TÉCNICO LISBOA

A Procedural Quest Generator for Mount & Blade II: Bannerlord

Marco António dos Santos Cabral

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Alexandre Simões dos Santos
Prof. João Miguel de Sousa de Assis Dias

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Pedro Alexandre Simões dos Santos
Member of the Committee: Prof. Carlos António Roque Martinho

November 2021

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

I would like to first give my thanks to my dissertation professors Pedro Santos and João Dias for giving me the opportunity to face this difficult project. The guidance, insights and advice they provided proved indispensable for the development of this project and the constant feedback helping improve it.

To my parents, I would like to express my deepest gratitude, for being there for me with constant support, in the ups and downs, through all the tears and laughs. For teaching me to fight for what I want to accomplish and for staying by my side throughout the whole journey and for helping me to finally see the end of it, so a new one can be started.

I would like to thank my family for their unconditional support no matter the distance and hardships we faced. To my aunt, Lilina and my grandfather, Raul, whom I won't be able to see anymore, a final thank you for the constant support and encouragement in pursuing higher education.

To my friends and colleagues, thank you for your support as well. For always being there to listen to my complaints and for helping me laugh them off every time.

And last but not least, a special thanks to the colleagues that joined me on this journey, David Ricardo, Filipe Silveira, Joaquim Quadrado and Marcos Pêgo. The mutual help we provided each other and comradeship pushed all our thesis to become great projects.

To each and every one of you – Thank you.

Abstract

We propose for a procedural quest generator for the game Mount & Blade II: Bannerlord. Although Bannerlord is a game that already has a questing system, the quest it presents are repetitive, leading to a gameplay experience with a tendency to become dull. With the procedural quest generator, we aimed to implement a questing system that can create an almost endless amount of quests that seem different from one another, providing the player with a prolonged and varied experience. This generator can also allow for a lower cost of production when creating quests since a game developer wouldn't have to spend so much time creating every single individual quest. The implemented system was based on a group of grammar rules that were inspired by the work developed by Doran and Parberry for quest generation. We will also analyze the different state-of-the-art engines made towards interactive storytelling to make sure the developed quests are coherent and believable in the storyline of the game. To properly evaluate the developed system, we conducted user tests and published the resulting mod online. These tests, which evaluated the quality and enjoyment of the quests, showed that our system was able to generate quests that could be enjoyed by players and presented a quality comparable to the Bannerlord quests.

Keywords

Mount & Blade II: Bannerlord; Procedural Generation; Quests.

Resumo

Proposta para um gerador de quests procedimental para o jogo Mount & Blade II: Bannerlord. Embora Bannerlord seja um jogo que já possui um sistema de missões, as missões que apresenta são repetitivas, levando a uma experiência de jogo com tendência de se tornar monótona. Com o gerador de missões procedimentais, pretendemos implementar um sistema de missões que pode criar uma quantidade quase infinita de missões que parecem diferentes umas das outras, proporcionando ao jogador uma experiência prolongada e variada. Este gerador também pode permitir um menor custo de produção ao criar missões, já que um programador de jogos não teria que gastar tanto tempo a criar cada missão individual. A nossa principal inspiração será o trabalho anterior desenvolvido por Doran e Parberry. Também analisaremos os diferentes engines de última geração desenvolvidos para a narrativa interativa para garantir que as missões desenvolvidas sejam coerentes e verossímeis no enredo do jogo. Estes testes, que avaliaram a qualidade e a satisfação das missões, mostraram que o nosso sistema era capaz de gerar missões que podiam ser apreciadas pelos jogadores e apresentavam uma qualidade comparável às missões do Bannerlord.

Palavras Chave

Mount & Blade II: Bannerlord; Geração Procedimental; Missões;

Contents

List of Figures

1

Introduction

Contents

Gaming has been an ever-increasing market, with each year getting more and more players. With this increase in demand, game companies have been spending even more time and funds creating their games than before, however, the consumption of the content present in the games is much faster.

This situation can be especially verified in the Open World Role Playing Games(RPGs) genre of games. This genre gives the player the possibility to play as a character or more from the game world that must complete quests, that might be connected or not, and follow the general story to reach an end goal and the conclusion of the storyline. The players can also develop their characters through the several decisions presented to them.

These genres are known for being content-heavy, with lots of non-playable characters (NPC's) together with a vast open-world to explore. All of this is used to create both the main quests and side quests of the game. The problem starts when a player completes all the quests and accomplishments (like collectibles) available to him, since it makes the replay value of the game decrease substantially. This makes these genres of games some of the perfect candidates to implement a Procedural Quest Generation System.

With the use of automatically generated quests, the players can both enjoy an almost endless game experience, and also replay the game without repeating the same quests over and over again. This also allows the developers to spend less time writing countless quests and more time refining other areas of the game. With this automatic generation of content, we can furthermore learn the player gameplay preferences, desires, abilities, and such and have the content presented to them adapted, possibly giving them a more personalized experience.

1.1 Problem Definition and Thesis Goals

One example of the kind of game talked about above is Mount & Blade II: Bannerlord (commonly referred to as only Bannerlord). Bannerlord is an Open-World, RPG type of game part of the Mount & Blade series with a medieval setting, where the player moves its character through the virtual world, completes quests (or missions) and conquers towns and villages to achieve its goal. At the time of writing this thesis, it possesses a somewhat limited and repetitive questing system. Each game day, a random number of quests are randomly chosen and applied to the NPC's that can receive them, for the player to complete or ignore. Although this allows for an endless experience for the player, since even though the main quest ends, the side quests are continuously assigned, it quickly becomes repetitive and tedious. This is a situation that might however change or improve, since the game is still in development.

So one of the goals here is to have a system that creates somewhat interesting quests, that present more variation in both type and structure, through the use of an automated algorithm that the game can then read and transcribe to in-game quests in order to lower the production costs of the quests while

maintaining a similar quality to the quests that are manually created. To accomplish this, we took a look at procedural quest generation systems to better understand how we can apply this concept to the game Bannerlord.

We analyzed other procedural quest generation systems, and we inspired ourselves in the work done by Jonathan Doran and Ian Parberry [3] with the modifications applied by António Machado [6]. This work is a quest system based on RPG games that presents a lot of quest variation, and we made an adaptation of it to apply in the game Mount & Blade II: Bannerlord.

1.2 Organization of the Document

This thesis is organized as follows: In Chapter 2, Related Work, we review the main work used on this thesis and the modifications done to it. We also review some storytelling systems. In the next chapter, Chapter 3, The Game - Mount & Blade II: Bannerlord, we focus on the target game of this thesis, its architecture and how the modding of the game works. Afterwards, we have Chapter 4, Bannerlord Quest Generator - BQG. In this chapter, we talk and show the system we implemented on the target game. Next, in Chapter 5, Evaluation, we present the results of the tests we made to assess the quality of our system. Finally, we reach Chapter 6, Conclusions and Future Work, where we discuss the test results, conclude this thesis and talk about possible future work.

2

Related Work

Contents

The focus of this work will be procedural quest generation, so we first need to define what a quest is. A quest or mission is a task given to a player, and it's usually found within role-playing games. These tasks create some sort of challenge the player needs to overcome to reach a goal and receive a reward. From the point of view of the author of the game, a quest will introduce much of the difficult aspects of gameplay and give the players concrete goals. From the player's point of view, the quest is a storytelling feature that tells them about the world and helps the player to acquire knowledge and control. Procedural generation is a means of creating data algorithmically rather than manually, using a combination of human-generated materials and algorithms, as well as computer-generated randomness and processing power. Let's now look at the work that had the most influence on this thesis.

2.1 Doran and Parberry's Prototype Quest Generator

According to Doran and Parberry's work on a prototype quest generator derived from the analysis of quests from MMORPGs (Massively Multiplayer Online Role-Playing Game) [3] a procedural quest generation system has the potential to increase the variability and replayability of games, leading to an increase in player interest in these games, since there is never a moment where the player has experienced and completed everything in the game. Through the analyses of almost 3000 quests from several games, Doran and Parberry concluded that even human-authored quests display significantly less structural variety than one might expect. By exploiting this common structure between the quests, they were able to show a prototype system that procedurally generates random quests appropriate to be used in RPGs.

The quests analyzed by Doran and Parberry fit into 9 different motivations: Knowledge, Comfort, Reputation, Serenity, Protection, Conquest, Wealth, Ability, and Equipment. These motivations correspond to the most important concern an NPC has, and the goal of a quest created by the prototype system would be to resolve this concern. Has the game progresses, this motivation would change according to the NPC, with the player having a high influence on this change. The motivations have associated to them strategies that, through the use of actions, dictate how a quest should be followed in order to reach the goal.

When an NPC delivers a quest to the player, it chooses from 2-7 specific strategies determined by their current motivation. Each strategy is described by a verb-noun pair: 'kill pests', 'steal supplies', 'rescue NPC', etc. Then, each strategy has a sequence of 1 to 6 actions associated to them, which are the actions the player needs to tackle to complete a strategy. An action can be a simple atomic action, or it can be expressed recursively as a sequence of other actions. Therefore, we can represent the actions as an infinite set of trees, where the leaves are atomic actions that the player can carry out and the internal nodes represent tasks that need to be achieved along the way.

For example, a new quest starts with the choice of the NPC motivation:

$$< QUEST >::=< Knowledge >|< Comfort >|< Reputation >|< Serenity > \tag{2.1}$$

$$|< Protection >|< Conquest >|< Wealth >|< Ability >|< Equipment > \tag{2.2}$$

After choosing a motivation, it then chooses a strategy. The result can be described in this manner:

<Comfort> ::= <Obtain Luxuries>

<Obtain Luxuries> ::= <get> <goto> <give>

At the end of their work, Doran and Parberry, stated that, although they had built a functional prototype quest generator based on NPC motivations, further work would be needed to demonstrate its ability to generate quests that are as good as quest hand made by developers.

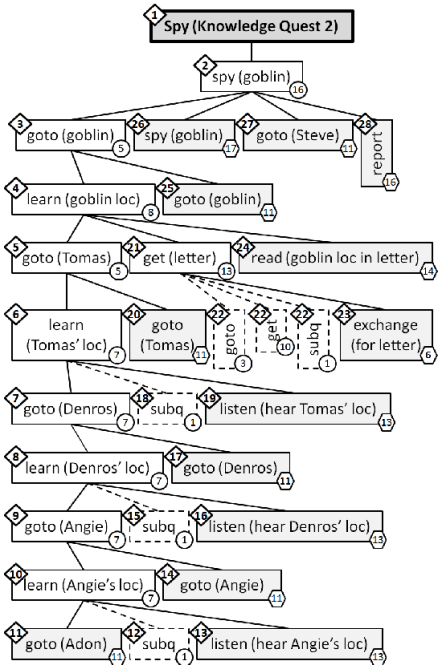

Figure 2.1: Example tree quest generated by the system [3]

2.2 Towards a Procedurally Generated Experience: A Structural Analysis of Quests

One example of further work on Doran and Parberry quest generator is the work done by António Machado [6]. Here, António Machado, made an analysis of the main quests from the award-winning

single-player RPG game 'The Witcher 3 - The Wild Hunt' [21] and implemented a grammar that extends what was previously presented by Doran and Parberry.

This grammar extension is, as claimed by the author, more expressive than the previous, which is something we agree on. This was accomplished by representing every main quest from 'The Witcher 3' in the same manner as Doran and Parberry had defined.

What was noticed while representing the quests was that, even though the NPC's that gave out the quests had the same or similar motivations as the ones defined by Doran and Parberry, the actions proved to have limitations making it impossible to fully describe the quests.

So a bigger analysis of the quests was conducted, where, while building the quest line trees as described by Doran and Parberry, the necessary changes were added. These changes were then attached to the previously built tables by Doran and Parberry.

These changes can be seen in the tables below.

