the GQ app in an organic way.

As the basis for VJ1222 is necessary the contents thought in VJ1215 Algorithm and Data Struts in order to implement the AIT techniques mentioned and accomplish the real time requirement in O3

For the visual aspect and for the UI, the subjects VJ1223 Videogame Art and VJ1216 3D Design are strictly necessary, for both 2D graphics and 3D models.

1.6 Tools

This section lists the tools that will be used during the design and development of this game. These tools can be grouped according to the topic they are related.

Programming

Unity 3D: The main tool for the game development along with Visual Studio. Unity 3D is a free game engine that supports every requirement the project could depend on. It supports both 2D and 3D elements, an essential feature for the creation of interactive UI.

Visual Studio: An integrated development environment from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as websites, web apps, web services and mobile apps. Along with Unity3D is the main core of the programming development of the project.

3D Art

Autodesk 3DS Max: A professional 3D computer graphics program for making 3D animations, models, games, and images. Furthermore, it incorporates rigging and skinning tools, as well as a full animation support. Models are then easily exported for their use in the selected game engine.

2D Art

Adobe Photoshop: A raster graphics editor used to create all the sprites, buttons and icons from the game.

Documentation

Overleaf: An online Latex editor used for the first deliverable D1-Technical Propose.

Microsoft Word: A word processor developed by Microsoft. Due to the lack of knowledge about Overleaf and Latex, Microsoft Word was chosen to write up this final report.

Microsoft PowerPoint: A presentation program that provides templates, transitions, popping elements and annotations, very useful for the final presentation.

1.7 Project Plan

This section explains the intended project schedule for the Gamification Quest: * game. The whole development of the project can be grouped in for sections. Each section has his own task specifying the task code, the task name, the initial date, and the expected duration.

Section 1 –Design (D)

General aspects of the game design. All these aspects will be explained ahead in section 3.

Table 1 Design Planning

D - Game Design			
Code	Task	Duration	Hours
D1	Mechanics Design	01/02 --- 05/02	7
D2	Characters Design	06/02 --- 27/02	17
D21	3 Familiars Design	06/02 --- 19/02	10
D22	4 Enemy Design	16/02 --- 27/02	7
D3	Level Design	25/02 --- 04/03	6
Total Hours			30

Section 2 –Implementation (I)

This section includes all the implementation of the project: mechanics, scene and the artificial intelligence technics required for the game. It also takes on the test and debugs aspect which is developed throughout the implementation process.

Table 2 Implementation Planning

I – Game Implementation

Code	Task	Duration (from --- to)	Hours
I1	Mechanics Implementation	04/02 --- 01/04	20
I11	Character Mechanics	04/02 --- 18/02	5
I12	Enemies Mechanics	18/02 --- 04/03	5
I13	Familiar Mechanics	04/03 --- 01/04	10
I2	Scene Implementation	01/04 --- 30/06	20
I21	Environment	19/06 --- 23/06	3
I22	Camera Movement	01/04 --- 04/04	2
I23	Object System	23/06 --- 30/06	5
I24	Enemies Respawn	16/06 --- 20/06	5
I25	User Interface	04/04 --- 12/04	5
I3	Artificial Intelligence Techniques	12/04 --- 15/05	80
I31	Familiars AIT Design	12/04 --- 19/04	7
I32	Enemies AIT Design	17/04 --- 19/04	3
I33	Familiars AIT Implementation	20/04 --- 08/05	45
I34	Enemies AIT Implementation	08/05 --- 15/05	25
I4	Testing & Debugging	04/02 --- 08/07	45
Total Hours			165

Section 3 –Art (A)

This section considers all the phases of the game art, from concept art to animation and the subsequence integration in the game.

Table 3 Art Planning

A – Game Art			
Code	Task	Duration (from --- to)	Hours
A1	Blueprints	15/05 --- 18/05	14
A11	Familiar's Blueprints	15/05 --- 16/05	6
A12	Enemies' Blueprints	17/05 --- 18/05	8
A2	Modelling	19/05 --- 01/06	35
A21	Familiars' Modelling	19/05 --- 25/05	15
A22	Enemies' Modelling	26/05 --- 01/06	20
A3	Animations	30/06 --- 09/07	21
A31	Familiars' Animations	30/06 --- 05/07	9
A31	Enemies' Animations	5/07 --- 09/07	12
A4	Game Integration	01/06 --- 09/06	10
Total Hours			70

Section 4 – Documentation (Doc)

In this section all the deliverable documents required for the project are reflected.

Table 4 Documentation Planning

Doc – Project Documentation			
Code	Task	Duration	Hours
Doc1	Technical Proposal	02/02 --- 12/02	10
Doc2	Game Document Design	20/02 --- 01/04	10
Doc3	Final Technical Report	15/05 --- 03/07	15
Total Hours			35

Figure 1 shows up the schedule in a more graphical way, created using Office Timeline, a Power Point extension.

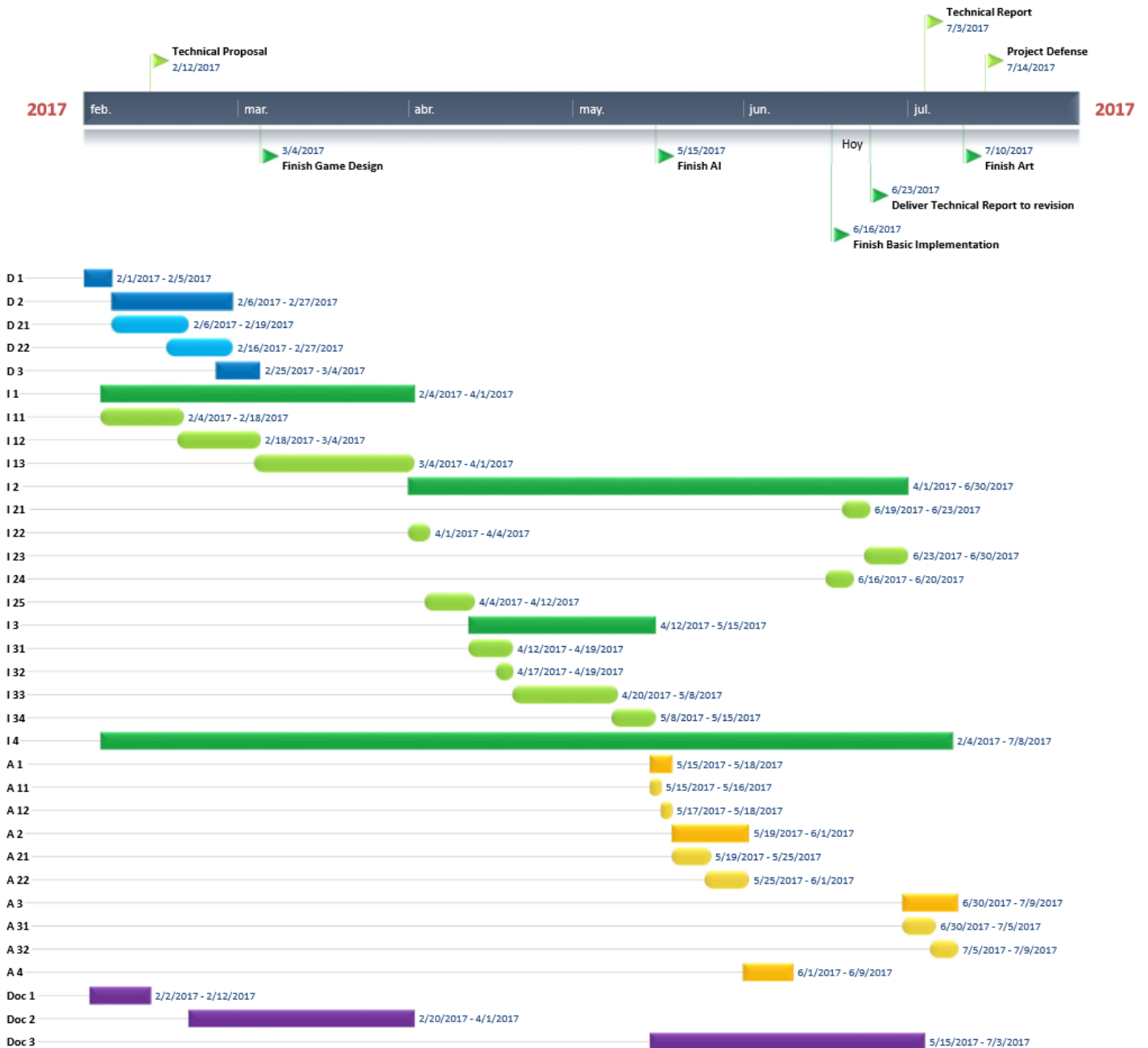


Figure 1 Grant diagram

1.8 Risk Management

In total, the number of hours planned for the project are 300. However, this is a schedule prior to any work on the project so it is very probable that some issues come up and breaks the schedule. To anticipate such problems, this section includes some of the most expected problems and a containment plan to minimize their impact in the project schedule and final result. Table 5 contains all this risks with an ID, Name, the probability (P) of it appears (High - H, Medium- M, Low - L), its impact (I) on the project schedule (same measure as the probability), a description of its effect and the containment plan to perform

Table 5 Risks

ID	Name	P	I	Effect	Containment Plan
R1	Miscalculate implementation task durations	H	M	Some task wouldn't be completed for the end of the project and the game can't be played	Continue to work on the most crucial tasks for the project presentation
R2	Lack of time to devote to art and modelling	H	L	The initial mock-ups of the development phase will continue to be, damaging the visual aspect of the game	<p>UI: Reuse as much elements from the GQ app as possible, provided they meet minimums</p> <p>3D : Model the minimum number of characters and use palette swap to differentiate them or download and use some free models from the Unity Asset Store</p>
R3	Lack of skill or knowledge in some aspects of the game development	H	H	Some task can't be completed or take too much time doing trial and error	Invest part of the time of the task in investigating
R4	Data loss	L	H	Some files or the whole project get lost	Regularly create back-ups
R5	Scheduled tasks turn out to be incorrect or poorly focused	M	L	The initial schedule turn out to be useless	Work on the new task and reflect them in the Results Section

Chapter 2

Gamification Context

This Chapter explains the basis of the Gamification Quest application, developed by Marina Granell and myself while working in the Instituto de Nuevas Tecnologías de la Imagen (INIT) during our internship. Nevertheless, the Gamification Quest: * project remain independent of the work done for the GQ app. Both the design and the implementation of GQ: * are completely developed for the bachelor's degree final project.

GQ* places inside a whole gamification system called Gamification Quest (GQ) which integrate a university degree inside a game app. GQ collects the students courses information from the university Moodle page and transforms it into game elements, motivating the student to increase his academic performance in order to progress inside the game.

On the basis of the Gamification Quest services, the main idea is to create a feedback between the university performance of the student and his gameplay in the application. On the one hand, if the student attends to class and does his homework will become in a power-up of his avatar in the game. On the other hand, make progress in the game will make the student achieve new educational challenges like secret tasks or exams to get more grades or unlock new features in the game.

The app reads the student's courses information and take the delivers and quizzes from the academic subjects and transforms them into Tasks. When the student completes the tasks on Moodle (delivering the delivers or answering the quizzes) he gets a world key from the GQ app. That key allows the user to unlock a new playable level.

The student is represented in the game by an avatar. The avatar level represents the player progress in the game. In order to level up, the player must collect experience, collecting the tasks rewards once they are completed.

The app creates also a ranking between the students who are using it. Each student has a personal ranking facing the students who have the same degree subjects and taking into account them avatar level and the number of completed task they have. By doing that the app creates a competitiveness between the students and motivates them to strive during the course.

Chapter 3

Game Design Document

Gamification Quest: * is a 3D game which complements the Gamification Quest app by carrying out the playable part of the gamification system. The app collects the academic performance of the student to unlock new levels and allowing the player to improve. This chapter is a game document design of the game implemented and corresponds to the items D1, D2, and D3 of the section 1.7 from the chapter 1.