#	Action	Pre-condition	Post-condition
1.	ε	None.	None.
2.	capture	Somebody is there.	They are your prisoner.
3.	damage	Somebody or something is there.	It is more damaged.
4.	defend	Somebody or something is there.	Attempts to damage it have failed.
5.	escort	Somebody is there.	They will now accompany you.
6.	**examine**	**Somebody or something is there.**	**You have information about it.**
7.	exchange	Somebody is there, they and you have something.	You have theirs, they have yours.
8.	experiment	Something is there.	Perhaps you have learned what it is for.
9.	explore	None.	Wander around at random.
10.	**follow**	**Somebody or something is there.**	**You will now accompany them.**
11.	**free**	**Somebody is there.**	**They are no longer prisoner.**
12.	gather	Something is there.	You have it.
13.	give	Somebody is there, you have something.	They have it, you don't.
14.	goto	You know where to go and how to get there.	You are there.
15.	kill	Somebody is there.	They are dead.
16.	listen	Somebody is there.	You have some of their information.
17.	read	Something is there.	You have information from it.
18.	repair	Something is there.	It is fixed, built or resolved.
19.	report	Somebody is there.	They have information you have.
20.	spy	Somebody or something is there.	You have information from it.
21.	stealth	Somebody is there.	Sneak up on them.
22.	take	Somebody is there, they have something.	You have it, they don't.
23.	use	Somebody or something is there.	It has affected characters or environment.
24.	**wait**	**None.**	**Wait for something to happen.**

Table 3. Atomic actions.

Figure 2.2: Table with all the actions and their conditions, added changes in bold [6]

Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> **<give>**
	Spy	<goto> spy **<report>**
	Interview NPC	<goto> listen **<report>**
	Use item on field	<get> <goto> use **<give>**
Comfort	Obtain luxuries	<get> **<give>**
	Kill pests	<goto> **<defeat>** **<report>**
Reputation	Obtain rare items	<get> **<give>**
	Kill enemies	<goto> **<defeat>** **<report>**
	Visit dangerous place	<goto> **<report>**
Serenity	Revenge, Justice	<goto> **<defeat>** **<report>**
	Capture Criminal	<goto> **<capture>** **<report>**
	Check on NPC (1)	<goto> listen **<report>**
	Check on NPC (2)	<goto> take **<give>**
	Recover lost/stolen item	<get> **<give>**
	Rescue NPC	<goto> **<rescue>** **<report>**
Protection	Attack threatening entities	<goto> **<defeat>** **<report>**
	Capture Criminal	<goto> **<capture>** **<report>**
	Treat or Repair (1)	<get> <goto> use **<report>**
	Treat or Repair (2)	<goto> repair **<report>**
	Create Diversion (1)	<get> <goto> use **<report>**
	Create Diversion (2)	<goto> damage **<report>**
	Assemble fortification	<goto> repair **<report>**
	Guard entity	<goto> defend **<report>**
	Recruit	<goto> listen **<report>**
Conquest	Attack enemy	<goto> **<defeat>** **<report>**
	Steal stuff	<goto> <steal> **<give>**
	Recruit	<goto> listen **<report>**
Wealth	Gather raw materials	<goto> <get> **<report>**
	Steal valuables for resale	<goto> <steal> **<give>**
	Make valuables for resale	<goto> repair **<give>**
Ability	Assemble tool for new skill	<goto> repair use
	Obtain training materials	<get> use
	Use existing tools	<goto> use
	Practive combat	<goto> damage
	Pratice skill	<goto> use
	Research skill (1)	<get> use **<report>**
	Research skill (2)	<get> experiment **<report>**
Equipment	Assemble	<goto> repair **<give>**
	Deliver supplies	<get> **<give>**
	Steal supplies	<steal> **<give>**
	Trade for supplies	<goto> exchange

Table 1. Strategies for each NPC's motivation.

Figure 2.3: Table with all the motivations and respective strategies, added changes in bold [6]

#	Rules	Explanation
0.	<Quest> ::= <Knowledge> | <Comfort> | <Reputation> | <Serenity> | <Protection> | <Conquest> | <Wealth> | <Ability> | <Equipment>	This is the root of a quest, which expands into one of the 9 motivations. Which will eventually be expanded into one of the strategies, specific to said motivation.
1.	<subquest> ::= ϵ	There is nothing to do.
2.	<subquest> ::= <Quest> **<goto>**	Go perform a quest an return.
3.	<goto> ::= ϵ	You are already there.
4.	**<goto> ::= goto**	Go to a known location.
5.	**<goto> ::= wait**	Wait at a location for someone or something.
6.	<goto> ::= explore	Just wander around and look.
7.	**<goto> ::= follow**	Follow somebody or something.
8.	**<goto> ::= stealth**	Sneak by someone.
9.	<goto> ::= <learn> **<goto>**	Find out where to go and go there.
10.	**<goto> ::= <prepare> <goto>**	Prepare before going somewhere.
11.	<learn> ::= ϵ	You already know it.
12.	<learn> ::= <goto> <subquest> listen	Go someplace, perform a subquest, get info from NPC.
13.	<learn> ::= <goto> <get> read	Go someplace, get something and read it.
14.	**<learn> ::= <goto> <subquest> examine**	Go someplace, perform a subquest, examine something.
15.	**<prepare> ::= <goto> <subquest>**	Go someplace and perform a subquest.
16.	<get> ::= ϵ	You already have it.
17.	<get> ::= <steal>	Steal it from somebody.
18.	<get> ::= <goto> gather	Go someplace and pick something up that's lying around.
19.	**<get> ::= <goto> take**	Go someplace and take something.
20.	<get> ::= <get> <goto> exchange	Get something, go to someonee and exchange.
21.	<get> ::= <get> <subquest>	Get something, perform a subquest.
22.	<steal> ::= <goto> stealth take	Go someplace, sneak on somebody and take something.
23.	<steal> ::= <goto> <defeat> take	Go someplace, defeat somebody and take something.
24.	<capture> ::= <goto> use capture	Go someplace, use something to capture somebody.
25.	<capture> ::= <goto> damage capture	Go someplace, damage to capture somebody.
26.	<capture> ::= <goto> capture	Go someplace and capture somebody.
27.	**<defeat> ::= <goto> damage**	Go someplace and damage somebody.
28.	**<defeat> ::= <goto> kill**	Go someplace and kill somebody.
29.	**<report> ::= ϵ**	There is nothing to report.
30.	**<report> ::= <goto> report**	Go someplace and report to somebody.
31.	**<give> ::= ϵ**	There is nothing to give.
32.	**<give> ::= <goto> give**	Go someplace and give something to somebody.
33.	**<rescue> ::= free**	Free somebody from imprisonment.
34.	**<rescue> ::= <defeat> free**	Defeat somebody and free somebody from imprisonment.
35.	**<rescue> ::= escort**	Escort somebody to someplace.
36.	**<rescue> ::= <defeat> escort**	Defeat somebody and escort a different somebody to someplace.

Figure 2.4: Table with all the rules and respective actions, added changes in bold [6]

2.3 Procedural Quest Generator for Conan Exiles

Given the task of implementing a procedural quest generator for the game, Conan Exiles [17] [5] that lacked any kind of questing system, António Machado, decided to implement an adaptation of the prototype generator model from Doran and Parberry [3].

To do so, he took the previous work he had already developed [6] and that was talked about in a

previous section.

Afterward, António implemented the procedural quest generator for the game Conan Exiles, which is a survival sandbox game that lacked any sort of questing system. The implemented system contained:

- An element to generate quests influenced by the NPC's motivations;
- An element that tracked the execution of the actions that were part of the quest;
- UI elements to represent the quests when the player and the NPC interacted.

The architecture implemented by António is a distributed architecture, meaning that to generate quests a generator was given out to each NPC capable of giving out quests. Each NPC then generates a quest that would satisfy a motivation strategy and is capable of communicating with other NPCs to generate sub-quests. The NPC motivations can be: randomly generated, predefined at game start, or they can be given by the game environment. To control the variation of action sequences, weights were added to each action sequence, based on the frequency that the sequences appeared.

Looking at the more technical side of the system. To generate a quest, the algorithm first needs to receive a grammar and a knowledge base. The grammar has a set of Nodes (that represent possible actions) a set of Rules and a set of Strategies. Each Node is divided in a set of Motivation nodes (these are the root of each quest) and a set of Actions. Each Action is further divided in a set of NonTerminal Nodes and a set of Terminal Nodes.

Both the Rules and the Strategies are represented as $head \rightarrow body$.

The knowledge base is composed of everything a NPC knows and everything that is relevant to it. It's defined in four different sets: Characters, Items, Enemies and Locations.

This system was tested with human players and gave out good results, since introducing the quest system did not harm the player's overall enjoyment of the game.

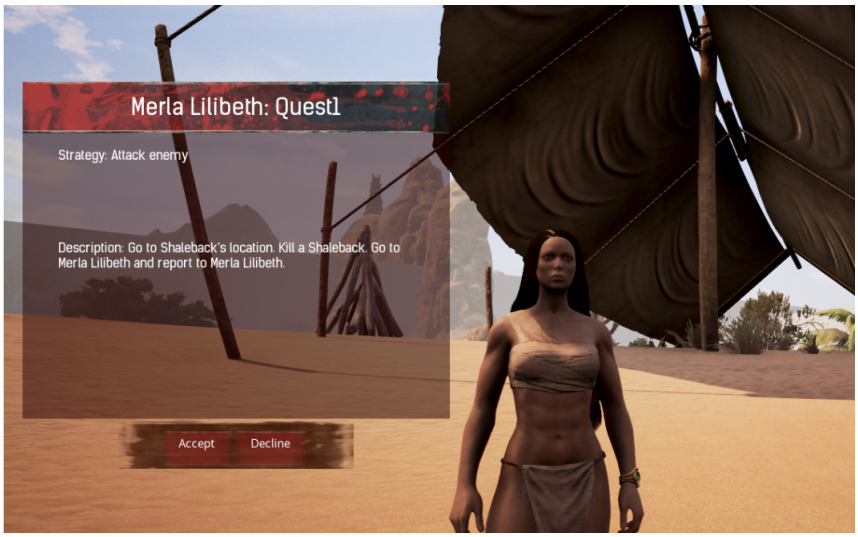

Figure 2.5: Conan Exiles Quest Example [5]

2.4 Quest Generation and Interactive Storytelling

Besides having to create an entertaining quest for the player, a procedural quest generation system should also be able to generate a quest that can be inserted in the narrative or story of the game in both a coherent and believable way. This is where interactive storytelling comes into play.

This sort of system can create credible narratives by simulating a story world with a cohesive storyline and convincingly directing or managing the characters in the story world.

Although this work doesn't require the full capabilities of an interactive storytelling system, it's still important to analyze the several works presented in this field since the engines they have can provide a suitable engine (after some adaptation) to this work, since procedural quest generation is part of interactive storytelling.

So, we're going to present three different works that differ in the type of architecture they use. This division was proposed by Broekema [2] and it helps us better understand the inner workings of the engines.

The architecture types are:

- Centralized Architecture: there is a central entity with complete control over the game world, choosing all actions for the characters and objects, but not for the player. Due to the player's unpredictability, this entity needs to make sure everyone and everything follows the story. To accomplish this, it intervenes in the game by slowing down some actions, making them happen later or blocking them, and by giving some hints to the player. This type of architecture is usually used in a deliberative, plot-driven system.
- Distributed Architecture: instead of having a central entity controlling the game world, this type

of architecture makes use of autonomous agents that each control their character in the story. This allows for more story variation and the prospect of player autonomy. It's implemented as a multiagent system and is generally used in either a simulation-based or an emergent system.

- Mixed Architecture: a combination of distributed and centralized drama management is used. In this kind of system, a weakened form of the other two systems is used to create a mixed architecture. A global plot line is followed by a planner, while there are also autonomous agents that represent story world characters. A central manager is used to handle plot events and direct the autonomous characters, so they don't digress from the story.

The works we are going to present for each architecture are, respectively: Mimesis, CONAN and IMPRATical.

2.4.1 Mimesis

The Mimesis architecture [12] provides the necessary components to incorporate narrative control into an already existing game environment. The architecture is structured to provide a storyline that the user will engage within current gaming contexts that would usually rely on fully scripted behaviors. Through the generation of plan-based behavior for both the characters and objects in the game environment, the architecture can provide the environment with instructions to implement the behaviors. This architecture was mainly implemented in the commercial game engine Unreal Tournament (UT) and was divided through multiple architectures. A part was integrated into the UT server, to give low-level access to the UT environment. The remaining part controlled the higher levels and created the narrative-based interactions. It was called the Mimesis Controller. Even though the architecture was distributed and ran along with different processors, its use was still limited to single-player games due to performance issues.

One use of this architecture [13] was an educational game where the players could move in a simulation of the Monterey Bay Aquarium. In this game, the players could learn from animated characters depicting tour guides, the labels at the habitats, and the animated animals themselves along with their behavior. The architecture generated plans to control the tour guides and the animals that were adapted to the environmental details that the player looked at and was focused on the condition of the simulated world, which included the location of the tour guide and the habitat inhabitants.