In the game, the player is a mage who controls its familiars and walks through the dungeons defeating the monsters and leveling. The game doesn't have a plot line which dictate the game advance, is the student performance in the course which makes the game advance unlocking new dungeons, each one harder than previous. Each dungeon will have a start point and an exit, the player, together with its familiar, has to cross it, defeating the enemies and collecting experience to level up both, the player and its familiars.

3.1 Characters

There are three archetypes of characters in the game divided in playable and no playable characters: the player, the familiars and the enemies. In this section all three are explained.

Avatar

The player is represented in the game through the avatar, an androgynous character who wears a long poncho which covers its body and its head with a hood and grips a magic wand to fight the dungeon monsters. Its appearance is as generic as possible in order to make easier the player embodiment. The player can do both physical and magical attacks to deal damage against the enemies. Nevertheless, its strength isn't enough to fulfill the dangerous dungeon.

Familiars

They are the magic creatures that accompany the player. Based in West Europe mythology, a familiar is a magic entity that make a pact with the witch or mage who summon it, becoming thus their servant and obeying all their orders.

Enemies

The enemies are the different types of monsters which live in the dungeon. They behave is similar to the familiars but they will attack the player's party. Two categories can be differentiated:

Regular monsters: They live in the dungeon and wander their passages. There are very aggressive and will attack the player's party when they see it.

Dungeon Boss: Is the main enemy who guard the dungeon exit. In order to get out, the player must defeat the boss and his minions (normally some regular monsters from the dungeon). The dungeon boss is one of the different familiars the game has but more powerful than his familiar version and the regular monsters of his dungeon.

3.2 Game mechanics

In this section the game mechanics are exposed. In pursuit of a more organic understanding, the different mechanics are explained in a short of chronological order, similar to a game flow explanation.

But, at this point, it is necessary to distinguish between three layers of mechanics in the Gamification Quest Project. The first layer is the one explained in section 2, the mechanics between the student academic performance and the application.

The second one is the mechanics of the GQ* game integrated on the application. These mechanics make sense based on the repetition of the GQ* levels and are in a deadlock between GQ app and GQ* levels. These mechanics are explained in this section for reasons of cohesion with the rest of the project, however its implementation is not contemplated as part of the work.

Finally, the third layer is the mechanics of the proper playable level. These mechanics are also explained ahead in this section and later, in section 4, their implementation is shown.

The representation of how each mechanic layer interacts with the whole gamification experience is show in figure 2.

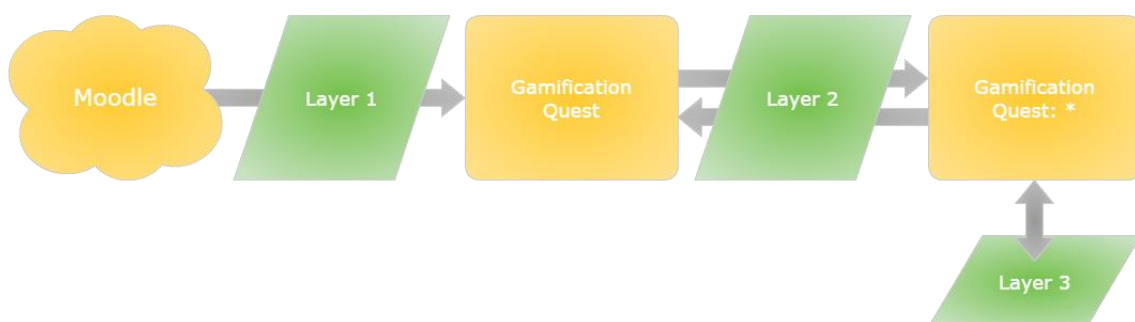


Figure 2 Mechanic Layers

So on, first the layer 3 mechanics are explained, in order to better comprehend the layer 2 mechanics.

Avatar

The avatar is the only character controlled by the player so their mechanics are the basis of most games, move and attack. For the attack the player can choose between physical attack or magical attack. Regardless of which one they use, there are a little cool down to both in order to prevent the attack spawn. To perform an attack, the avatar must have an enemy target.

Familiars

The familiars are NPCs controlled by AIT to respond in real time to the player's commands. Each familiar has five basic stats which express the familiar strength in the game. These stats are: Attack (Physical damage), Defense (Physical damage resistance), Special Attack (Magical damage), Special Defense (Magical damage resistance) and HP (Health Points). Each type of familiar has a specific set of stats which increase by leveling up.

Based on the stats set of each familiar, it is possible to perceive a function of that familiar inside the player's party, for example, as the attacker, the defender or the magical attacker. Each type of familiar has its own stats set allowing to categorize them based on their total amount of stats. That means there are familiars stronger than others without considering their level.

Besides these stats sets, each familiar can play up to three different roles in the group, each one focused in one job inside the group. The game AI takes care of controlling all the characters except the avatar, but, with the role system, the player can guide their familiar behavior depending on their interests. The roles are:

- Fighter (F): Is placed on the front line and takes care of inflict physical damage to the enemy party. Powered Stats: Attack and VP.
- Wizard (W): Is placed in the rear of the formation and cast powerful spells to inflict magical damage to the enemies. Powered Stats: Magic Attack.
- Defender (D): Is the spearhead of the formation and his duty is to receive as most damage as possible from the enemies in order to protect his weaker partners. Powered Stats: HP, Defense and Magical Defense.

It is important to note that the stats set and the role aren't related. The stats set pertains to the familiar, while the player decides its role and, consequently, which stats are powered. In this sense, the player can choose an attacker type familiar to play the defender role, and its stats will be powered as such.

The way to choose the familiar role and how to change between them are explained ahead.

Formations

A formation is the combination of up to three familiars with one role each. As a formation can have two or the three familiars playing the same role, there are sixteen different formations, plus the three one member formations and the player going without familiars. Each formation has a name to easily and quickly distinguish it from another. The names are show in tables 6 and 7 for the two and three formations respectively. Note that the names are based on the FFXIII Paradigm system.

Table 6 Dual Formations

Formation Name	Roles
Double Trouble	F + F
Slash & Burn	F + W
Lifeguard	F + D
Dualcasting	W + W
Arcane Defense	W + D
Twin Shields	D + D

Table 7 Triple Formations

Formation Name	Roles
Cerberus	F + F + F
Aggression	F + F + W
Legion of Steel	F + F + D
Relentless Assault	F + W + W
Delta Attack	F + W + D
Impregnable	F + D + D
Aquelarre	W + W + W
Mystic Tower	W + W + D
Patient Probing	W + D + D
Great Wall	D + D + D

Before entering the dungeon, the player can choose up to three of their familiars to accompany and fight with them. Once the familiars are chosen, the player have to configure their roles to create three different formations. The idea is to have an assorted set of formation to be able to adapt different situations once inside the dungeon, because once the player enters, they can't change the formations.

Dungeons

When the player finishes building the formations, is time to actually play the dungeon. During level, the player can freely swap between the chosen formations. The familiars will adapt both their position and their behavior according to their new role in real time. Is in this dynamic change where the main combat mechanic remains.

To clear the dungeon, the player has to defeat the dungeon boss who guards the exit. Going straight to the boss or defeating all the enemies before exit the dungeon is up to the player.

Enemies

The regular enemies are also controlled for the game AI and have a role as the familiars but, unlike these, they cannot change their role during battle. This is to bring the player an idea of their stats and hints of the strategy to defeat them.

As mentioned before, the dungeon boss is one of the familiars powered-up. Once the boss is defeated the level is over and the player receive an according amount of experience points (exp) and some familiar essences, which will be explained ahead.

So far this are the layer 3 mechanics, that means, the mechanics inside the level. In order to motivate the player to replay the dungeons the layer 2 mechanics are essentials.

Levelling Up

When a dungeon is complete, the player receives exp according to the enemies defeated. The familiars with whom the player cleared the dungeon take that exp and, when

they get the necessary amount, raise their level. A familiar start with level 1 and can raise up to level 100. Every time it levels up, its stats increase and it becomes stronger.

Unlock new Dungeons

As referred in section 2, when the students complete a task, then obtains a word key, which allows them to unlock a new dungeon. The different dungeons are shorted by levels, where dungeon B1 is the easier and dungeon B15 is the hardest. The enemies inside the dungeon are more powerful and more numerous as the level of their dungeon increase, dropping more exp as hard as they are.

Getting new familiars

The player starts with one familiar only. In order to unlock new familiars their must collect the enough familiar essences of that familiar to summon it. As mentioned before, the familiar essences are collected from the dungeon boss, depending on the boss type will drop that familiar essences. If the player collects the essences of one of their familiars, they turn out in more exp for that familiar at the end of the level.

To allow the player to obtain all the familiars, the dungeon boss changes every 8 hours, but not randomly. In the deepest levels, the hardest ones, the strongest and rarest familiars appear and in the superficial ones, the weakest familiars. This feature encourages the student to fulfil as much task as possible in order to obtain the best familiars.

3.3 Player Controls

The game is meant for be played on a mobile device, therefore the control system is entirely tactile. The mobile orientation is landscape. As mentioned before, the game AI takes care of all the character's actions except the avatar. So the player only has four different actions:

To move the party group, the player has a virtual joystick placed on the screen's bottom left corner. Actually the player only moves the avatar, their familiars follow him by AIT.

To change the current formation of their familiar the player has two transparent buttons on the screen's right bottom corner. By pushing the up or the down button the player swaps the current formation for the next one or the previous one respectively. The formation change is cyclic, so if the player is in formation 1 and press three times the down button their will change from formation 1 to formation 0 then to formation 2 and again to formation 1.

To avatar's attacks there are two circular buttons above the change formation's buttons. The left one is for physical attack and the right one for the magic attack. Each time one of the buttons is pressed, both of them have an animation to represent the attack cool down.

Finally, to choose a target the player has to touch the enemy, if their touches it again it will be dis-targeted.

3.4 Gameplay

In a short of resume this section explains the general overflow of the game.

First the player chooses three of their familiars and configure three different formations with their roles. Then enters the dungeon and begins to go through its halls looking for the exit or for enemies to fight.

When they found an enemy quickly identify its role and change to the accurate formation to defeat it. Their goes on until find the exit and the dungeon boss. When defeats them the level is clear and the player receives exp and, if it comes to that, familiar essences.

3.5 Game Visual Design

As mentioned before, GQ: * is a part of the Gamification Quest app, that means they share a common visual design. The art can be distinguishing in 2D Sprites, for the UI and textures, and 3D models. Both types keep a similar style with a medieval aesthetic and elements with magic motifs but use different color palettes.

The sprites use a desaturated color palette and are elements without boundary lines, hard shadows and plane aspect.

On the other hand, the 3D models use a more colorfully palette with simply degradants textures and, occasionally, bump and transparency maps. The models are low poly (around 1500 polygons) because the game is for mobile devices.

3.6 Game Music & Sound Design

The music and sounds of the game are completely originals. The composition of the music is borned by Victor Avila, student from the Conservatori Superior de Música de Castelló. Information regarding the composing of these and other tracks included in the gamification application can be found on Appendix A.

The music is composed taking care of the medieval aesthetic, in order to keep cohesion in the game. Table 8 shows the list of sounds used for the game and when are they played.

Table 8 Sounds & Music List

Sound Code	Sound Name	When is played
S_01	Button click	When the player touch the GUI buttons
S_02	Magic Attack	When any character uses a magical attack
S_03	Physical Attack	When a familiar or an enemy does a physical attack
S_04	Avatar P Attack	When the avatar executes a physical attack
S_05	Magic Charge	Previous to any magic attack
S_06	Avatar treads	While the avatar is walking or running
S_07	Familiar damage	When a familiar gets hurt
S_08	Enemy Death	When an enemy dies
S_09	Avatar's shout	When the avatar attacks or gets attacked
M_01	Track 01	Always as background music

Chapter 4

Project Development

This section explains the concrete work done for the project, in terms of implementation. This chapter, along with 5 and 6, describes the work done for each item scheduled in Section 1.7.