The Mimesis controller was divided in three different components:

1. The story planner. It activated when the game engine requested a plan, something that only happens when the game starts. It received the current state of the world and a set of actions and goals. Afterward, it created a plan containing all the actions for all the characters and objects of

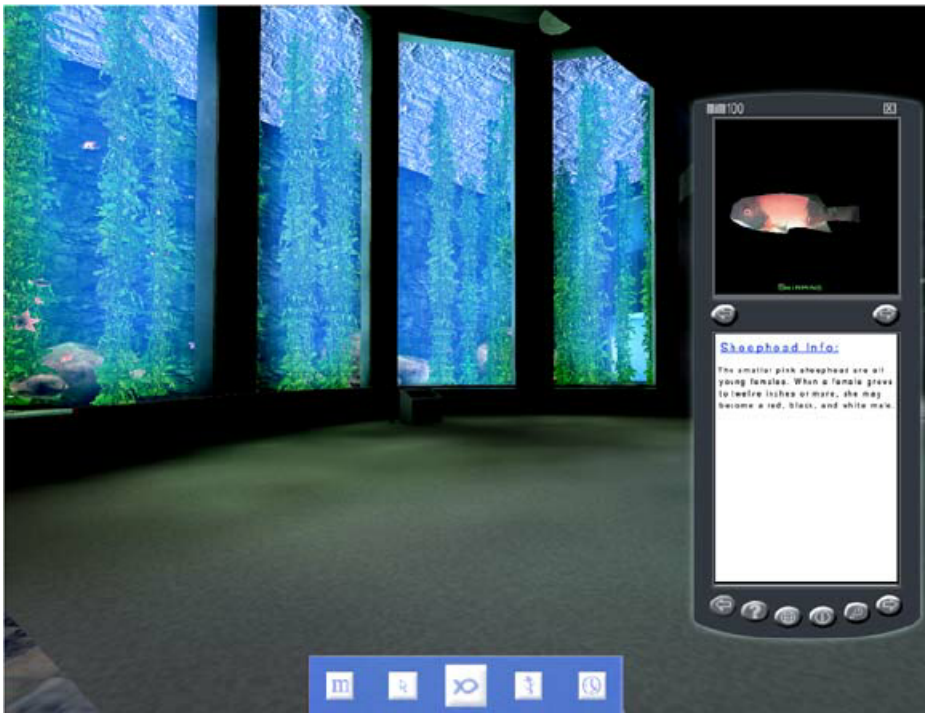

Figure 2.6: Mimesis Virtual Aquarium [9]

the world. These actions could be executed over time. The plan also has a sequence of actions that satisfy the goals. This plan was then passed to the second component.

2. The discourse planner. The job of this component was to build a plan to control the camera, the background music, and anything else media-related that the game engine had access to. This plan, along with the one from before, resulted in a narrative plan, that described all the actions that both the system and the user could execute. The narrative plan was then passed to the third component.

3. The execution manager. Having received the narrative plan, this component created a directed acyclic graph of all the actions and the time constraints associated with them. This graph was then used to choose the most appropriate action at the correct time. Besides this, the execution manager also had to communicate with the game engine as it executes the actions and processed the resulting information from the success of this execution.

Mimesis could handle the impact of the player in the world on two levels. The first level was used to handle execution conflicts. For example, an NPC could be trying to open a door, but the player could be blocking it. Something like this could happen because the player is capable of influencing the state of the world, and the player can influence it during the execution of a primitive action (like opening a door). The plans created by Mimesis can tell the difference between the preconditions that establish

the necessary state that the environment should be in effect immediately before the action is taken and the continuous preconditions that must be maintained after the action before the action is done. These continuous preconditions are constantly controlled in the component that is embedded with the game server to ensure that the machine will respond to a situation where the user interrupts a primitive operation. The second level was used to handle story conflicts. The player can cause an impact on the story plot constructed by Mimesis. These types of conflicts are controlled by intervening between the user commands and the execution of the game environment. They are controlled in two ways: either the player is prevented from completing its action, or its action is accommodated to the story. For example, when the player tries to open a door with a specific key, but he's not yet supposed to learn the information on the other side of the door, Mimesis can prevent the player from entering the room saying that the key doesn't match the lock, or instead it could accommodate to the user trying to open the door and move the information on the other side to a different room.

2.4.2 CONAN

The engine for the Creation Of Novel Adventure Narrative (or CONAN) proposed by Breault [1], is a distributed quest generation system. It is capable of creating quests that are cohesive and accomplish goals that are important to the preferences of the character. So, if we give an NPC the initial state and the list of all possible actions, it will be able to create plans to reach a goal in line with its preferences. The quests given to the players are these plans. While resolving these quests and having word-state altering events, the engine can produce increasingly more context-relevant quests as time goes on.

The initial state is composed of NPC's characters with predefined interests, locations, monsters and items, and rules governing the environment (for example: when a tree is cut down, there are fewer trees), a list of the possible actions with their prerequisites and their outcomes. All of this will be considered an object, and each one will be defined within the world state by a statement. These statements are a combination of predicates and the before-mentioned objects (for example, location(Castle) pertaining to the castle object, defining it as a location).

To exemplify an initial state, Breault presented two initial world states with two different sizes. The first world is a modified version of the Aladdin world (a small world) and a large world defined by himself, that contains much more objects and predicates. The Aladdin world had, for example, elements like:

- 'King Jafar who lives in a Castle'
- 'Aladdin, a knight who has cooperative attitude towards King Jafar'
- 'Jasmine Who also lives in The Castle'
- 'A Genie who is in the location 'Magic Lamp' and unable to get out'

- 'A Dragon who lives in the Mountain, is hostile to agents and guards the 'magic lamp"

The world defined by Breault had, for example, elements like:

- Agents: Baker, King Lumberjack, Blacksmith, Merchant, Guard, Daughter
- Locations: Castle, Village, Wheat field, Cave, Bakery, Forge, Forest, Shop
- Items: hammer, wheat, sword, magic spell book, etc.
- Monsters: troll, wolves, slimes, etc

The domain files contain the list of all possible actions, needed for the character to reach their goals. These actions are the same as the ones established by Doran and Parberry [3] in their structural analysis and are executed using PDDL. The actions are DoNothing, Capture, Damage, Defend, Escort, Exchange, Experiment, Explore, Gather, Give, GoTo, Kill, Listen, Read, Repair, Report, Spy, Stealth, Take and Use. And each action is composed of: a name, the parameters used, preconditions, and the effects of the action on the world after it has been performed. It's implemented in the system using PDDL syntax in the following way:

(:action move

:parameters (?p ?to ?from)

:precondition (and (location ?to)(location ?from)(player ?p) (at ?p ?from))

:effect (and (at ?p ?to)(not (at ?p ?from))(increase (total-cost) 2)))

The actions possess a weight that depends on the character's likes and dislikes. This is mapped specifically for each character in the form of an array, and the weights are taken into account when the plan is developed. So, actions with a lower cost will be more preferred by the character, while actions with a higher cost will be more undesirable. By having these weights for each character, a more coherent choice of actions is achieved, and the characters become more believable.

Character	Preference
Aladdin	[”+kill”,”-exchange”,”-use”,”+escort”]
Dragon	[”-damage”,”-take”,”-report”,”+escort”,”-defend”]
Genie	[”-kill”, ”-exchange”, ”-defend”, ”-read”]
Jasmine	[’+kill’, ’-spy’, ’-take’, ’-stealth’]
Jafar	[”-kill”, ”-spy”, ”-take”, ”-stealth”, move”]

Figure 2.7: Aladdin world Character Preference [1]

The goals of the characters are represented as a set of statements in a combination of predicates and objects. However, instead of having the object specified, it has variables limited to the type of

object. For example: '(has 'c' 'i'), where 'c' is a character and 'i' is information', '(explored 'l'), where 'l' is a location'.

CONAN uses two different algorithms to choose the goals of the characters to compare them. The first algorithm randomly selects goals for each NPC and uses them as a guideline to compare against the NPC's preferred goals. To do this, it chooses a random number of predicates from a list of predicates and afterwards, it goes through each one and assigns a random allowed item in place of the unknown object in the goal (for example the 'l' in '(explored 'l')'). The second algorithm will start by doing the same as the first one, by selecting a random number of goals, but after that, it ranks them according to the characters' preferences and keeps only the one with the best score, this is, the one that is more suitable for the character. It accomplishes this by getting the list of goals for each character and figuring out the plan to reach that goal. During this, the average cost of the actions in that plan is calculated, meaning that the plan with the average cost closer to 1 is the one more desired by the character. The algorithm only does this with 4 goals, because, although a higher number of goals would mean a higher chance of finding the ideal goal, it would also mean it would be more computationally taxing, thus slowing down the algorithm too much. Finally, the plan with the lowest cost will be given to the player that has a quest.

Breault defined some example quests using the world defined by him paired with the motivations defined by Doran and Parberry [3], seen in the table below. He also exemplified a plan to solve the need for wheat of an NPC, seen in the graph below.

Motivation	Example of quest
Knowledge	Find the location of the king's stolen treasure"
Comfort	Get rid of the wolves in the forest that are preventing the lumberjack from getting wood."
Reputation	Get granite and build a statue of me in the town square."
Serenity	Rescue the daughter of the baker that was taken by a troll."
Protection	Go kill the troll that has been traumatizing the village"
Conquest	Go kill my enemies."
Wealth	Go get some wood for the lumberjack to sell."
Ability	Find me the ancient spell book."
Equipment	Repair the lumberjacks' axe."

Figure 2.8: Example quests with their respective NPC motivation [1]

2.4.3 IMPRATical

A system proposed by Teutenberg [11], the Intentional Multi-agent Planning with Relevant Actions (or IMPRAtical), has characters that are in charge of arguing around their particular motives and searching for acceptable behavior about those same motives, while at the same time it also has a single planner Director that acts as an external guide and creates stories assuring the pertinent action limits. This way,

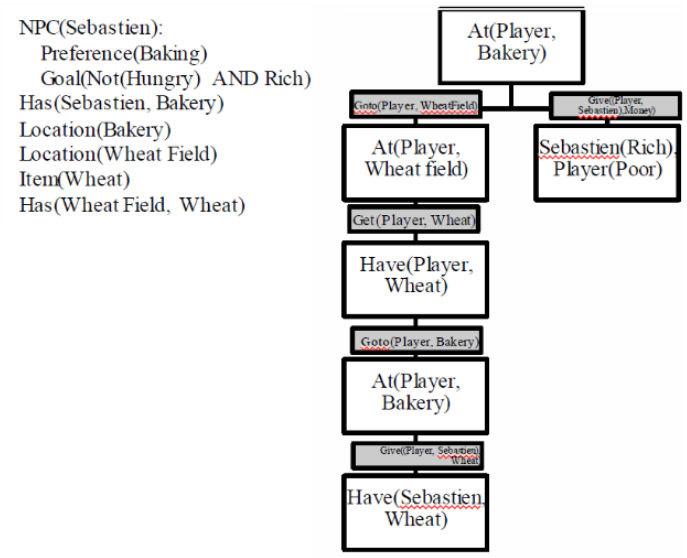

Figure 2.9: Example plan to solve the need for wheat [1]

according to Teutenberg, the system is capable of producing a more efficient and expressive plot when comparing it to systems of centralized architecture.

The system can model both purpose and cause of action in the narrative, conduct global heuristic searches, and assign intent to several planning agents. To conduct this method, the system requires for each narrative action to be causally related to the intent of the particular character and that the intent needs to be accomplished before the execution of the action, since the actions must be relevant according to the knowledge of the world story.

The characters have three different types of reasoning behind their intentions:

- If a character is isolated and can't fulfil its intention by itself, but there is another character with the same intent as the first one and in the same situation of unfulfillment, then both characters can cooperate so that they may achieve their intent;
- A character can also anticipate the actions of another character and use those same actions in its plan;
- A character can also command other characters for them to help it reach its goal. This can be repeated throughout the characters being issued commands, thus creating a chain of commands.

IMPRATical then creates a group of characters that reason about intent to give the Director all the relevant actions, after being provided with the current world state and the domain information. The system afterwards searches through several possible branches. This is done by constructing a relaxed planning graph with all the reachable actions and applying them to found facts by moving forward from

the current state. The solution is drawn out by searching back through the graph.

Bellow, we can better observe how the system works when trying to decide the best actions that lead to the goal. Teutenberg describes the image:

> Determining relevance for a single agent's actions. Solid lines indicate effect to fact or fact to precondition relations, dotted lines indicate a no-op.After the backwards step, of the two actions with their preconditions, met in the current state, one is found to also be relevant to the actor's intents.

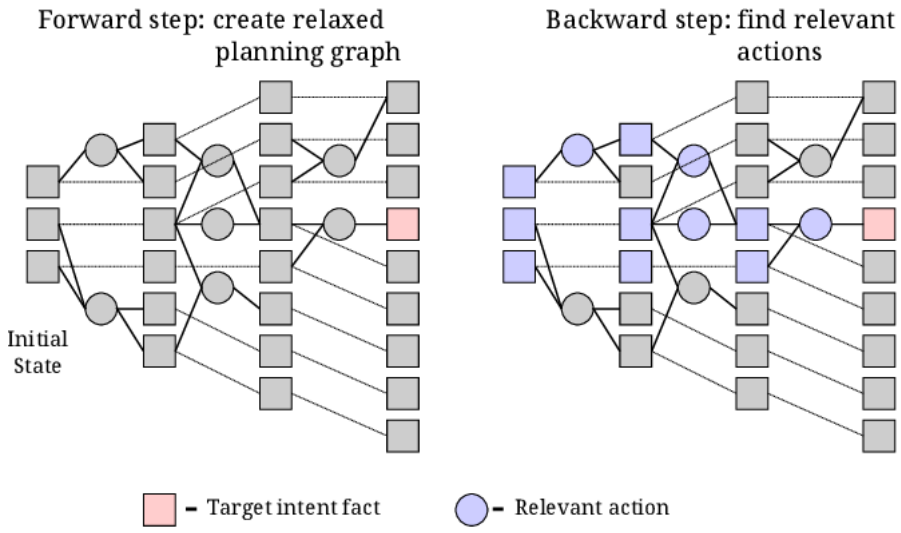

Figure 2.10: Graph of how the best actions are decided [11]

3

The Game - Mount & Blade II: Bannerlord

Contents

Figure 3.1: Mount & Blade II: Bannerlord Gameplay example

Bannerlord [15] is the fourth game of the Mount & Blade series, released and developed by TaleWorlds Entertainment. It's a prequel to the game Mount & Blade: Warband having its storyline take place 210 years before Warband with a setting inspired by the Migration Period [1]. It's an Open-World, Action, RPG (Role-playing game), Simulation, Strategy game (according to Steam, which is an online game platform) that was announced in 2012 and had its early access version released on March 30, 2020.

The game features both a single player and a multiplayer mode, but this works only focuses on single player:

- Single Player: In this mode, the players creates an avatar that you use to control your party containing troops. These troops can either be unnamed NPC's or named NPC's with their backstory, personality, and skills. The general gameplay is walking around the virtual world, fighting other party's that are led by NPCs. These NPC lead parties can range from a small group of looters to a giant army belonging to a faction, so the player needs to improve its power. To do so, he can recruit more troops for his party and/or level up with the XP (experience points) gained from each battle. The troops from the player's party also gain XP, levelling up and even getting promoted to a different rank. Besides fighting with other parties, the player can also experience the social side of the game, role-playing with the NPCs by talking to them and accepting quests from them, getting into the political system existent in the game (interacting with the several factions in the game world) or even getting married to an NPC and have offspring.

- Multiplayer: Works in the way of a Team death match between two teams, each consisting of 6

[1] '(...)period in the history of Europe (...) during which there was widespread migration of and invasions by peoples'

players. To win, your team needs to occupy specific strategy sites. Having the most sites means the other team's score will start decaying and when it reaches zero the battle ends and the team that still has score wins the battle. To participate in a battle, the game gives a player a party of his own to command, and the player can choose the type of party and equipment before each battle.

3.1 Bannerlord Quest System

Looking at the single-player quest system in more detail, the quests can be handed out by either an NPC living in a town, castle or village or an NPC roaming around the virtual world with its army. This is represented with a blue exclamation mark next to their avatar. When talking with these NPC's with quests they have extra dialog, in comparison with the other NPC's, so they can explain the quest to the player. After accepting a quest it appears on the quest management screen with the title of the quest, a small description, our current progress and step, and how many days there are left to complete it. There are currently (29 October 2021) over 25 different quests, all with a different goal and a different way to achieve said goal. In the examples below, we accepted a quest from an NPC that asked the player to train some soldiers.

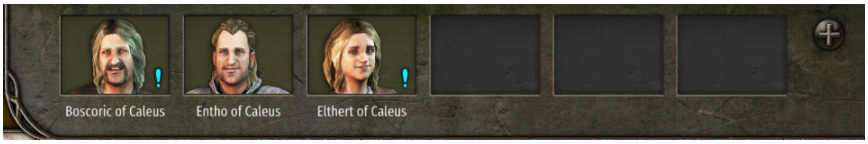

Figure 3.2: NPC's with a quest

Figure 3.3: Quest Management Screen

Taking a general look at the inner workings of Bannerlord: there is a central class called **Campaign** that initializes everything and adds the behaviors responsible for the detection and creation of the quests. The initialized classes **QuestManager** and **IssueManager** are responsible for keeping a list of all current quests and issues, each one being of the type **QuestBase** and **IssueBase**, respectively. The way a quest is implemented is: first an event listener is added, detecting when an issue is requested and creating a new object of the type **IssueArgs**. If the issue is requested, then it checks if a certain number of conditions are met and if they are, the **PotentialIssueData** of the **IssueArgs** is set and a new object is created extending **IssueBase**. The new issue is then defined through several either general functions or issues specific functions and a new object is created extending **QuestBase**. Of the several added behaviors, one is the class **IssuesCampaignBehavior**, which is responsible for the creation and allocation of new issues (and therefore the creation of new quests). It creates a controlled random number of issues for the heroes (the name given to the NPC's capable of handing out quests, represented by the class **Hero**) existing in the settlements (the name given to the villages and towns present in the game map) and for the existing clans. It does this at the start of a new game and afterwards once every day.

To handle the character interactions, a class called **DialogFlow** is used. There the dialogues are given to the characters, either taken from the XML files or directly from the code. It is also through this that the player has the liberty to decide the outcome of a quest (for example: Given the task to deliver some products, upon arriving at the NPC to deliver the products, the player can instead choose to keep them to himself), having all the different paths and consequences delegated through conditions.

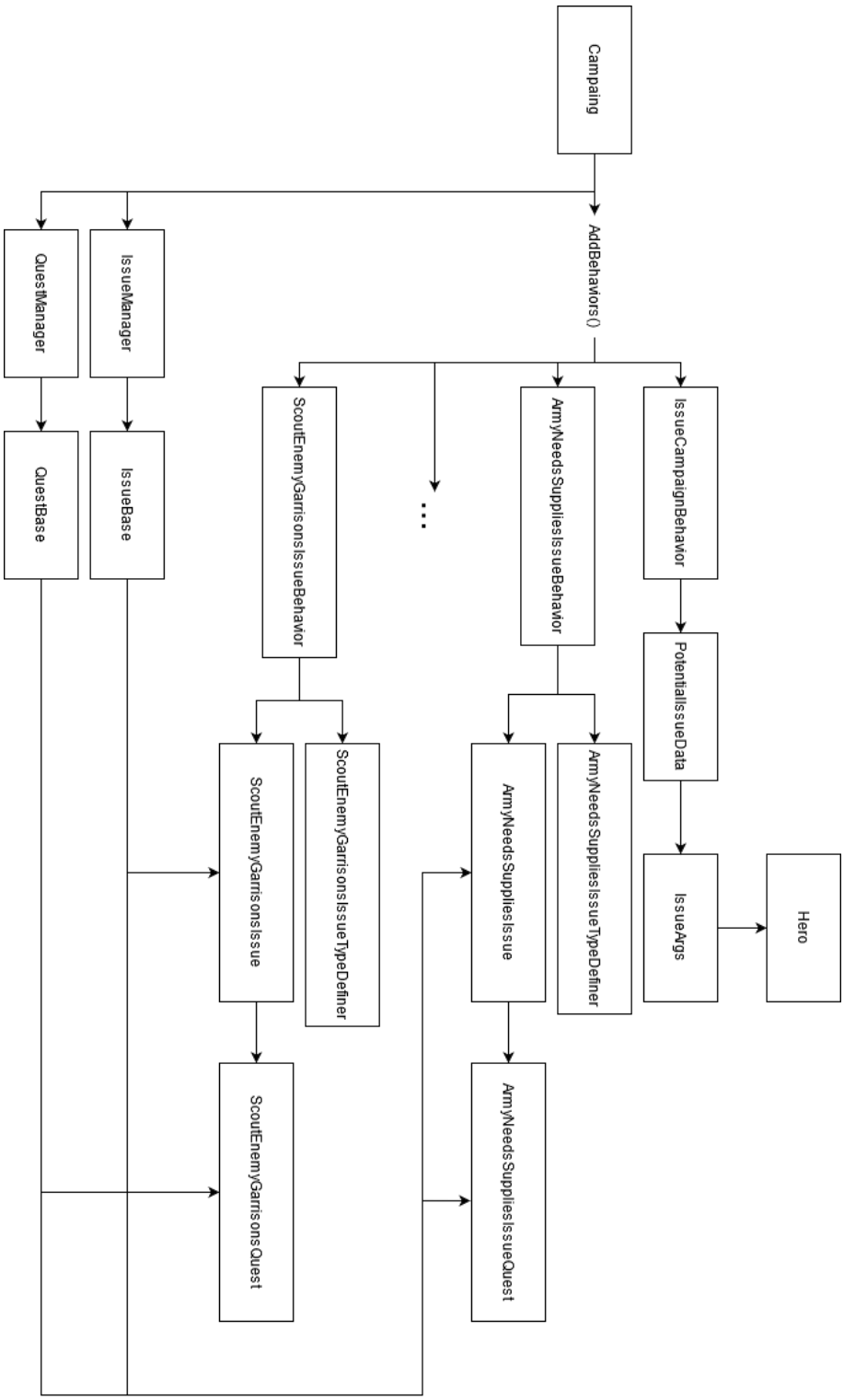

Figure 3.4: Class Diagram

3.2 Modding in Bannelord

To implement the questing system, we made use of modding [2]. Bannerlord is relatively easy to start modding and there are plenty of tutorials on how to do it. To start, we first need to create the Mod folder in the Module's directory in the game files. Afterwards, all we need to do is set up a project in the environment of our choice (for example Visual Studio) and compile the code to the mod folder (more information can be consulted in the Bannerlord documentation page [16]). To distribute our mod, we only need to share the Mod folder with other players. One of the main challenges when designing the mod came from the fact that, since it's a recent game (still in early access at the time of writing this thesis), there is no available documentation of the game, and we had to make use of our IDE (integrated development environment) and trial and error to know what methods we had available to implement the quest system.

3.2.1 Harmony

We made use of the Harmony Library [18]. This is a library used for patching, replacing and decorating .NET methods during runtime. Since we didn't have access to the source code of the game and altering the game's dll's could cause legal issues, we had to use Harmony when wanting to affect Bannerlord methods. Harmony allows us to execute our code before or after the original method through the use of either a *Prefix* or a *Postfix*.

Bannerlord didn't trigger all the events we needed or had them implemented, so we had to add some *Postfix* methods to cover for that. These methods run after the original method. For example: Bannerlord has a smiting mechanic in the game where there are three possible actions: create a new weapon, dismantle a weapon or refine a new material (i.e. make Iron and Steel). There are events that get triggered when a new weapon is created and when it's dismantled, but there isn't any event for the refinement of a new material, so we had to use a *Postfix* method to run after the original method that handles the refinement so that we could detect when the player did this action.

[2] 'an alteration by players or fans of a video game that changes one or more aspects of a video game, such as how it looks or behaves' [10]

4

Bannerlord Quest Generator - BQG

Contents

The goal is to implement a procedural quest generation system for Bannerlord, a game that already has a questing system. With this new system, we aim to improve Bannerlord by providing quests that are less repetitive and have a different more varied structure, while at the same time reducing the effort needed to make every single individual quest. With the new system, we can produce an almost infinite number of different quests. To accomplish this, three modules have been developed: the Generator module, the Instantiator module and the Monitor module.