4.1 Game User Interface (I21)

The game user interface (GUI) is in charge of receiving and transforming the player's input into action on the game, also shows to them the internal changes of the game in a visual way. It is composed for several elements, shown ahead in Figure 3, plus a Script named GUIController.

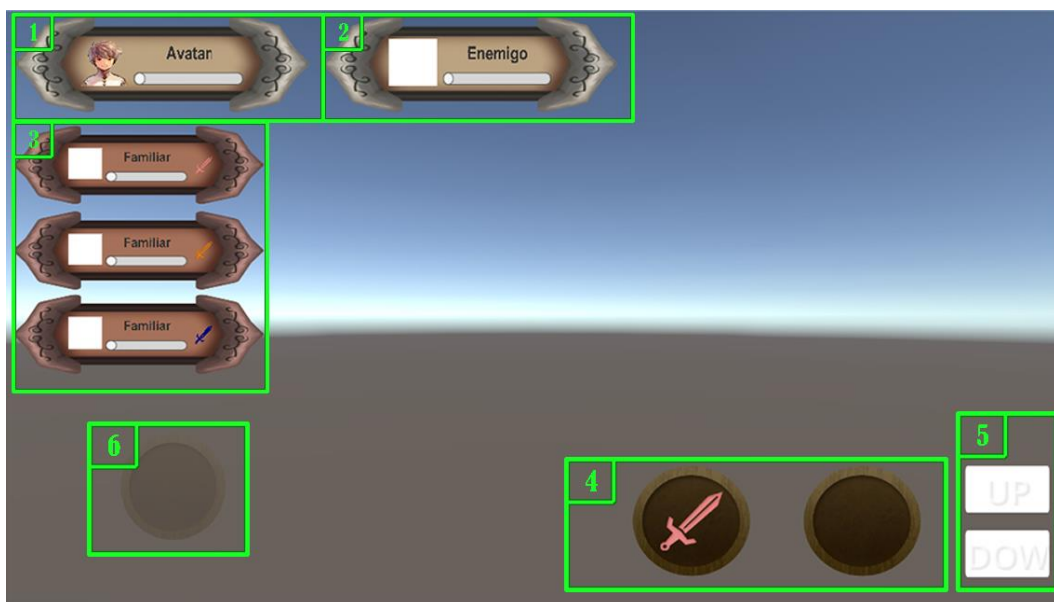


Figure 3 GUI

The Controller receives the player touches and execute a proper function for each button, calling the other controllers methods. It also has function to upload the player's, familiars' and enemy's health bar and the familiar's role color.

1 Player's GUI: This element is composed by a background sprite, a sprite with the avatar's image, a slider which represents the player's hp and a Text Component with the player's name. The slider is the same for the rest of elements, it lerps its color from green when it's at max hp to red at zero passing by yellow at half hp.

2 Enemy's GUI: Similar to the Player's GUI but holds the player's current target enemy information. If the character doesn't have a target, the GUI is deactivated.

3 Familiar's GUI: It is composed of three to zero individual components, depending on the number of familiar the player has at that level. Each component it's very similar to the player's and enemy's GUI but have another sprite which represents the role of the familiar in that moment. The background image is also slightly tinted with red, blue or yellow depending on the familiar role as shown in Figure 4.

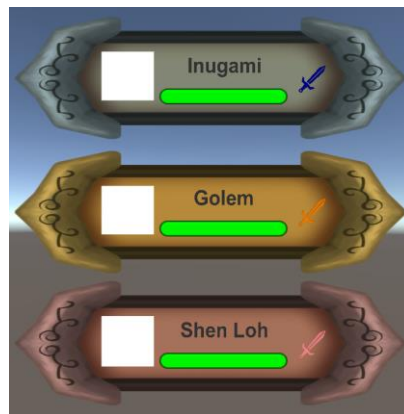


Figure 4 Familiar's GUI

4 Physical & Magical attack: This buttons trigger the avatar's physical and magical attack when pressed. They have an animation to show the delay between attacks.

5 Change Formations: This pair of buttons change the current formation for the next or the previous one, allowing the player to select the desire formation directly.

6 Joystick: This component receive the player's movement input and send it to the PlayerController to move the avatar.

4.2 General Information Classes

To communicate the game with the GQ app an information script is necessary. This script, named *StaticInfo*, holds the user preferences chosen on the app such us the familiars to use and their stats, the configured formations, the player level and stats and so on. The characters' scripts and the game controller take the information they need from this script.

4.3 Game Characters(I1)

The characters of the game are four: The player Avatar, the familiars, the enemies and a special kind of enemy, the dungeon boss. In this section the components and the scripts of each character are shown, explaining their relevant functions but omitting the AIT used to control them, that part will be explained in detail later in Chapter 5.

GameObjects & Components

Before explaining each element individually, the common features are explained. Each element is represented in game by a `GameObject` with four main components. First, a mesh render and a mesh collider to represent the proper model for each element and for the proper collision detection with the rest of the game elements. Second, a `Rigidbody` component to be able to apply the characters' movement physics, again explained later in Section 4.4. And finally a `Script` which makes the character's main controller function.

Furthermore, all the elements have another `GameObject` nested named `FTCanvas` for Floating Text Canvas. As name suggest, is a `Canvas` for showing the character incoming damage during game. It's placed above the character and renders a string of numbers which represent the incoming damage for the character. The numbers appear when the character is damaged, they rise for a few seconds and then disappear. The numbers' color shown the type of the incoming damage, red for physical attacks and blue for magical. If the character is defending themselves, then the number's color is yellow, no matter the damage's type.

Except the player avatar, the rest of characters have another `Canvas` named `GroundCanvas`. It is placed at the bottom of the character and renders different things for the familiars and the enemies. For the first one, the canvas is always active and shows a halo image with different colors depending on the familiar's role. The colors are red for fighters, blue for wizards and yellow for defenders. The enemy's ground canvas renders an image too but is the target sprite. The canvas is normally deactivated and only shows up when that enemy is targeted for the player.

The player avatar has several children `GameObjects` that serve as placeholder for their familiars. They are divided into vanguard and rearguard. The first ones for the fighters and the defenders and the second ones for the wizards. As children of the avatar `GameObject` they move and rotate according to it so they always keep their relative position to it. The familiar only have to seek their placeholder without taking care of the avatar position or orientation.

Scripts

In terms of coding, as said before, each type of character has its own Controller Script. Nevertheless, this scripts share some common features between them. These ones are explained first and after that the particularities of each controller. As before, all about AIT is omitted, to be explained in Section 4.4.

All controllers inherit from the Unity class `MonoBehaviour` in order to use the class' functions `Awake`, `Start`, `FixedUpdate` and `OnMouseOver`. Also, they are related to two main interfaces to distinguish between them, the `IEnemy` interface and the `IPlayerGroup` interface. The first one for the `EnemyControLLer` and the `BossControLLer` classes, the second one for the `Player Controller` and the `FamiliArControLLer` classes. This interface level is necessary

to simplify actions between both groups, for example when an enemy has to choose a target does not care if it is the player or a familiar, knowing it is an *IPlayerGroup* object is enough. Figure 5 shows the functions of each interface. Be noticed that the difference in the number of functions is due to the AIT. As the avatar doesn't use any AIT most of the functions are not needed in the *PlayerController*, they are implemented on the *FamiliarController* and taken by the AI from there directly.

```

public interface IPlayerGroup
{
    Vector3 GetMyPosition();
    void takeDamage(int cant, int type);
}

public interface IEnemy
{
    SteeringManager GetSteeringManager();
    ActionManager GetActionManager();
    EnemyBlackBoard GetEnemyBlackboard();
    IPlayerGroup GetTarget();
    Vector3 GetMyPosition();
    Vector3 GetMyInitialPos();
    float GetWanderRange();
    float GetDetectionRange();
    int GetHP();
    int GetMaxHP();
    string GetName();

    void TakeDamage(int cant, int type);
    void SetAsTarget(bool b);
    void ChooseTarget();
    void LooseTarget();
    void MagicalAttack();
    void PhisicalAttack();
    void Defend(bool b);
}

```

Figure 5 Characters' Interfaces

As can be noticed, the functions' names are self-describing. The first block are get functions that only return the value described, the second are method without any special feature or complicated algorithm.

Going into detail of each controller, the *EnemyController* and the *BossController* are very similar. The main difference is in the AIT used and the behavior, explained in detail in section 4.4. They are composed basically by the functions of the *IEnemy* interface which are now explained:

TakeDamage: The variable "cant" is the value of incoming damage (calculated for the attacker) and "type" is the type of damage (Physical or magical). The method checks if the enemy (or the boss) is defending itself and discount the defense or special defense value to the incoming damage, if it is defending discounts twice the value. Then calls the method to instantiate the floating text in the FTCanvas, updates the enemy hp value and calls to the GUI controller to update to. Finally, it checks if its current hp is less or equal to zero in which case calls the Death method.

SetAsPlayer: the variable "b" is a Boolean to set the enemy as target. If it's true, the GCanvas with the target image is activated.

ChooseTarget: This method looks for the player's familiars and checks if any of them has the defender role. If one has it, it is set as the enemy target, else, the player is set as target.

LooseTarget: Set the enemy's target to null

Physical&MagicalAttack: These methods work the same. They calculate the damage dealt to the target depending on the enemy's physical/magical attack value, with a randomness of 50% of the value.

Defend: this method just set the defending bool to true or false.

The *FamiliarController* share the *IEnemy* methods *TakeDamage*, *Defend* and the attack ones. The specific methods of this type of character are two and are related to the role formation:

ChangeRole: This method receives the new familiar role and changes the familiar's GUI component, the halo image and the behavior.

NewPosition: Assigns the new familiar's position within the formation. This has to be done separately because the *PlayerController* must know all the familiars position in the formation before they can be assigned to each familiar. This is detailed explained ahead.

The *PlayerController* takes the initializer function in the game. It also shares the *TakeDamage* and the attacks methods but its most important methods are the followings:

Awake: In the Unity *Awake* method the *PlayerController* takes care of extract the number of familiars from the *StaticInfo* class and initialize all the familiars. It also calls the next method *ChangeFormation*.

ChangeFormation: This method is called when the GUI buttons for change formation are pressed, this task is divided in two. The first one is the *AskNewPosition* method. This method is used to preconfigure the new familiar's formation. It is called once per familiar and assigns the new role for each of them and place it in one of the two auxiliary list the *PlayerController* has, the front list and the back list. This is because, as said before, the *Player* GameObject has six placeholders for the familiar's formation position but it doesn't place the familiars always in the same way, it depends on the number of front or back familiars the new formation has. Figure 6 shows the different options for two and three familiars formation.

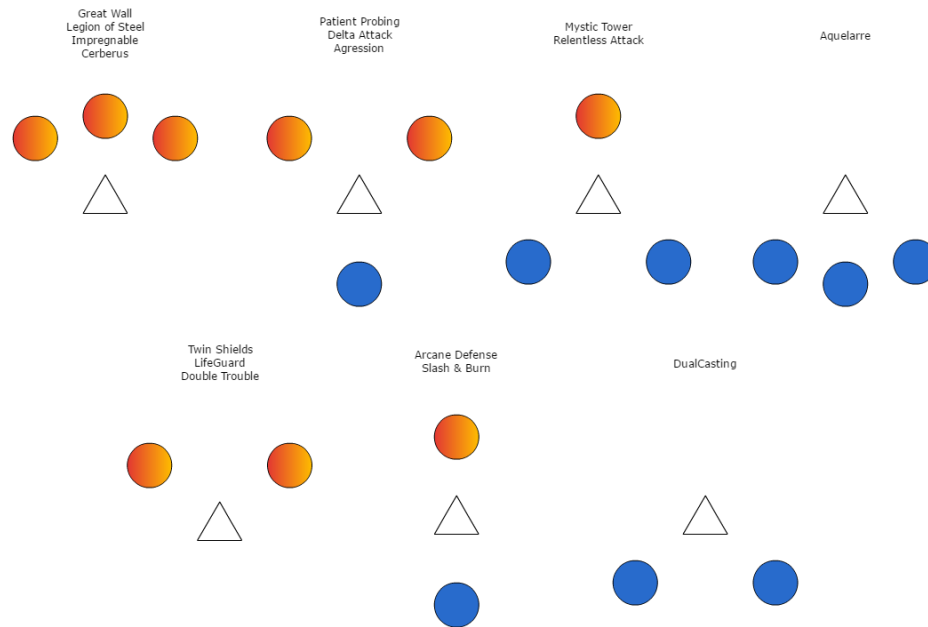


Figure 6 Familiars' Position in all the formation

When all familiars have been placed in one of the two auxiliary lists, the *AssignPosition* method is called. This method checks the two auxiliary lists and assigns a position for each familiar based on the Figure 6 positions.

Move: This method transform the joystick input into the player movement.

4.4 Testing & Debugging

The testing task was taking on during all the implementation work. Each time a new feature was added it was tested and debugged to avoid having a stack of bug at the end of the development of the game.

The testing & debugging of the implementations described in this chapter were quite easy to perform due to the atomic performance of each task. If something was odd, it was simple to track down the error and solve it. This is not the case with the implementations of the next chapter, that's why it has its own Testing & Debugging Section.

Chapter 5

AI Techniques

This chapter explains the AIT used for the familiar and enemy control and accomplish the items I3 listed on Section 1.7. The two main techniques are Steering Behaviors and Behaviors tree. Both are implemented completely from zero, using the Unity tools just in the essential cases. To implement them the book Artificial Intelligence for Games, by Ian Millington and John Funge[6], is used.

Both techniques have a huge potential to be expanded much more than this game does, but this aspect overtake the project's objectives, so the work donned is focused on creating the structures needed for the implementation of each technique and creating only the necessary conducts (steering behaviors) and action (behaviors trees) for the correct game performance.

5.1 Steering Behaviors

This technic is used to perform the character's movement. It applies an addition of different forces to the character's *Rigidbody*. The different conducts are implemented as functions and are called from a *SteeringManager* class which every character has at his Controller. The different implemented conducts are:

Seek: This conduct receives a target position and calculate the force needed to take the character from its position to the target one. It also applies an Arrive conduct, which slows down the character when is close to the target position, in order to avoid an abrupt stop.

Flee: The inverse conduct of Seek. It also implements the Arrive conduct and calculates the force needed to move away the character from the target position.

Wander: This conduct makes the character stroll around an area. To do so uses the Seek conduct to a target which is slightly moved each frame ahead the character.

Stop: This conduct resets the Kinematic to stop applying velocity to the *Rigidbody*, making it to stop.

In order to standardize the different conducts and the characters which implement them several C++ *Structs* and interfaces are needed:

SteeringOutput: This *struct* is used to store the resultant force of each conduct. It contains a *Vector3* representing the linear force resulted and a float for the angle to axis Z.

```
public struct SteeringOutput {  
  
    public Vector3 linear;  
    public float angular;  
  
}
```

Figure 7 SteeringOutput struct

Kinematic: This *struct* holds the physical information needed to update the *RigidBody* component of each character. It contains the basic physical parameters, position, velocity, orientation and angular velocity (rotation).

```
public struct Kinematic  
{  
    public Vector3 position;  
    public float orientation;  
    public Vector3 velocity;  
    public float rotation;  
}
```

Figure 8 Kinematic struct

ISteeringObject: This interface is implemented in all the AI controlled characters' controllers, as known, the familiar, the enemy and the boss controllers. It contains the functions that the *SteeringManager* needs

```
public interface ISteeringObject  
{  
    Vector3 GetMyPosition();  
  
    float getMaxVelocity();  
    float getMaxRotation();  
    float getMaxAcceleration();  
    float  
    getMaxAngularAcceleration();  
}
```

Figure 9 SteeringObject interface

The *SteeringManager* class is created in every character which implements the steering behaviors technique. This class contains the previous conducts as functions. Each time a conduct is called the manager add its resultant *steeringOutput* to the result of the others functions using the *AddSteering* function and save it in a variable named *SteeringResult*. Once per frame, the Update function is called from the character controller. This functions transforms the *SteeringResult* into a *Kinematic* and send it to the controller to be applied. Then reset the *SteeringResult* to zero to handle the new upcoming function calls.

5.2 Behavior Trees

Behavior Tree (BT) technique is used to decide which action will the character do. They are a synthesis of different AI techniques such as Hierarchical State Machines, Scheduling, Planning, and Action Execution. Their strength comes from their ability to interleave these concerns in a way that is easy to understand and easy for non-programmers to create.

A behavior tree's main building block is a task. A task can be something as simple as looking up the value of a variable in the game state, or executing an animation. Tasks are composed into sub-trees to represent more complex actions. In turn, these complex actions can again be composed into higher level behaviors. It is this composability that gives behavior trees their power. Because all tasks have a common root class and are largely self-contained, they can be easily built up into hierarchies without having to worry about the details of how each sub-task in the hierarchy is implemented. Figure 10 shows the task root class for the bt nodes together with the *enum BT_Task_Result*, which holds the different state of a task.

```
public enum BT_Task_Result {True, False, Terminated, Running}

public class BT_Task : MonoBehaviour
{
    public BT_Task_Result result;
    public IBlackBoard blackboard;
    public List<BT_Task> childrens;

    public BT_Task Set(IBlackBoard B)
    {
        blackboard = B;
        childrens = new List<BT_Task>();
        result = BT_Task_Result. Running;
        return this;
    }

    public BT_Task Set(IBlackBoard B, BT_Task[] c)
    {
        blackboard = B;
        childrens = new List<BT_Task>();
        result = BT_Task_Result. Running;
        foreach (BT_Task child in c) { AddChild(child); }
        return this;
    }

    public void AddChild(BT_Task t) { childrens.Add(t); }

    public virtual void Run() { }
    public virtual void Terminate()
    {
        StopAllCoroutines();
        foreach (BT_Task child in childrens) { child.Terminate(); }
        result = BT_Task_Result.Terminated;
    }
}
```

Figure 10 BT_Task Node

Each node contains a *BT_Task_Result* to holds its state of execution, a blackboard to read the information it needs (blackboards are explained in detail later), and a list of children. It also has an *AddChild* function to have new nodes under the node and two virtual

functions Run and Terminate. That a function is virtual means it could be overridden for its subclasses.

The set functions are used as constructors to pass the parameters needed for the node and returns the node itself. But why use this functions instead of a constructor? By doing some search [7] it is said that Unity isn't thread safe, in terms of coordination, and can't interact with other threads. They can be used however to do calculations in background like in any other C# program but this is insufficient for the behavior trees, as is explained later.

Nevertheless, Unity provides *Courutines* [8]. A *courutine* is like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame. That functionality can be derived in some sort of parallel execution, like a thread, and that's why they are used for the BT. This tool is inherited from the *MonoBehavior* [9] class that is why the *BT_Node* uses it. But the problem comes when this type of class is instantiated. As a *MonoBehaviour* the script has to be attached to a game object by the *AddComponent* function, but this function doesn't accept parameters to configure the component. Also a *MonoBehaviour* class can be created with "new *Constructor*()" but will not work properly [10]. This is why the *AddComponent* plus Set functions are used.

Explained that issue, the BT nodes explanation is resumed. Tasks in a behavior tree all have the same basic structure. They are given some CPU time to do whatever they have to do, and when they are ready, update their result with a status code indicating either success or failure. While they are running, the result holds the Running value to indicate the parent node to wait until they finish. When an action is terminated, the result value is terminated.

The BT used for the game consist of four kinds of tasks: Conditions, Actions, Composites and Decorators.

Conditions test some property of the game. There can be tests for proximity (is the character within X units of an enemy?), tests for line of sight, tests on the state of the character (am I healthy? Do I have ammo?), and so on. Each Condition returns the success status code if the Condition is met and returns failure otherwise. Each of these kinds of tests needs to be implemented as a separate task. Due to the information they need they are divided into *FamiliarChecker* and *EnemyChecker*, shown in Tables 9.

Table 9 BT Checkers

Checker Division	Checker Name	What to Check
EnemyChecker	WanderRangeCheck	Am I within the wander range?
	PlayerRangeCheck	Is the player in my vision range?
	TargetNullCheck	Have I a target?
	CombatRangeCheck	Is my target at combat range?
FamiliarChecker	EnemyRangeCheck	Is the enemy in my vision range?
	PlayerTargetCheck	Have the player a target?
	CombatRangeCheck	Is my target at combat range?
	MyPositionCheck	Am I at my formation position?
	AmITargetCheck	Am I the enemy target?

Actions alter the state of the game. There can be Actions for animation, for character movement, to change the internal state of the character ... Just like Conditions, each Action will need to have its own implementation. Most of the time Actions will succeed, if there's

a chance they might not, a Condition is used to check for that before the character starts trying to act. Same as the conditions the actions are divided into *EnemyActions* and *FamiliarActions*. Nevertheless, an action can't be directly called from the BT, Action Nodes are inserted in the behavior tree and they only instantiate their action to be execute by the character. Actions in the strict sense are explained at the end of this section because of the structure needed to perform them. However, a list of the action is show in Table 10 now in order to better comprehend the behavior explanations.

Table 10 Actions

Action Division	Action Name	Performance
EnemyAction	WanderAction	Calls the Steering Manager Wander function
	SeekAction	Calls the Steering Manager Seek function setting the target as one of the IPlayerGroup Characters' position.
	ReturnAction	Calls the Steering Manager Seek function setting the target as the initial position of the enemy. This action is used to return to the enemy's wander zone.
	ChooseTargetAction	Calls the ChooseTarget function from the IEnemy character.
	EnemyMagicAttackAction	Since the action is call once per frame this action holds a counter to only calls the MagicAttack IEnemy function once each 60 frames (more and less one second).
	EnemyPhysicAttackAction	Similar to the EnemyMagicAttackAction but calls the PhysicalAttack IEnemy function.
	EnemyDefenseAction	Calls the Defend IEnemy function.
	EnemyStopAction	Calls the Steering Manager Stop Function.
	LooseTargetAction	Set the enemy target to null
FamiliarAction	GoToPositionAction	Calls the Steering Manager Seek function setting the target as the familiar's formation position.
	LookAheadAction	Rotates the familiar to align its orientation with player's.
	SeekEnemyAction	Calls the Steering Manager Seek function setting the target as the player's target position.
	FamiliarStopAction	Calls the Steering Manager Stop Function.
	FamiliarPhysicalAttackAction	Equal to the Enemy Physical Attack Action
	FamiliarMagicalAttackAction	Equal to the Enemy Magical Attack Action
	FamiliarDefendAction	Equal to the Enemy Defend Attack Action

Both Conditions and Actions Nodes sit at the leaf nodes of the tree. Most of the branches are made up of Composite nodes. As the name suggests, these keep track of a collection of child tasks (Conditions, Actions, or other Composites), and their behavior is based on the behavior of their children. Unlike Actions and Conditions, there are only five different Composite tasks because with only a handful of different grouping behaviors it is possible

to build very sophisticated behaviors. The Composite Nodes used in this game are listed next:

Selector: It runs their children in order waiting the first one to be done before running next. If a child success the Selector stops to run the rest of its children and itself set its result as True. A selector only fails when all of its children does.

Sequence: It runs their children in order the same way as the selector but to run the next one the previous child has to success, if it doesn't, the sequence fails itself and set its result as False. A sequence only success if all of its children does.

NonDeterministicSequence: It behaves the same as a normal sequence but before running its first child it shuffles its children to running them in a different order each time.

Parallel: It runs all of its children in parallel using the *Courutine* function. The parallel success if all of its does, but fails at the moment one of its children does. When the parallel fails, it forces all of its children to Terminate. Using this performance is possible to use a parallel as a trigger, running an action while checking in parallel a Condition. If the condition fails, the action will be interrupted.