4.1 A General Overview

The Generator module is responsible for building a behavior tree-like quest with placeholder names for the world objects (such as ”place1”, ”npc1”, ”item1”) similarly to what was proposed by Ian Parberry and Jonathon Doran with the modifications made by António Machado. The generated quests present a lot of variation in both quest type and structure. The Motivations selected for the quest are chosen at random, this allows the next module to have plenty of different quests with different motivations to assigned to the correct NPC's.

The Instantiator module, as said before, is responsible for taking the quests generated by the previous module and assigning them to NPC's with the same Motivation. This is module is also responsible for assigning the Motivations to the NPC's, that are randomly assigned depending on the type of NPC (i.e.: if it's a noble NPC it can have Reputation type quests, but a farmer NPC can't). After assigning a quest to a NPC, it then takes the placeholder world objects of each action and replaces them with world objects that currently exist, this is, it replaces ”place1” with a settlement, ”npc1” with an NPC and ”item1” with an item.

Finally, the Monitor module starts when the player accepts the quest. Its job is to update the quest logs that appear on the quest screen by monitoring the player progress throughout the quest. This is done by listening to the several events the game launches every time certain actions happen. It also assigns dialogue to the NPC's so that the player can progress in the quest.

Strategies, Rules and Actions used can be seen on the tables below.

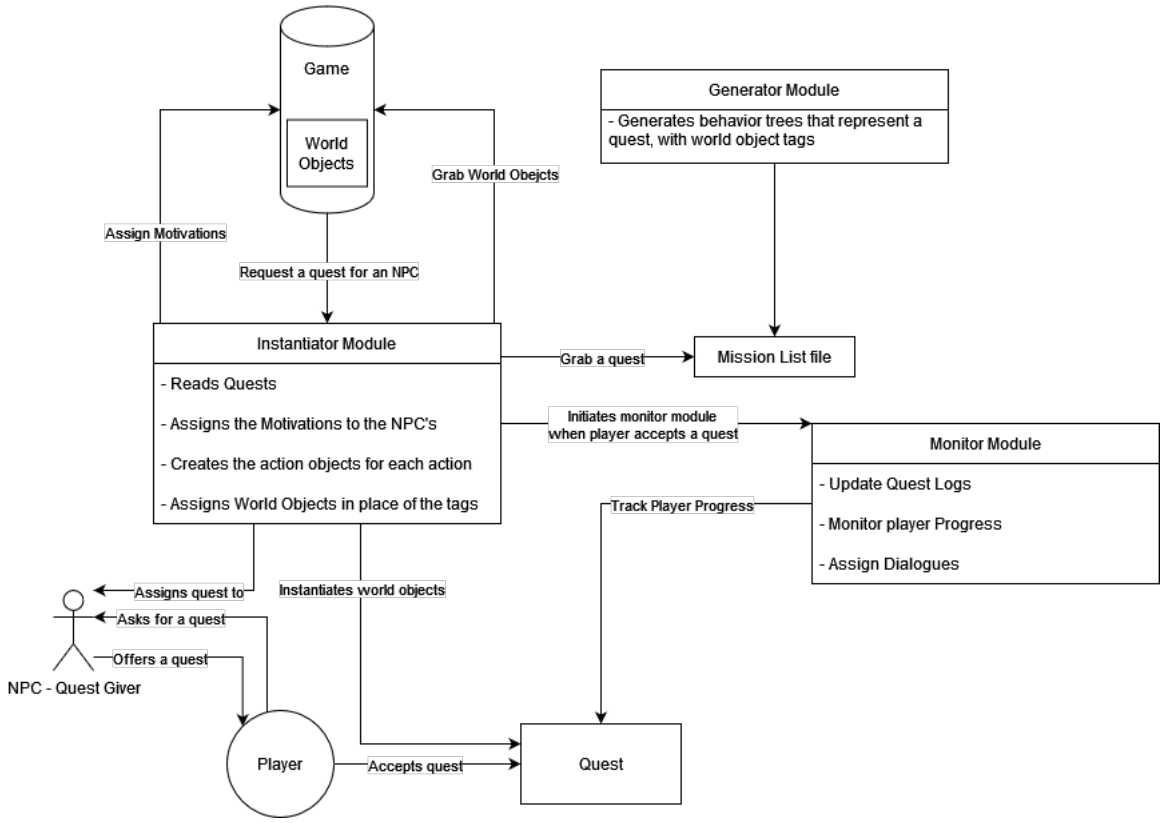

Figure 4.1: Diagram of the implemented system

#	Rules	Explanation
0.	<Quest> ::= <Knowledge> | <Comfort> | <Reputation> | <Serenity> | <Protection> | <Conquest> | <Wealth> | <Ability> | <Equipment>	This is the root of a quest, which expands into one of the 9 motivations. Which will eventually be expanded into one of the strategies, specific to said motivation.
1.	<subquest> ::= ϵ	There is nothing to do.
2.	<subquest> ::= <Quest> <goto>	Go perform a quest and return.
3.	<goto> ::= ϵ	You are already there.
4.	<goto> ::= goto	Go to a known location.
5.	<goto> ::= explore	Just wander around and look.
6.	<goto> ::= <learn> <goto>	Find out where to go and go there.
7.	<goto> ::= <prepare> <goto>	Prepare before going somewhere.
8.	<learn> ::= ϵ	You already know it.
9.	<learn> ::= <goto> <subquest> listen	Go someplace, perform a subquest, get info from NPC.
10.	<prepare> ::= <goto> <subquest>	Go someplace and perform a subquest.
11.	<get> ::= ϵ	You already have it.
12.	<get> ::= <steal>	Steal it from somebody.
13.	<get> ::= <goto> gather	Go someplace and pick something up that's lying around.
15.	<get> ::= <get> <goto> exchange	Get something, go to someonee and exchange.
16.	<get> ::= <get> <subquest>	Get something, perform a subquest.
17.	<steal> ::= <goto> <defeat> take	Go someplace, defeat somebody and take something.
18.	<capture> ::= <goto> damage capture	Go someplace, damage to capture somebody.
19.	<capture> ::= <goto> capture	Go someplace and capture somebody.
20.	<defeat> ::= <goto> damage	Go someplace and damage somebody.
21.	<defeat> ::= <goto> kill	Go someplace and kill somebody.
22.	<report> ::= ϵ	There is nothing to report.
23.	<report> ::= <goto> report	Go someplace and report to somebody.
24.	<give> ::= ϵ	There is nothing to give.
25.	<give> ::= <goto> give	Go someplace and give something to somebody.
26.	<rescue> ::= free	Free somebody from imprisonment.
27.	<rescue> ::= <defeat> free	Defeat somebody and free somebody from imprisonment.

Figure 4.2: Table with all the rules and respective actions

Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> <give>
	Interview NPC	<goto> listen <report>
Comfort	Obtain luxuries	<get> <give>
	Kill pests	<goto> <defeat> <report>
Reputation	Obtain rare items	<get> <give>
	Kill enemies	<goto> <defeat> <report>
	Visit dangerous place	<goto> <report>
Serenity	Revenge, Justice	<goto> <defeat> <report>
	Capture Criminal	<goto> <capture> <report>
	Check on NPC (1)	<goto> listen <report>
	Recover lost/stolen item	<get> <give>
	Rescue NPC	<goto> <rescue> <report>
Protection	Attack threatening entities	<goto> <defeat> <report>
	Capture Criminal	<goto> <capture> <report>
	Create Diversion (2)	<goto> damage <report>
	Recruit	<goto> listen <report>
Conquest	Attack enemy	<goto> <defeat> <report>
	Steal stuff	<goto> <steal> <give>
	Recruit	<goto> listen <report>
Wealth	Gather raw materials	<goto> <get> <report>
	Steal valuables for resale	<goto> <steal> <give>
Ability	Practice combat	<goto> damage
	Practice skill	<goto> us
Equipment	Deliver supplies	<get> <give>
	Steal supplies	<steal> <give>
	Trade for supplies	<get> <goto> exchange

Figure 4.3: Table with all the motivations and respective strategies

#	Action	Pre-condition	Post-condition
1.	ϵ	None.	None.
2.	capture	Somebody is there.	They are your prisoner.
3.	damage	Somebody or something is there.	It is more damaged.
4.	exchange	Somebody is there, they and you have something	You have theirs, they have yours.
5.	explore	None.	Wander around at random.
6.	free	Somebody is there.	They are no longer prisoner.
7.	gather	Something is there.	You have it.
8.	give	Somebody is there, you have something.	They have it, you don't.
9.	goto	You know where to go and how to get there.	You are there.
10.	kill	Somebody is there.	They are dead.
11.	listen	Somebody is there.	You have some of their information.
12.	report	Somebody is there.	They have information you have.
13.	take	Somebody is there, they have something.	You have it, they don't.
14.	use	Somebody or something is there.	It has affected them.

Figure 4.4: Table with all the actions and their conditions

We're now going to take an in-depth look at each module and each terminal action.

4.2 The Generator module

This generator is an extension of what the generators made by António Machado and Ian Parberry and Jonathon Doran. Besides the grammar based generation of the previous system, we also create a behavior tree during the generation process of a quest, implemented an option to give the player alternative ways to complete the quests, adapted the grammar to fit Bannerlord and . It was written using C#, which is the language used by the Bannerlord engine.

These behavior trees allow us to more easily traverse the quest tree when needed. They also allowed us to implement a new feature, an alternative way to complete quests. This is, when an NPC gives a quest to the player, there is a chance that besides having the regular path to complete the quest, the player can receive an alternative path that also leads to the completion of the quest (e.g.: If the regular way to complete a give quest is “Visit settlement X”, the alternative way can be “Visit settlement Y”). The player can choose either way to follow, and can even proceed to play out both ways at the same time and choose later which one to complete. The different node types of the behavior tree allow us to do these actions more efficiently and easily.

The generator is similar to what António Machado implemented. The Initial Input that is composed of a grammar and a knowledge base has a similar grammar, however, the knowledge base works differently.

For the grammar, we first had to see what actions were possible, which ones made sense to implement in a game like Bannerlord and what could actually be accomplished in the time we had. After that analysis, we reached the conclusion that: the same 9 motivations could be kept, however not all the Strategies could be used, and neither could all the Rules. This was because we cut down the number of possible terminal actions from 24 to 13. The actions that we decided to implement were: capture, damage, exchange, explore, free, gather, give, go to, kill, listen, report, take and use. The other actions weren’t added mostly due to limitations from Bannerlord itself and partly because some of them were already present in a lot of the quests from the base game.

This implementation of 13 actions, together with subquests and alternative ways to complete the quest, present an improvement to the system António Machado implemented in Conan Exiles by developing quest structures much more complex and varied. This module can generate quests for any other game, as long as the game can translate them into quests fitting to it.

The grammar contains a set of Nodes that are divided into Motivation Nodes (9 of them) and Action nodes that further divide into NonTerminal Nodes (12 of them) and Terminal nodes (13 of them) that represent the actions the player makes in-game, a set of Rules and a set of Strategies. Every Action also contains a set of parameters that make a reference to the game objects.

The Rules are represented in the manner of *head* $\rightarrow$ *body*, where the *head* is always a NonTerminal node and the *body* is a sequence of Action nodes. Every time a rule is selected, the body is added to the quest structure.

The Strategies are also represented as *head* $\rightarrow$ *body*, however the *head* must be a Motivation node instead. The *body* is still a sequence of Action nodes.

The knowledge base is where our implementation differs the most. Instead of having direct access to all the game objects, the generator instead goes through four simple lists of tags:

- Character list: npc1, npc2, npc3, npc4, npc5, etc.
- Items list: item1, item2, item3, item4, item5, etc.
- Enemies list: enemy1, enemy2, enemy3, enemy4, enemy5, etc.
- Locations list: place1, place2, place3, place4, place5, etc.

So, when filling in the parameters, the names above are used instead of directly using the settlements, NPC's, items and enemies names. In the end, all the actions with the parameters filled in order of a quest look similar to this example: *Protection,goto place3,quest npc2,explore place5,gather item2,quest npc4,goto place7,kill enemy2,explore place9,report npc5,goto place6,explore place10,give npc3,give item1,goto place4,explore place11,capture enemy1,goto place12,report npc1*. Having the same name means it will refer to the same game object.