A Decorator is a type of task that has one single child task and modifies its behavior in some way. It could be thought of like a Composite task with a single child. Unlike the handful of Composite tasks, there are many different types of useful Decorators.

The name "decorator" is taken from object-oriented software engineering. The decorator pattern refers to a class that wraps another class, modifying its behavior. If the decorator has the same interface as the class it wraps, then the rest of the software doesn't need to know if it is dealing with the original class or the decorator. The same idea is used in the behavior trees. Table 11 shows the list of the two implemented Decorators and explains its performance.

Table 11 BT Decorators

Decorator Name	Performance
UntilFail	Runs its child endlessly until it fails
Inverter	Runs its child and, when it is done, inverts it result.

Once all the node types are explained, it is time to speak about the behavior trees implemented. There are five different behavior trees in the game, one for the enemies, another for the boss and three for the familiar, each per role. Table 12 shows the miniature for every node explained before.

Table 12 BT Nodes Miniatures

Node Type	Node Name	Miniature
Composite	Sequence	
	NonDetSequence	
	Selector	
	Parallel	
Decorator	Until Fail	
	Inverter	
Action	<Action>	
Condition	<Checker>	

Enemy behavior Tree

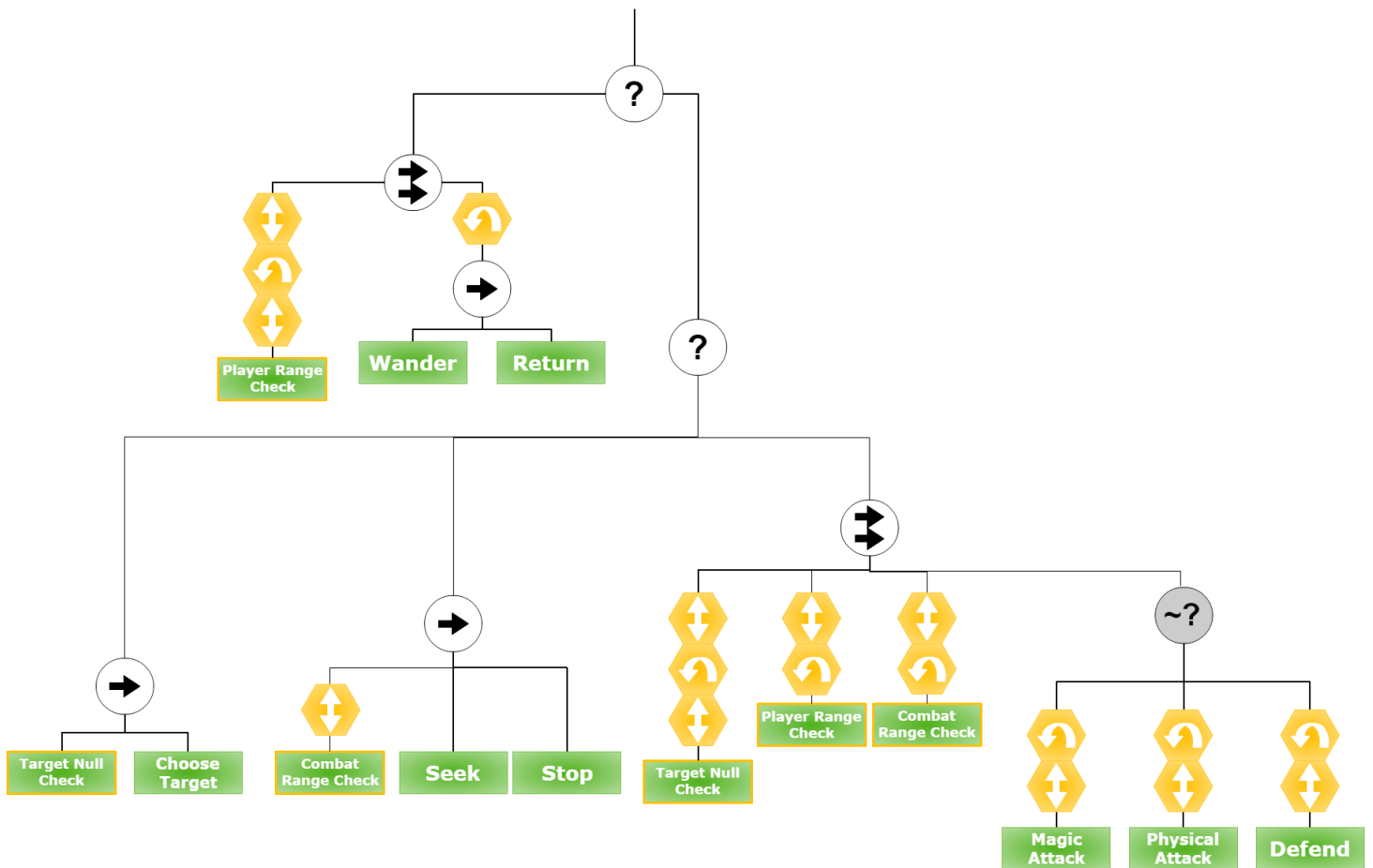


Figure 11 Enemy Behaviour Tree

The boss behavior tree works similar to the enemy BT. The left sub tree checks if the boss is at its initial position and if it's not, go there. It also checks if the player is on its range, while it is not the boss does not do anything. When the player comes in the boss behaves as a normal enemy, choose a target among the player group, seek it till itself is at combat range and then starts the attack subtree.

Familiar Behavior Trees

Each familiar role has its own behavior tree composed by two subtrees, one for movement and another for combat. All three role BT shares the same movement subtree ,but they behave different when battle comes. Figure 13 shows the shared movement subtree in all the roles.

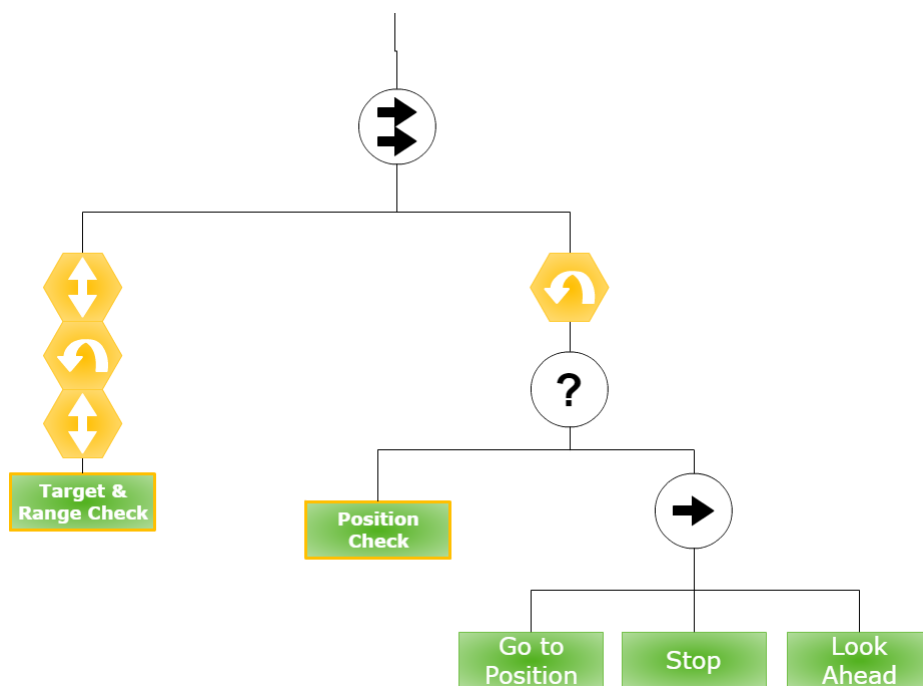


Figure 13 Familiar Movement s

The *target&RangeCheck* makes sure that the player doesn't set any enemy as target or, if it has, that the enemy is not in range of the familiar. While these conditions match, the familiar only concerns if it is at its formation position, going to there if it is out of place. When the conditions are true, that is when the player has chosen a target and it is at range, the parallel node fails and throw the running to the Attack subtree. As said before, this three is different for each role.

Figure 14 shows the Fighter attack subtree. While checking if is still a target and it is at range of detection, first checks if it is at its own combat range, if it's not seeks it and then starts the physical attack.

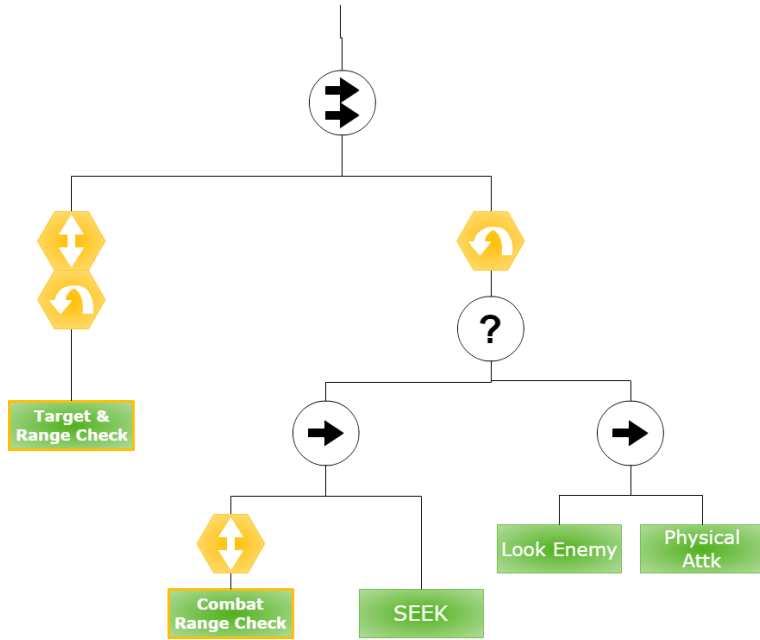


Figure 14 Fighter Attack Subtree

The defender role has the most different attack subtree, as shown in figure 15. Despite the familiar is at combat mode it still stays at its formation position, then it checks if it's the enemy target, defending itself in that case. If it's not the target, then it physical attacks the enemy.

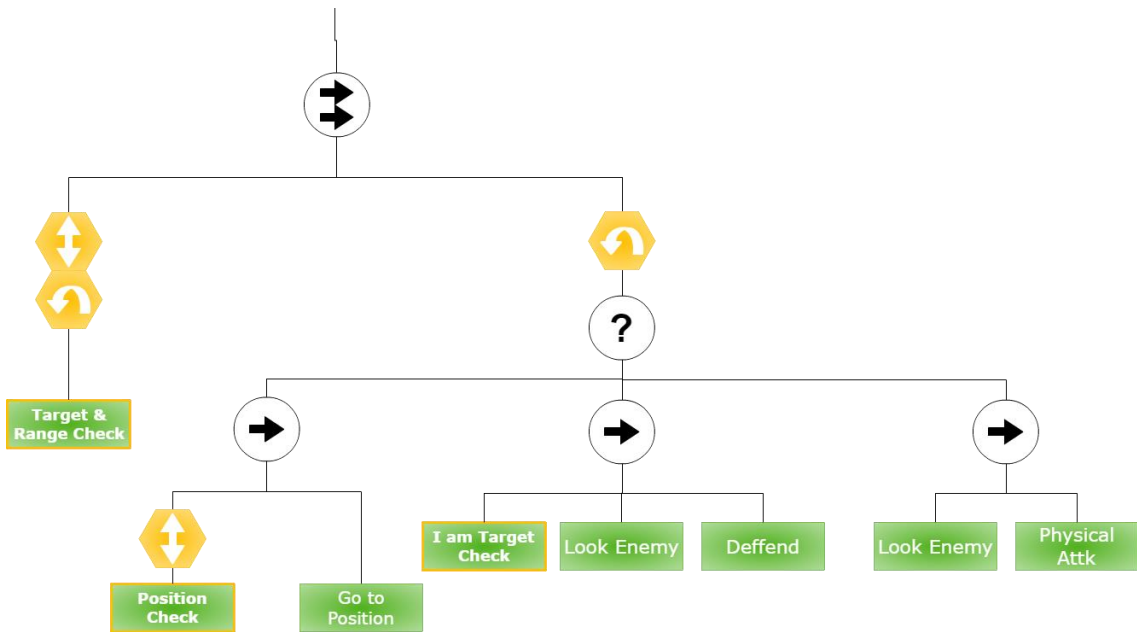


Figure 15 Defender attack subtree

Finally, the Magician attack subtree combines the previous both subtrees. The familiar stays at its position and inflicts magical damage to the enemy. This is shown in Figure 16.

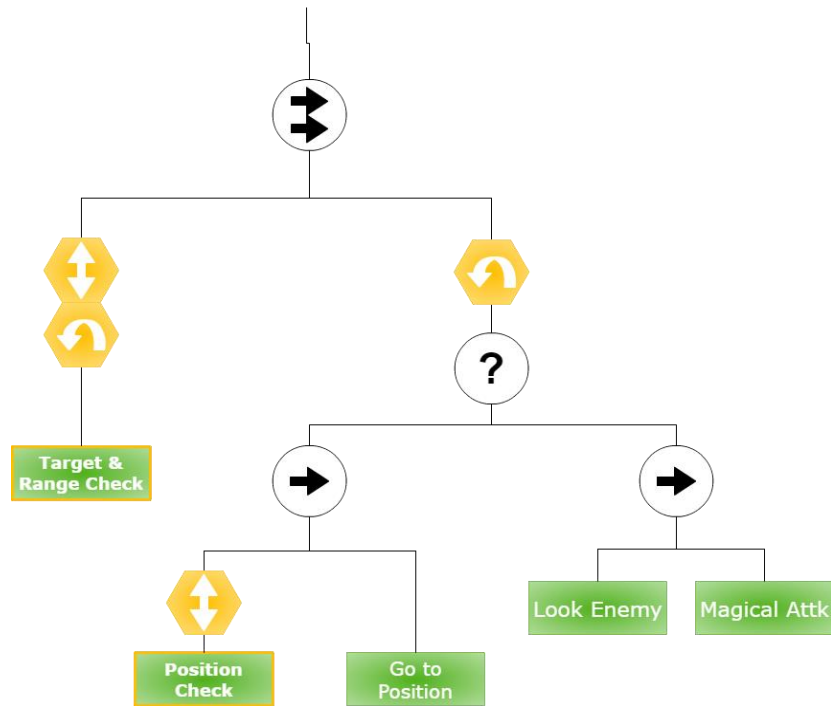


Figure 16 Magician attack subtree

Each Behavior Tree is saved as a class in order to be able to instantiate them at any moment. For the enemies the BT is instantiated on the Start function when the enemy is created. For the familiar, the BT is instantiated when it changes its role, allowing it to change its behavior according to its new role.

5.3 Actions & Action Manager

As mentioned before the leaf nodes of the behavior trees are called action-nodes, but they are not actions, they just create an action and wait until it is finished. So what is an action? An action is the piece of code that makes a character do something. It could be something explicit like seek an enemy or something internal, like choosing a target. The action system is used in order to schedule the different requests the BTs do. Figure 17 shows the father class for all the actions along with the Priority enumerator.

```

public enum A_Priority { Low, Medium, High }

public class Action : MonoBehaviour
{
    public string Name;

    public float expiryTime;
    public A_Priority priority;

    public virtual bool canInterrupt() { return true; }
    public virtual bool isComplete() { return false; }
    public virtual void execute() { }
    public virtual void destroy() { Destroy(this); }
}

```

Figure 17 Action Class

When the BT reaches an action it is schedule with a given Name, Priority and an expiry time, which sets the maximum time the action can be schedule without being executed. The virtual functions are override in each concrete action, similar to the BT nodes. Each action determinates if it can or not interrupt a current action in the *canInterrupt* method. The *isComplete* function checks if the action is or not done. The execute function contains the proper code of the action. In Figure 18 the *EnemySeekAction* is shown as an example. Its execute action calls the enemy *SteeringManager* in order to move the character. In the *SeekAction* the *isComplete* function checks if the distance to the target is minor than the *COMBAT_RANGE* distance.

```

public class SeekAction : EnemyAction
{
    Vector3 targetPosition;

    public SeekAction Set(IEnemy ene, float ext, A_Priority p, string n)
    {
        Name = n;
        expiryTime = ext;
        priority = p;
        base.Set(ene);
        return this;
    }

    public override bool canInterrupt() { return true; }

    public override void execute()
    {
        targetPosition = blackboard.GetTargetPosition();
        steeringManager.Seek(targetPosition, 30, (StaticInfo.COMBAT_RANGE / 1.5f));
    }

    public override bool isComplete()
    {
        return Mathf.Abs((blackboard.GetMyPosition() - targetPosition).magnitude) <
(StaticInfo.COMBAT_RANGE / 1.5f);
    }
}

```

Figure 18 Enemy Seek Action

In order to manage the actions, each character has an Action manager which takes cares of all the actions request made by the Behavior Trees and schedule the character's actions. The

pseudocode for this Action manager appears in Figure 19 and explains the Action Manager algorithm.

The *execute* function performs all the scheduling, queue processing, and action execution. The *scheduleAction* function simply adds a new action to the queue. The *TerminateActive* function get rid of the current active when it is finished or when is interrupted. The *ChangeBehavior* function is used for the familiars and change the BT referred in the variable *behavior*. It is used when the familiar changes its role.

The *Set* functions is needed as in the BT Nodes. In order to work correctly, the Action Manager class has to inherit from the *MonoBehaviour* class, so the *Set* function replace the *Constructor* of a normal class.

In order to achieve a proper efficiency, the queue variable has to be implemented as a priority heap. That way the actions are sorted by their priority and when the execute function checks the queue actions can interrupt when it reaches an action whit less priority than the currently active, avoiding going over all the queue actions.

```

class ActionManager

    behavior          //Holds the behavior tree used by the character
    sManager          //Holds the character Steering Manager
    queue             //Holds the schedule actions sorted by priority

    active            //The current action
    currentTime       //The current time

    void Set()

        //Sets the variables when the Action Manager is initialized

    void scheduleAction(Action)

        //Adds an action to the queue

    void Execute()

        //It is called once per frame and processes the manager tasks

        Update the time

        If there is no active action
            If the queue is empty
                If the BT is not running
                    Runs the behavior tree again
                Else
                    Do nothing this frame
            Else
                Dequeue an action and set as active

        //Checks for interrupters in the queue

        Foreach Action in the queue
            If the action expiration time exceeds the current time
                Dequeue and destroy the action
            If the action has lower priority that the current active
                Break
            If the action can interrupt
                Terminate current and set the action as active

        //Process the current action

        If active is complete
            Terminate active
        Else
            Execute Activate

    void TerminateActive()

        //Destroy the current action and set active to null

    void ChangeBehaviour(IBehaviour)

        //Changes the character behavior tree

```

Figure 19 Action Manager Pseudo-code

5.4 Blackboard

As seen so far, the behavior trees are created in a modular way that allows them to be instantiated in any character which has an action manager and a steering manager. However, there is another component needed in order to correctly run the character's IAT. The widespread characteristic of the behavior trees interferes with the information they need to run. They need to collect information from the character controller and the `GameObject` of it such as its position, its rotation, if it has a target ... In order to make it compatible with any character type, this information has to be stored in some common structure the BT can access without taking care of which type of character it has.

This structure is called Blackboard and Figure 20 shows the interface of it. These classes communicate with their characters and brings the BT Nodes the information they need. Despite its intention of unify all the information it is necessary to distinguish two types of blackboards, one for enemies and another for the familiar. This is because some variables will be from different types depending on the character, for example the target variable will be an *IEnergy* item when the character is a familiar and an *IPlayerGroup* item when it is an enemy or the boss. Furthermore, the behavior trees are already separated in *EnemiesBehaviors* and *FamiliarBehaviors* and each of them needs different information so treating the blackboards differently fits perfectly.

```
public interface IBlackBoard
{
    GameObject GetActionManager();
    PlayerController GetPlayer();
    Vector3 GetPlayerPosition();
    Vector3 GetTargetPosition();
    Vector3 GetMyPosition();
    float GetDetectionRange();
}
```

Figure 20 Blackboard Interface

5.5 Putting all together

Figure 21 shows a complete AI structure for a character using the action manager, the behavior trees and steering behavior techniques and the blackboard system.

The Character controller holds the Action manager and calls it every frame. The Action Manager gets its next action from the queue. There is a single Behavior Tree, which is called whenever the action queue is empty. It takes the information it needs from the Blackboard which takes it from the character controller. Finally, when an action is executed the output is a change in the character controller state or an update of the steering manager, which involve moving the character's *Rigidbody*.

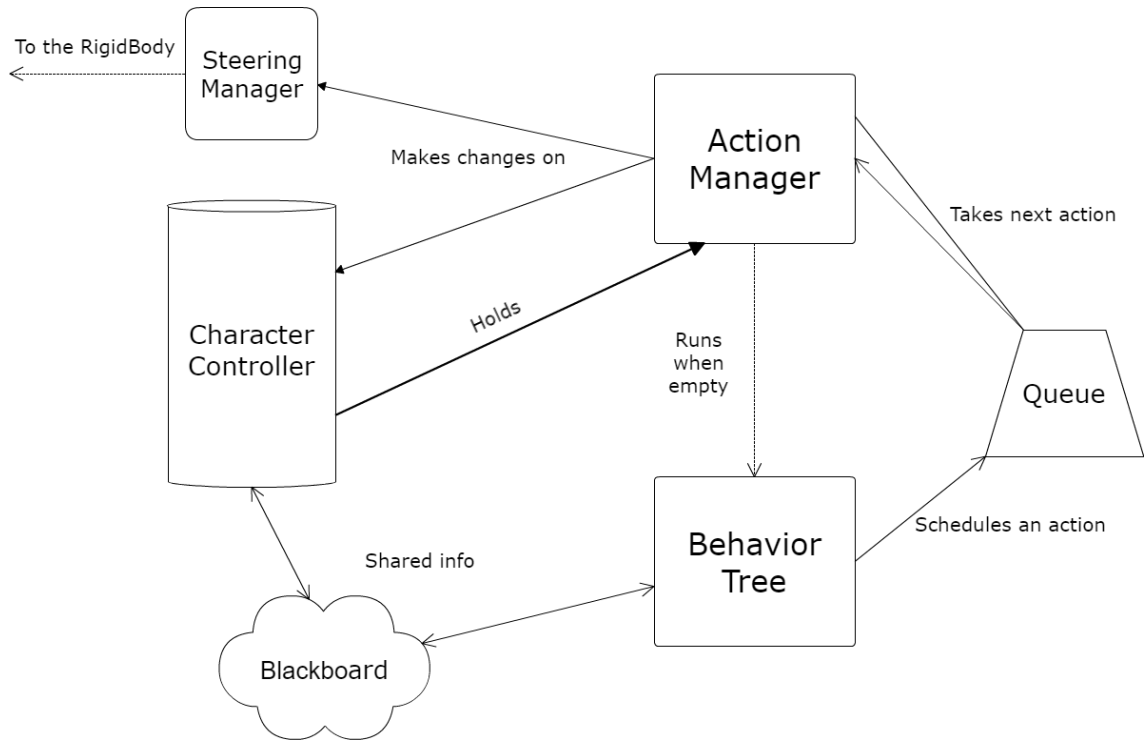


Figure 21 The complete AI Structure

5.6 Testing & Debugging

The testing & Debugging of this AI techniques was extremely time-consuming and complicate. Not only because of the complexity of each technique, but also because of the interactions between them.

First the correct implementation of the steering behaviors took quite more time than the expected. It was pretty complicate to accomplish the stop of the character. At first, when the seek action was called, the character started to move but when it reached its target and the action was over the *SteeringManager* continued to apply forces to the *Rigidbody*, not only causing the character not to stop, but also accumulating forces into the kinematic. The result was a complete nonsense movement. Once this bug was fixed, the Steering behavior works without further issues.