Parallel to this, the generator is also creating the previously mentioned behavior tree [8]. It uses 4 node types. A Selector (represented by an "?") and Sequence (represented by an "$\rightarrow$") nodes from a usual behavior tree and two custom ones, Motivation and Action node, where the Motivation node usually represents the head of the tree (except when there is an alternative way to complete the quest) and the Action nodes are the leaves. The Sequence node is used to guarantee that each Action is executed in the correct order. The Selector node is used for the event where there is an alternative way to complete a quest, its children will be two different quests and only one of them needs to be completed. The Motivation node will act similarly to a Sequence node. The Action node represents the action the player needs to perform to complete the quest. It acts parallel to the usual generation because the usual generation is the one responsible to allocate the right game objects substitutes to the parameters.

4.2.1 Quest Generator Algorithm

Will now take an in depth look at how a quest is created.

4.2.1.A CreateQuest algorithm

The first step is to call the CreateQuest algorithm. This starts the generation process by first deciding if there's going to be an alternative way to complete the quest or not. Depending on the decision it either creates one or two instances of a quest and assigns root nodes to them, these root nodes are the motivation the NPC needs to have this quest. This motivation is randomly generated, so that, when

the Instantiator module needs a quest for an NPC with a certain Motivation, if there is a large number of pre-generated quests, it's very likely that there is one available for the NPC. After the assignments, the expansion phase begins. Also, depending on the decision, either one or two behavior trees are created with a motivation node as the head. When the two trees are created, they are added as children of a Selector node, so that later only one of the trees is completed when completing the quest.

Algorithm 1 Create Quest

```
1: function CreateQuest(Grammar, KB)
2:     if alternative then
3:         BehaviorTreeSelector ← NEWOBJECT<BEHAVIORTREE_SELECTOR>();
4:         BehaviorTree1Motivation ← NEWOBJECT<BEHAVIORTREE_MOTIVATION>();
5:         BehaviorTree2Motivation ← NEWOBJECT<BEHAVIORTREE_MOTIVATION>();
6:         BehaviorTreeSelector.Children.Add(BehaviorTree1Motivation)
7:         BehaviorTreeSelector.Children.Add(BehaviorTree2Motivation)
8:         quest1 ← NEWOBJECT<QUEST>();
9:         quest1.Root ← QuestGiver.Motivation;
10:        Expand(quest1.Root, BehaviorTree1Motivation);
11:        quest2 ← NEWOBJECT<QUEST>();
12:        quest2.Root ← QuestGiver.Motivation;
13:        Expand(quest2.Root, BehaviorTree2Motivation);
14:    else
15:        BehaviorTreeMotivation ← NEWOBJECT<BEHAVIORTREE_MOTIVATION>();
16:        quest ← NEWOBJECT<QUEST>();
17:        quest.Root ← QuestGiver.Motivation;
18:        Expand(quest.Root, BehaviorTreeMotivation);
19:    end if
20: end function
```

4.2.1.B Expand algorithm

Afterwards, we call the recursive algorithm *Expand(node, BT)*. It is called every time a node is generated and added to the quest structure, with a different behavior depending on the type of node that is selected:

- **NonTerminal** - If a NonTerminal node is selected, its children will be selected from the Rules Table from Chapter 3. The rules are chosen stochastically to assure a certain variety exists in the generated quests. To accomplish this, weights were attributed to each rule and a rule is chosen using the *WeightedChoice* algorithm. Each rule is given a weight ranging from [0,1]. This list of rules is then added to the quest structure and to the behavior tree as Sequence nodes.

- **Terminal** - If a Terminal node is selected, it means we've reached a leaf of the tree and this node can not be expanded. This node represents the actual action the player has to perform, and it's added to a list belonging to the quest in the order the player has to complete them. It's also added to the behavior tree as an Action node.

- **Motivation** - If a Motivation node is selected, its children will be selected from the Strategies Table

from Chapter 3. The selected Strategy is chosen randomly.

When a rule/strategy is selected, a node is generated for every action (both Terminal and NonTerminal) in it. The action parameters are then associated with world object tags. To create the subquests we simply generate a new Motivation node and place it as the child of an action like node with the name of "quest" (this represents a subquest so that the translator can know there is one) and add a Motivation node to the behavior tree too. Then when the *Expand* algorithm reaches this node, it develops a new quest attached to the original one.

Algorithm 2 ExpandNode

```
    function EXPANDNODE(node, BT)
 2:     if node ∈ T then
          quest.Steps.Add(node);
        else
 4:         if node ∈ NonTerminal then
                if node is Subquest then
 6:                 node.Children.Add(subquestMotivation);
                    BT.newNode(Motivation);
 8:             else
                    rules ← Rules(node);
10:                 index = WEIGHTEDCHOICE(weights);
                    sequence = R(node, index);
12:                 weights = UPDATEWEIGHTS(rules,weights,index,depth);
                end if
14:         else if node ∈ Motivation then
                strategies ← Strategies(node);
16:             index = SELECTSTRATEGY(strategies);
                sequence = Strategies(node, index);
18:         end if
            for all action ∈ sequence do
20:             node.Children.Add(action);
                BT.newNode(Action);
22:         end for
            BINDPARAMETERS(node);
24:         for all child ∈ node.Children AND BT.Children do
                EXPAND(child, BT.child)
26:         end for
        end if
28: end function
```

The *Expand* algorithm calls another three methods: *WeightedChoice*,*UpdateWeights* and *BindParameters*. Let's first take a look at the *WeightedChoice* algorithm.

4.2.1.C WeightedChoice algorithm

Each rule has weights associated with its sequence of actions so that we can have control over the variety of actions sequences. The purpose of the *WeightedChoice* algorithm is to generate an index based on a range of weights so that we can then use that index to select the sequence of actions from the rules table. To obtain that index, a random number between 0 and 1 is first generated, and then it's compared to a set of ranges. This set of ranges is based on the weights shown in the table below, and the sum of all the weights of a sequence of actions must always be 1. These values were reached through trial and error, and are adapted for Bannerlord. The values were determined empirically and can later be revised. The value for the "explore" rule, in particular, was chosen because the generator kept choosing that route that lead to having quests with a lot of unnecessary and inexplicable explore actions. The null actions are at a very low value so that the chance of being chosen at the start of a quest is very low, but it increases throughout the quest generation. The other were derived from what António Machado had in his work.

#	Rules	Weights
1.	<subquest> ::= ϵ	0.3
2.	<subquest> ::= <Quest> <goto>	0.7
3.	<goto> ::= ϵ	0.2
4.	<goto> ::= goto	0.34
5.	<goto> ::= explore	0.01
6.	<goto> ::= <learn> <goto>	0.2
7.	<goto> ::= <prepare> <goto>	0.25
8.	<learn> ::= ϵ	0.1
9.	<learn> ::= <goto> <subquest> listen	0.9
10.	<prepare> ::= <goto> <subquest>	1.0
11.	<get> ::= ϵ	0.1
12.	<get> ::= <steal>	0.35
13.	<get> ::= <goto> gather	0.25
15.	<get> ::= <get> <goto> exchange	0.15
16.	<get> ::= <get> <subquest>	0.15
17.	<steal> ::= <goto> <defeat> take	1.0
18.	<capture> ::= <goto> damage capture	0.45
19.	<capture> ::= <goto> capture	0.55
20.	<defeat> ::= <goto> damage	0.3
21.	<defeat> ::= <goto> kill	0.7
22.	<report> ::= ϵ	0.3
23.	<report> ::= <goto> report	0.7
24.	<give> ::= ϵ	0.1
25.	<give> ::= <goto> give	0.9
26.	<rescue> ::= free	0.3
27.	<rescue> ::= <defeat> free	0.7

Figure 4.5: Set of weights used by the generator

Algorithm 3 Weighted Choice

```
    function WeightedChoice(weights)
        rand = RANDOM(0,1);
 3:     limit1 = 0;
        limit2 = 0 ;
        for all weight ∈ weights do
 6:         limit2 = limit2 + weight;
            if limit1 =< rand <= limit2 then
                return weight.index;
 9:         else
                limit1 = limit2;
            end if
12:     end for
        rand1 = RANDOM(weights.Count);
        return rand1;
15: end function
```

4.2.1.D UpdateWeights algorithm

We'll now take a look at the *UpdateWeights* algorithm.

After selecting the index to select the action sequence, the set of weights of the selected sequence will be updated. To do so, the algorithm receives the sub-set of rules, the sub-set of weights, the index chosen and the current depth of the tree.

First, the algorithm reduces the weight of the selected sequence of actions. It accomplishes this by dividing the weight by the value of a converge function. This function can be defined in different ways that allow to control how big a quest can be. For example: $\sqrt{depth}$ for bigger quests, $2^{depth-1}$ for medium quests and $depth^2$ for smaller quests. Meaning that this is the function that ends up controlling the size of a quest. For our system, the converge function was set at $depth^2$ because we found that generated the right size quests for Bannerlord, this is, quests that took an appropriate amount of time to complete.

With the weight of the chosen sequence update, now the sum of all the weights doesn't equal 1. Therefore, the rest of the weights needs to be updated so that the sum is equal to 1 again. The missing portion is divided between all the rules in the sub-set that contain Terminal actions.

It's important to update the weights so that the quest doesn't grow out of control. When the algorithm reaches a certain depth, we want it to avoid choosing a sequence of actions that contain NonTerminals and favors choosing one that either contains Terminal actions or reaches the $\in$ (none) action.

Algorithm 4 Update Weights

```
function UpdateWeights(rules, weights, index, depth)
    weights[index] = weights[index]/CONVERGENCE(depth);
    portion = (1 - weights[index])/NumberOfUpdates;
    for all weight ∈ weights do
        if rules[weights.index.CONTAINSTERMINALS] then
            weight += portion;
        end if
    end for
    return weights;
end function
```

4.2.1.E BindParameters algorithm

Finally, we're going to see the *BindParameters* algorithm.

When the process of creating nodes for each action from the sequence of actions is completed, we selected we have to bind each parameter from the actions to a world object tag before adding them to the quest structure and call the *Expand* algorithm for each of them. To do this, we use the *BindParameters* algorithm. This algorithm is necessary to ensure there is coherent logic between sequential actions. The parameters associated with each action can have three different restrictions types (except for the actions *steal*, *get* and *give* that have specific behaviors): parent node restrictions, sibling node restrictions or not restricted at all. For the last restriction type, we give the parameter a new object tag that hasn't been referenced yet. The encoding for the restrictions is as follows: flag 0 for parent restriction, 1 for no restriction (also known as new case) and 2 for sibling restriction. As an example, we can take the rule *<capture> → <goto> damage capture*. After passing through the algorithm, the rule would end up as *<capture>(enemy1) → <goto>(place1) damage(place1, enemy1) capture(enemy1)*.

Each parameter needs to have the following properties predefined at the start of the game:

- Parameter type (NPC; item, place, enemy);
- Restriction flag (0,1,2, or -2 for the *give*);
- Sibling reference if the flag is equal to 2;

At the start, the algorithm will deal with the first two exceptions. These exceptions are for the NonTerminal actions *steal* and *get*. If either the *steal* or the *get* action have as a parent another *get* NonTerminal action, then their original restriction flag value of 1 changes to 0. This is because these actions need to have the restriction flag 1 when they have no parents, and the restriction flag 0 when they have a parent of the action type *get* to make sure there is coherent logic between the sequential actions. Afterwards, the algorithm checks the value of the restriction flag.

If the flag has the value of 1, then a reference to a new world object tag is created. If the flag has a value of 0, then the algorithm checks if the parent action parameter type is the same as the current

action type, and if it is, then it just binds the parent target to the current action. If the flag has a different value from 0 and 1, then the algorithm first sees the last exception. This is, if it's dealing with the Terminal action *give*, if it is, then it changes the sibling reference of the action to the index of the node prior to the current action. This is done to make sure there is coherent logic between sequential actions since the action *give* parameter always gets its world object from the NonTerminal action *get* that always comes prior to it, however, it's not always in the same index. After this exception the algorithm checks if the sibling (that has the stored index) action parameter type is the same as the current action type, and if it is then it just binds the sibling target to the current action.

The parameter values used can be seen in the table below the algorithm.

Algorithm 5 Bind Parameters

```
function BindParameters(node)
    for all child ∈ Children do
        if child.action == "steal" AND child.parent.action == "get" then   ▷ Steal action parameter exception
            child.parameter.Restriction = 0;
        end if
        if child.action == "get" AND child.parent.action == "get" then      ▷ Get action parameter exception
            child.parameter.Restriction = 0;
        end if
        for all param1 ∈ child.Parameters do
            if param1.Restriction == 0 then                                 ▷ Parent Case
                for all param2 ∈ node.Parameters do
                    if param1.Type == param2.Type then
                        param1.Target ← param2.Target;
                    end if
                end for
            else if parameters.Restriction == 1 then                        ▷ New Case
                param1.Target←NEWPARAMETER(param1.Type,K);
            else                                                            ▷ Sibling Case
                if param1.SiblingIndex == -2 then                           ▷ Give action parameter exception
                    param1.SiblingIndex = node.Chilren.IndexOf(child) - 1;
                end if
                for all param2∈node.Children[param1.SiblingIndex].Parameters do
                    if param1.Type == param2.Type then
                        param1.Target ← param2.Target;
                    end if
                end for
            end if
        end for
    end for
end function
```

Actions	Parameters:		
Terminal:	Type:	Flag:	Sibling_ref:
null			
capture	Enemy	0	-
damage	Location |Enemy	2 | 0	0
exchange	Char | Item | Item	1 | 2 (to give) | 1 (to receive)	0
explore	Location	0	-
free	Char	0	-
gather	Item	0	-
give	Char | Item	0 | 0	-
goto	Location	0	-
kill	Enemy	0	-
listen	Char	1	-
report	Char	0	-
take	Enemy | Item	2 | 0	1 | -
quest	Char	1	-
use	Item	1	-
NonTerminal:			
capture	Enemy	1	-
defeat	Enemy	1	-
get	Item	1	-
give	Char | Item	1 | 2	- | index minus one
goto	Location	1	-
learn	Location	0	-
prepare	-	-	-
report	Char	1	-
rescue	Char	1	-
steal	Item	1	-
subquest	-	-	-

Figure 4.6: Parameter values used

All the generated behavior trees are stored in an XML file called *MissionList* that the translator module has access to.

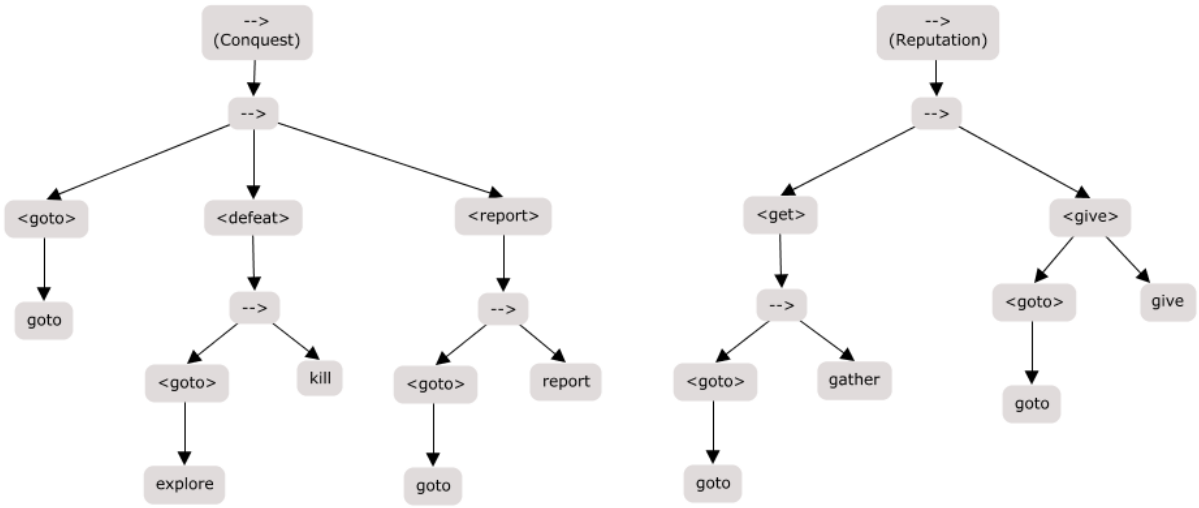

Figure 4.7: Two small Example quests generated by the module

4.2.2 Subquests

As said before, when a quest is being generated, there is a chance that a subquest will be added to the tree structure. For simplicity's sake, this subquest is considered an action.

A subquest can derive from three different intentions (these are taken from the different Strategies and Rules): getting an item, learning information or preparing for an upcoming action. Each of these intentions has the subquest giver say a different line. For example: for the get intention, the subquest giver can say "You're collecting materials for HERO right? That rascal has been owning me a favor for a while. Do this task for me, and I'll consider it paid."; for the learn intention it can say, "You're looking for information? I could give it to you for free... Or you could do something for me first." and for the prepare intention (that changes with the strategy of the subquest) it can say "You should learn a thing or two before continuing your task. I'll help you out with it, since I own one to HERO."

For the subquest to be implemented into Bannerlord, its own "Issue" (name given in the game code to a quest that hasn't been accepted by the player yet) and "Quest" are generated at the appropriate time and from the game's point of view it's just another regular quest from the mod.

These subquests allow the quests to be much more developed with a better and more varied structure, resulting in more interesting quests.

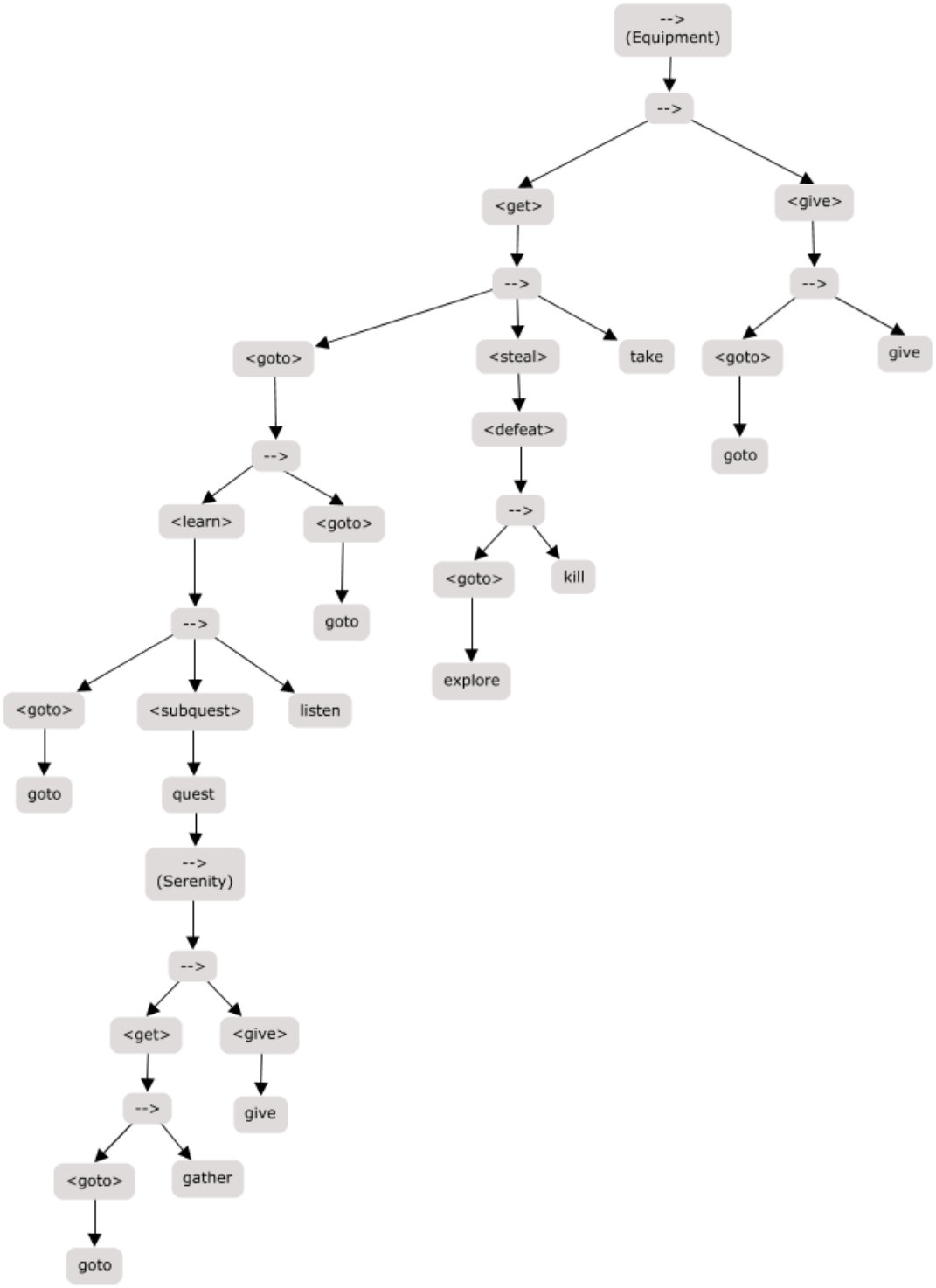

Figure 4.8: Simplified quest tree with a subquest

4.2.3 Alternative way to complete a quest

As mentioned in section 4.2, sometimes quests can be generated with an alternative way to complete them. This means that when the NPC quest giver gives the player a quest, an alternative way to complete it is also presented to the player, and they can choose which way to follow to fulfil the quest.

The Instantiator module knows when there is an alternative way when the head node of the behavior tree is a Selector node instead of a Motivation node. With this, the module separates the tree in two (the main way and the alternative way), but only creates an ”Issue” for the main way. When the quest is accepted, the module creates two ”Quest” instances, one for the main way and one for the alternative way (both with references to each other). When one of them gets completed by the player, the other one is also automatically completed.

From the game’s point of view, the alternative way to complete the quest is considered a ”Special Quest”. This allows it to exist as a quest without the need for an ”Issue”.

For the player, these alternative ways allow them to have quests with a lot more structural variety.

Figure 4.9: Alternative way to complete a quest in game

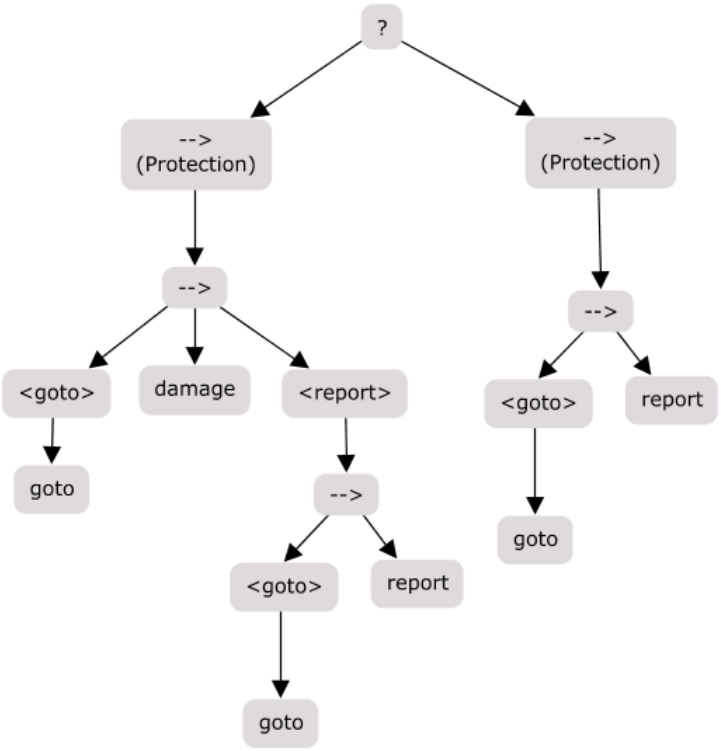

Figure 4.10: Simplified quest tree with an alternative way

4.3 The Instantiator module

Taking a look at the Instantiator module. As said before, this module starts by giving each *Hero* NPC a motivation that depends on the type of NPC.

Having done this assignment, it waits for the game to request a quest for a given NPC. To give a quest to the NPC, it searches in the quest file if there is a quest with the same motivation as the NPC. If there is one, it assigns it to the NPC and creates a new ”Issue”. Afterwards, it deletes the tree from the file and assigns a new Motivation to the NPC.

Upon creating this ”Issue”, the module goes through the selected behavior tree of the quest and creates what we called an *actionTarget* object for each action found in the tree. These *actionTarget* objects change depending on the type of action (e.g.: a go to action will lead to the creation of a *gotoAction* object that extends the *actionTarget* class).

At the same time of creating the *actionTarget*'s, the module also assigns existing world objects to the actions by replacing the world object tags. Actions with the same world object tag end up with the same world object (i.e.: if two actions for example the tag ”place1”, they will both end up with, for example, the settlement ”Poros”) since after choosing the world object, the action propagates its decision through all other actions and if they have the same world object tag it replaces it with the chosen world object.