The implementation of the BT was also very hard to debug. The main issue was that the nodes worked without any error but the resulting behavior was not the desired. This leads to redesign the different trees several times until the correct characters' behavior was achieved, digging deeply in each node behavior and output.

The last main issue was the communication between the BT and the action manager. Due to the performance of the courutines the execution was not really parallel so there were times were the *ActionManager* destroyed the current action because it was completed but the behavior tree does not have time to check if it was. So the execution of the BT was hold trying to get a response from an action already destroyed. Become aware of this issue was extremely complicated because it was an issue due to Unity concurrency, not for a code bug.

Chapter 6

Game Art

In this section the art created for the game is shown, both the 2D and the 3D models. This work corresponds to the art planning listed in Section 1.7.

6.1 Visual Identity

As pointed by the project's objectives, the game has to be integrated within the GQ application. In order to achieve that the game mechanics have a very important role as explained before but, from the player's point of view at least, the visual look of the game is equally important, even more so. Figure 22 shows a screenshot of the GQ app Main Menu. All the art created for the game is designed using the same aesthetic and the same techniques in order to achieve the resemblance. Even some of the sprites used are recycled from the GQ app.



Figure 22 GQ Main Menu Screenshot

6.2 2D Elements

In this section the sprites created for the GUI are listed in Table 13.

Table 13 2D Game's Elements

Sprite Code	Sprite Name	Sprite	Notes
S_01	Stats Background		Modified from a GQ Sprite
S_02	Player's Face		-
S_09	Physical & Magical Attack		Combination of two GQ Sprites
S_11	Formation Button		Modified from a GQ Sprite
S_13	Familiars' Halo		-
S_14	Target		-

6.3 Characters

The only character modeling for the game is the played Avatar. The familiars and the enemies are made by Marina Granell as she also uses the gamification app in her project.

Figure 23 shows the avatar designs used as blueprint for modeling. The avatar must fit any type of player so the design does not have to show any distinctive trait.

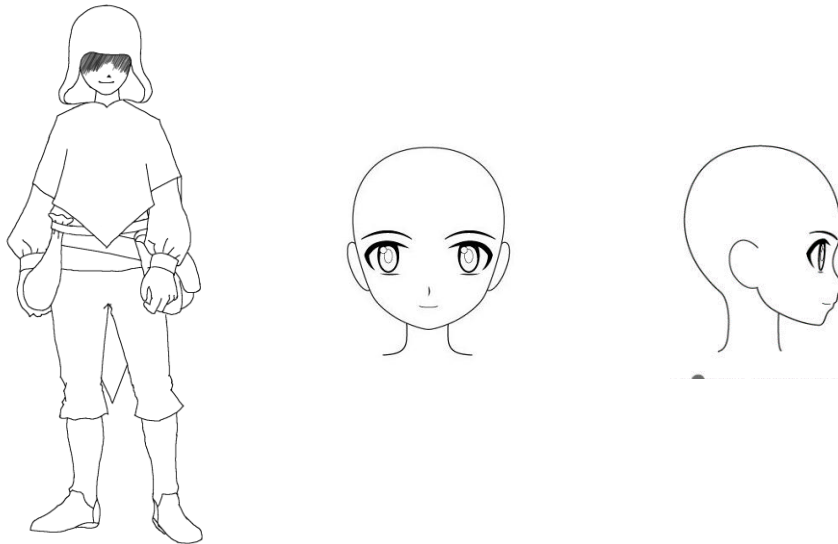


Figure 23 Avatar's Designs

The avatar 3D model is modeled using primitives for every different part of the body. Despite the first design of the avatar wears a poncho the final model only has the hood because the folds of the poncho need a lot of polygons to fit the avatar's movement and that amount of polygons is not viable to run the game on a mobile platform. Figure 24 shows some key points of the avatar's model.

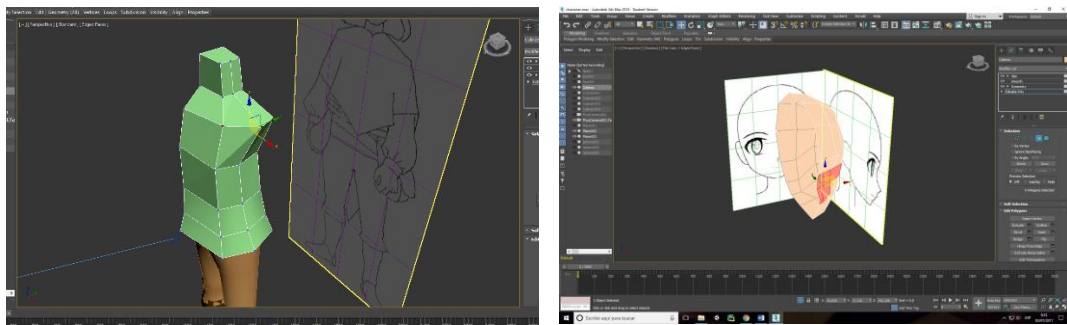


Figure 24 Avatar's modelling

The final model shown in Figure 25 has a total of 1940 polygons, this amount is perfectly suited for running in a mobile phone.

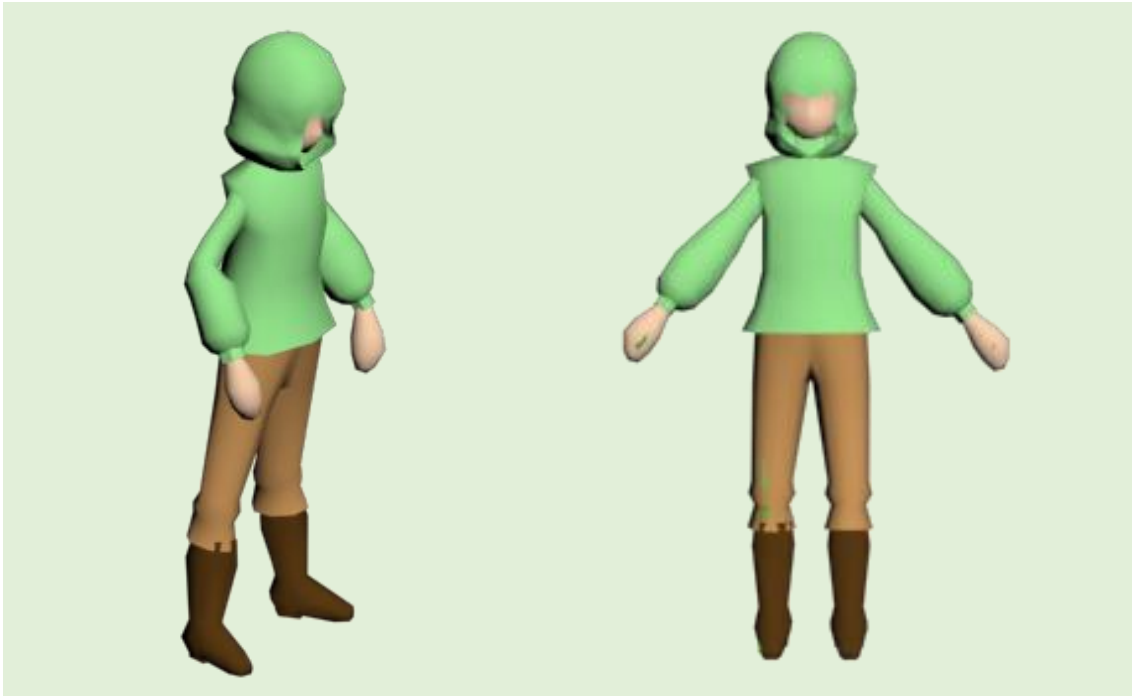


Figure 25 Avatar's final model

The rigged and the skinned of the avatar is made using the biped system of 3Ds Max and the envelopes of the skin component. Skinning means to attach the mesh polygons to a bone. That makes the mesh to move with the bones it is attached during the animations.

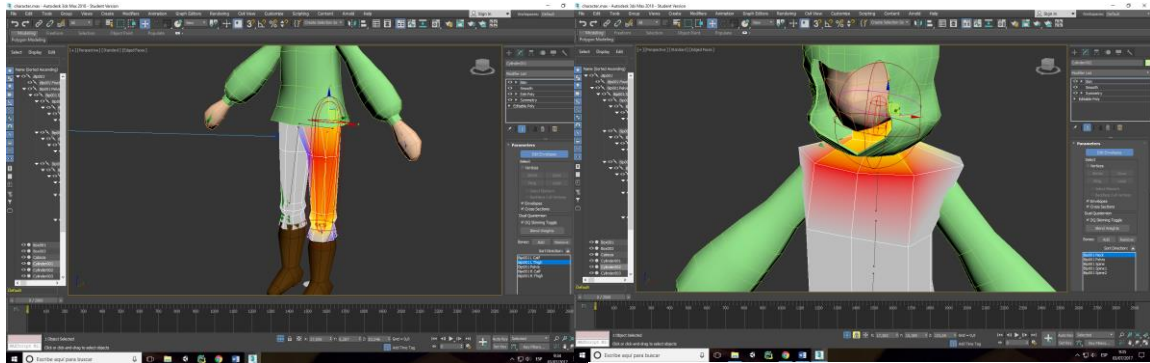


Figure 26 Avatar's skinning

The animations are download from the LexLuthor1's contributions in the forum of unreal engine [11].

5.4 Environment (I2):

The game locates inside a dungeon so the environment elements have a dark aesthetic. The Table 14 shows the different elements created for the project along with their textures.

Table 14 Environment Elements

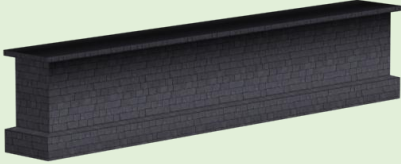

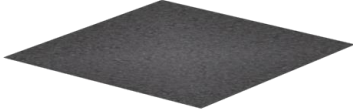
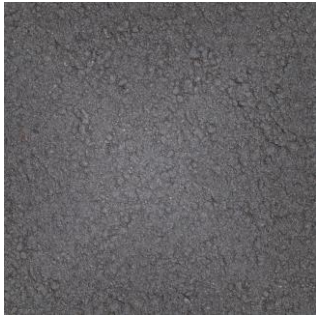
Element Code	Element Name	Model Render	Texture
E_01	Wall		
E_02	Ground		

Figure 27 shows the final map of the level with all the rooms of the dungeon.

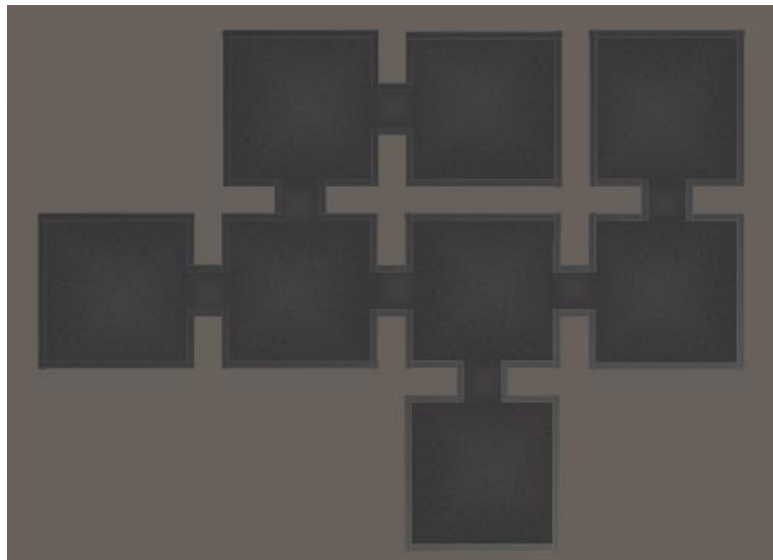


Figure 27 Dungeon Level

Chapter 7

Results

In this section the result of the work done for the project is summarized. For do it, a list of elements is presented. Each of the element is evaluated and compare with the initials objectives of the project enumerate at section 1.

6.1 Technical Proposal

At the beginning of the project a Technical proposal was delivered with date 12/2. This document presents a first approach to the project before any work was done and serves as base for this report's Chapter 1.

6.2 Game Document Design

The Chapter 3 section of this document is the GDD of the GQ* game which contains all the specification and design elements of the game. This document and the mechanics detailed inside it satisfy the design part of the second objective listed on section 1.4: "O2 - Design and implement fun game mechanics that integrate the use of complex AIT."

6.3 Technical Report

This same document is another result of the work done. It has a detailed explanation of the work done for the project together with a technical overview of the game.

6.4 Gamification Quest: *

The implemented game, Gamification Quest: *, is probably the most tangible result of the project. The application is available at the link and QR code the Figure provides. In order to install it in your android device you have to allow the installation of apps from unknown sources in the security related options on the smartphone.

<https://drive.google.com/file/d/0B2Vlwmho0tTSbtZUWVFVeWpXYWM/view?usp=sharing>



Figure 28 Download code

Figures 29,30, and 31 show several screenshots of the game final result.



Figure 29 Formation Screenshots



Figure 30 Combat Screenshots

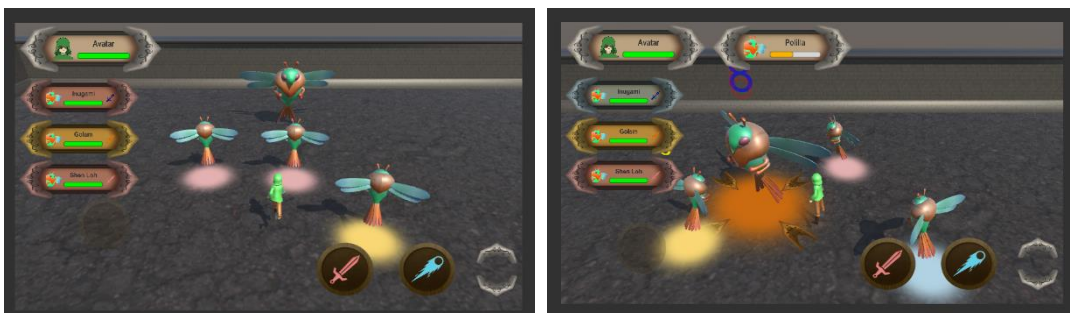


Figure 31 Boss Combat Screenshots

In order to summarize the work done for the project, Table 15 shows a final recount of different elements done for the project

Table 15 Project's Item Count

Item	Value
scripts	16
code lines	About 3750
Images created	10
Models created	2
Game levels	1

6.5 Project Exhibition Video

A video summarizing the game development will be created for the exhibition on 10/6. Currently in development.

6.6 Project Defense Presentation

For the defense of the video a presentation will be performance. Along with a PowerPoint it will summarize all the work done for the project.

Chapter 8

Conclusions

This document has followed all the work done during the Gamification Quest: * development and in this section the conclusions of the experience are exposed. In first place, this section value the deviation from the initial proposal of the project with the actual final result, considering mostly the project's objectives shown in section 1.4, the project schedule expose in section 1.7, and the risks listed on section 1.8. In second place, a section considering possible future development plans.

Project Objectives

Once the project is finished it is time to look to the objectives established at the beginning of the work and check if the project's results fulfill them.

- O1 - Implement a video game which takes advantage of the Gamification Quest app.

As can be extracted from the Chapter 3 the mechanics of GQ: * suits in the game-flow of the Gamification Quest up. The mechanics of both are designed to feed between them and, together, motivate the student to increase their academic performance.

- O2 - Minimize the use of Unity tools to implement the AIT used on the game, creating them from scratch whenever possible.

All the AI Techniques used on the project are implemented from scratch mostly. However, there are two issues where the project falls into Unity tools.

First is the Quaternions system for rotation. Due to my lack of knowledge in the mathematical aspect of the Quaternions it turns impossible to suit the angle calculations of the Steering Manager into the Quaternions, demanded for the Transform Component to apply rotations to the GameObjects.

Second is the Unity physics engine. The Steering behaviors indeed calculate from scratch the force needed to apply to the GameObject to reach its destination, but transform that force into movement is something that overdue the projects boundaries.

- O3 - Design and implement fun game mechanics that integrate the use of complex AIT.

As seen in Chapter 3 the mechanics designed and implemented for GQ: * are meant to include the AIT created for the game.

- O4 - Implement the familiars so that they respond in real-time to the player's orders.

The AI techniques implemented allow the familiars to change its behavior according to the role the player assign to them, and the reactive behavior of the BT allows the characters to respond to their environment in real time.

Project Deviation

In Section 1.7 a schedule for the project was exposed. In this section that schedule is looked over and corrected to reflect the real work done for the project. Table 16 list all the element named in Section 1.7 with their estimate work hours along with the real time expended in each task.

Table 16 Project's Schedule Deviation

Task ID	Task Name	Time estimation	Time required	Notes
D_01	Mechanics Design	7	10	-
D_02	Characters Design	17	-	Note 1
D_21	3 Familiars Design	10	-	Note 1
D_22	4 Enemy Design	7	-	Note 1
D_03	Level Design	6	6	-
I_01	Mechanics Implementation	20	24	-
I_11	Character Mechanics	5	9	-
I_12	Enemies Mechanics	5	2	-
I_13	Familiar Mechanics	10	13	-
I_02	Scene Implementation	20	14	-
I_21	Environment	3	3	-
I_22	Camera Movement	2	1	-
I_23	Object System	5	-	Not implemented
I_24	Enemies Respawn	5	-	Not implemented
I_25	User Interface	5	10	-
I_03	Artificial Intelligence Techniques	80	-	Note 2
I_31	Familiars AIT Design	7	-	Note 2
I_32	Enemies AIT Design	3	-	Note 2
I_33	Familiars AIT Implementation	45	-	Note 2
I_34	Enemies AIT Implementation	25	-	Note 2
I_04	Testing & Debugging	45	60	-
A_01	Blueprints	14	-	Note 1
A_11	Familiar's Blueprints	6	-	Note 1
A_12	Enemies' Blueprints	8	-	Note 1

A_02	Modelling	35	-	Note 1
A_21	Familiars' Modelling	15	-	Note 1
A_22	Enemies' Modelling	20	-	Note 1
A_03	Animations	21	-	Note 1
A_31	Familiars' Animations	9	-	Note 1
A_32	Enemies' Animations	12	-	Note 1
A_04	Game Integration	10	10	-
Doc_1	Technical Proposal	10	25	-
Doc_2	Game Document Design	10	-	Note 3
Doc_3	Final Technical Report	15	60 + 20	Note 3

Note 1: At the beginning it was schedule that 4 enemies and 3 familiars will be design, modelling and animated. Finally, I was meant to do the avatar design, modelling and animation. This change the initial schedule adding new tasks as shown ahead in Table 17.

Note 2: Due to my lack of knowledge in this subject the tasks needed to do were completely different to the schedules ones. This is reflected in Table 17 as well.

Note 3: Initially the game document design and the final technical report was meant to be different documents but, in order to unify the work done, they result in one document. That is why the Doc_3 has 40 hours (Doc_3) + 20 hours (Doc_2)

As seen in the previous notes, new tasks were added to the schedule for the project's work. This new tasks, along whit their required time, are shown in table 17.

Table 17 New Schedule tasks

Task Name	Time required
AI Techniques research	10
Steering Behaviors Functions	10
Steering Behaviors Manager	2
Behavior Tree Nodes Implementation	20
Behavior Trees Implementation	2
Actions Implementation	15
Action Manager	9
Blackboard Implementation	5
Avatar Design	2
Avatar Modelling	10

Whit this new schedule the project total hours of work turns to be 365 hours, far ahead the initial estimated time.

Project Risk

In Section 1.8 some risks were considered which could have taken down the project schedule. This section takes a look at these risks and listed the ones that have occurred and the contention plan used to deal whit them.

The probability pointed in Table 5 was pretty accurate and almost all the risks have occurred. Fortunately, the data loss hasn't occurred but its contingency plan was carried through anyway.

The first risk occurs in some minor tasks, I_23 (Objects system) and I_24 (Enemy Respawn). The no implementation of these tasks does not affect highly to the project result so they were put aside.

The second risk happen too and the containment plan was carry out. The familiar model is take from Marina Granell project's since we both work with the GQ app.

The risks three and five are already considered in the previous section, adding new tasks to the project's schedule, one of them the AIT research.

Future Work

This section considers new horizons for the GQ: * project. In general terms, the time to devote to the project was short so the first future work to consider is the plenty implement of all the features mentioned in the game document design.

Integrate the project with the GQ app and implement the mechanics needed to do so is the next logical future work to do. Once this is accomplished, the whole Gamification Quest project will be complete but it still has the opportunity and the resources to grow up more.

As mentioned in the beginning of chapter 5, the AIT implemented for the project has the capability to perform a lot more complex behaviors so design them and implement the subsequent actions is something to be considered. Also, due to the modular fashion of all the AI components, the game has the opportunity of implement another Decision making AI techniques asides from BT, like Finite state machines, neuronal networks, decision trees, ...

Chapter 9

References

- [1] Gamification from en.wikipedia.org/wiki/Gamification
- [2] ClassDojo from www.classdojo.com/es-ES/
- [3] Final Fantasy XIII from <http://finalfantasy13game.com/>
- [4] Final Fantasy XIII Paradigm System from <http://finalfantasy.wikia.com/wiki/Paradigm>
- [5] VJ1241 Teaching Guide from https://eujier.uji.es/pls/www/gri_www.euji22883.html?p_curso_aca=2016&p_asignatura_id=VJ1241&p_idioma=CA&p_titulacion=231
- [6] Artificial Intelligence for Games (2nd edition), by Ian Millington and John Funge. Elsevier, 2009. Chapter 5 Decision Making Available online: <http://www.sciencedirect.com/science/book/9780123747310>
- [7] Unity Answer Forum: Threading in Unity , from <http://answers.unity3d.com/questions/180243/threading-in-unity.html>
- [8] Unity Documentation: Coroutines from <https://docs.unity3d.com/Manual/Coroutines.html>
- [9] Unity Documentation: MonoBehaviour from <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>
- [10] Unity Answer Forum: new MonoBehaviour from <http://answers.unity3d.com/questions/653904/you-are-trying-to-create-a-monobehaviour-using-the-2.html>
- [11] Unreal Forum : Character Animations download from <https://forums.unrealengine.com/showthread.php?1501-Free-Character-animations-in-bip-and-FBX-Download>

Appendix A

Sound Design Document by Victor Avila

La banda sonora Gamification Quest ha sido compuesta para trasladar al jugador al mundo de fantasía con toques medievales que se plantea en el juego. Para ello se decidió optar por unas melodías definidas desde un principio y ritmos bien marcados, dando dinamismo a las partidas.

Elegimos temas inspirados en obras populares medievales, tanto en sus armonías como en los procesos rítmicos, y por la estética del juego optamos por temas alegres a medios tiempos intercalando melodías menores, dando citas a los enemigos que vas a encontrarte en todo el camino

Tenemos claro que el soporte del juego son teléfonos o tablets, es por ese motivo que adaptamos las sonoridades a estos dispositivos, pero cuidando al máximo la calidad de las bibliotecas de sonidos utilizando, llegando a rediseñar cada sonido utilizado para que casen perfectamente unos con otros, y no suene excesivamente artificial.

Cada sonido incidental utilizado ha sido diseñado específicamente para que el jugador, conciba la acción del juego tanto visualmente como auditivamente, pasos, magias etc.

En resumen, esta banda sonora ha sido un resultado del trabajo conjunto entre historia, jugabilidad y estética del juego, ofreciendo una experiencia tanto divertida como adictiva en el plano sonoro.