The behavior used to search for these world objects changes depending on the action (subquest is considered an action for the sake of simplicity):

- *captureAction* - The purpose of this action is to capture a criminal (from the quest givers point of view). Can have, as targets, either an *Hero* NPC or an NPC belonging to a bandit faction (e.g.: Looters). If these targets are still undefined when this action is reached, it searches in the armies roaming around the game world which ones are both enemies and have their faction at war with the quest giver for the party leader. If there are any, it chooses one at random. If there are none, it searches for the armies roaming around the world that are bandit parties and assigns a random one as a target.
- *damageAction* - This action requires the player to damage an entity. Can have as targets, either an *Hero* NPC, an NPC belonging to a bandit faction (e.g.: Looters) or a settlement. If these targets are still undefined when this action is reached, it first gets a random number (either 1 or 2), if the number chosen is 1, there is a village whose faction is at war with the quest giver faction and the next action in line is not a capture action, then that village is selected as the target. Otherwise, it searches in the armies roaming around the game world which ones are both enemies and have their faction at war with the quest giver for the party leader. If there are any, it chooses one at random. If there are none, it searches for the armies roaming around the world that are bandit parties and assigns a random one as the target.
- *exchangeAction* - This action involves a trade of items between the player and the quest giver. Has three targets, two items (one to give and one to receive) and one *Hero* NPC. If these targets are still undefined when this action is reached, it first tries to get an NPC target by seeing what the previous action is. If it's a go to action, then it chooses a random NPC from the settlement target by the go to action. If it's not a go to action, it chooses a random NPC from the settlement that the quest giver belongs to. To get targets for the items, the item to give is chosen at random from a list with all the items in the game. The item to receive is chosen from the items the village has available. The item amounts are defined in a way so that both items have the same total value.
- *exploreAction* - This action makes the player explore a certain area by visiting settlements. Has a dictionary of settlements has a target, each settlement having a tag to indicate if they've been visited already or not. If these targets are still undefined when this action is reached, it searches for three random settlements around the quest giver settlement that belong to the same faction (before distances were calculated using a method from Bannerlord, however, for unknown reasons this could cause the game to crash).
- *freeAction* - This action asks the player to free one of the quest giver allies that has been imprisoned. Has its target, it has a *Hero* NPC. If this target is still undefined when this action is reached,

it first gets a list of all the allies and neutrals of the quest giver. Then finds the nearest enemy settlement whose faction is at war with the quest giver and checks to see if any of its prisoners are in the previously defined list. The first one it finds, it's defined as the target. If none is found, it chooses one at random from the list and imprisons it.

- *gatherAction* - This actions requires the player to gather a randomly chosen item. Its only target, it's an *ItemObject* item. If this target is still undefined when this action is reached, it defines the target by choosing a random one from the list of all the items. It also searches settlements that might contain the item so that it can give a tip to the player. The item amount is set so that the total value the player needs to collect is near 300 of the game currency.

- *giveAction* - The goal of this action is to give the requested item to an NPC. The targets of this action are an *ItemObject* item and a *Hero* NPC. If these targets are still undefined when this action is reached, it first searches for a random NPC from either the quest giver settlement or, if the prior action is a go to action, from the target settlement of the go to action. The item is chosen at random from a list with all the items, and the amount is set so that the total value the player needs to collect is near 300 of the game currency.

- *gotoAction* - This action just asks the player to go to a settlement. Its target is a *Settlement* settlement. If these targets are still undefined when this action is reached, it looks for a random settlement that belongs to the same faction as the quest giver.

- *killAction* - This action asks for the player to kill an NPC. Its target can be either a *Hero* NPC or a Bandit party. If these targets are still undefined when this action is reached, it searches in the armies roaming around the game world which ones are both enemies and have their faction at war with the quest giver for the party leader. If there are any, it chooses one at random. If there are none, it searches for the armies roaming around the world that are bandit parties and assigns a random one as the target.

- *listenAction* - This actions requires the player to talk with an NPC. Its target is a *Hero* NPC. If this target is still undefined when this action is reached, the action first checks if the previous action is a go to action. If it is, the target is chosen at random from the target settlement of the go to action, otherwise, the target is chosen at random from the quest giver's settlement.

- *reportAction* - The purpose of this action is to have the player report to an NPC a certain event. Its target is a *Hero* NPC. If this target is still undefined when this action is reached, the action first checks if the previous action is a go to action. If it is, the target is chosen at random from the target settlement of the go to action, otherwise, the target is chosen at random from the quest giver's settlement.

- *subquestAction* - This is considered an action for the purpose of simplicity, and its goal is to add a subquest to the quest. Its targets are a *Hero* NPC and a *Settlement* settlement. If these targets are still undefined when this action is reached, the action first checks if the previous action is a go to action. If it is, the NPC target is chosen at random from the target settlement of the go to action and the settlement target is defined as the same as the target settlement of the go to action, otherwise, a random settlement from the same faction as the quest giver is found, defined as the target settlement and a random NPC is chosen as the target NPC.

- *takeAction* - The goal of this action is to have the player steal an item from either a *Hero* NPC, a bandit party or a settlement. Can have as targets, either an *Hero* NPC, an NPC belonging to a bandit faction (e.g.: Looters) or a settlement and a *ItemObject* item. If the NPC/settlement target is still undefined when this action is reached, it means the target is going to be and settlement, so it checks if the prior action is a damage action and if it is, it gets the settlement target from that action. If the item target is still undefined, it checks what kind of NPC/settlement target it has and the item target is chosen at random from the NPC/settlement target inventory. The amount is chosen so that the total value is near 300 in-game currency.

- *useAction* - The purpose of this action is to have the player level up a skill. Its target, it's a *SkillObject* skill. If this target is still undefined when this action is reached, then a skill from the list of all the skills is chosen at random and set as the target. The amount of levels needed is set as a random number between 5 and 10.

Usually, behavior trees are traversed every execution tick, however, we only require for it to be traversed once to define all the world objects.

This module is also responsible for defining the titles, descriptions and quest giver dialogues that the player interacts with.

The title is made from the chosen Strategy and the world object assigned to the main action of the Strategy. The description is just the Strategy. When the quest giver gives the quest it first gives out a sentence related to the Motivation, then gives a general description of the quest depending on the Strategy and the main action and finally describes the main steps the players needs to take to complete the quest.

The quest reward is also defined here. It depends on the type of actions of the quest and the amount of time the quest might take to complete.

Figure 4.11: Quest description and title example

Figure 4.12: Accepting a new quest

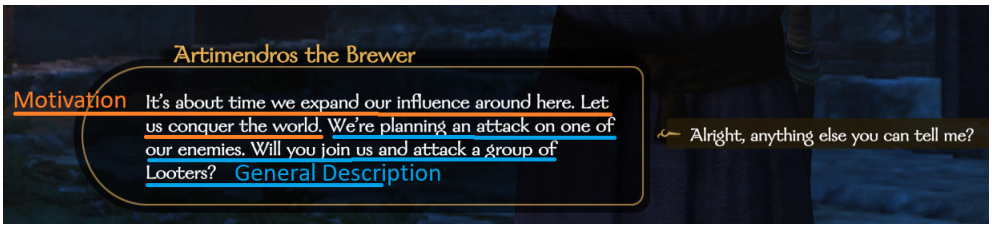

Figure 4.13: Motivation and general description example for an NPC with *Conquest* Motivation and *Attack Enemy* Strategy

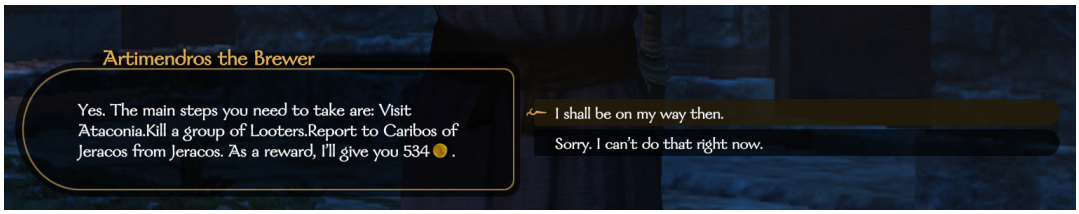

Figure 4.14: Step by step description of a quest

4.4 The Monitor module

This is the final module. It starts when the player accepts the quest from the NPC quest giver, and its main tasks are to track the player progress and to add and update the quest logs that appear on the quest screen. It's also responsible for adding the necessary dialogues to the NPC's the player needs to interact with to complete some actions.

To track player progress, the module makes use of events. Events are what the game uses to know when anything happens (for example: when the player enters a settlement, an event is triggered and any method that is listening to that event is also triggered). Each action makes use of one or several events to track their progress. For each action they are:

- *captureAction* - the quest logs are defined as, for example, "You've been requested to capture HERO". When the player captures the target, the events that can be triggered are *HeroPrisonerTaken* or *OnPrisonerTakenEvent*, the quest log changes to, for example, "Deliver the prisoner to HERO in QUEST_SETTLEMENT." and a dialogue is added to the quest giver, so it can receive the target. The action ends as successful when the player delivers the target to the quest giver.
- *damageAction* - the quest logs are defined as, for example, "Attack HERO's party.". When the player defeats the target, the events that can be triggered are *RaidCompletedEvent*, *HeroKilledEvent* or *MapEventEnded* and the action ends as successful.
- *exchangeAction* - the quest logs are defined as, for example, "Exchange ITEM_AMOUNT1 ITEM_NAME1 for ITEM_AMOUNT2 ITEM_NAME2 with HERO". A new dialogue is added to the NPC target so

that the trade can be conducted. The dialogue itself can detect if the player has enough items, and the action ends as successful when the trade is completed.

- *exploreAction* - the quest logs are defined as, for example, ”Explore the area and visit SETTLEMENT1, SETTLEMENT2, SETTLEMENT3.”. When the player enters a settlement, the event *SettlementEntered* is triggered and the action sees if the settlement is part of the dictionary and if it has not been visited yet, updating the dictionary accordingly. The action ends as successful when the player visits the three chosen settlements.

- *freeAction* - the quest logs are defined as, for example, ”Find a way to liberate HERO from SETTLEMENT.”. When the player frees an NPC, the events that can be triggered are *HeroPrisonerReleased* or *PrisonersChangeInSettlement* and the action is considered successful when the NPC is released.

- *gatherAction* - the quest logs are defined as, for example, ”Gather ITEM_AMOUNT ITEM_NAME. You can maybe find some in SETTLEMENT.” The gathering of the item can be detected by the events: *PlayerInventoryExchangeEvent*, *OnEquipmentSmeltedByHeroEvent*, *OnNewItemCraftedEvent*, *OnItemProducedEvent* or *ItemsLooted* and the action ends as successful when the desired amount is reached.

- *giveAction* - the quest logs are defined as, for example, ”Give ITEM_AMOUNT ITEM_NAME to HERO”. The action also adds a new dialogue to the target NPC so that the item can be given. The dialogue itself can detect if the player has enough items, and the action ends as successful when the item has been given.

- *gotoAction* - the quest logs are defined as, for example, ”Visit the settlement of SETTLEMENT.” When entering a settlement, the event *SettlementEntered* is triggered and the action checks if it’s the target settlement. If it is, then the action checks if the next action is also a go to action. When the next action is also a go to action, a dialogue is added to a random NPC of the settlement and the quest log changes to ”Talk with HERO”. This dialogue is the NPC telling the player that whatever they are looking for is not on that settlement, and doing this dialogue ends the action as successful. If the next action is not a go to action, then the action ends as successful.

- *killAction* - the quest logs are defined as, for example, ””Kill HERO.” When the player kills the target, the events that can be triggered are *HeroKilledEvent* or *MapEventEnded* and the action ends as successful.

- *listenAction* - the quest logs are defined as, for example,”HERO from SETTLEMENT might have the information you need, listen to what he has to say.” and a new dialogue is added to the target NPC. The information given by the NPC to the player changes depending on the Strategy chosen

for the quest and if there is a subquest action prior to this listen action. Talking with the target NPC ends the action as successful.

- *reportAction* - the quest logs are defined as, for example, ”You’ve completed your task. Report the events to HERO from SETTLEMENT.”. and a new dialogue is added to the target NPC. The report given by the player through the dialogue changes depending on the Strategy chosen for the quest. Talking with the target NPC ends the action as successful.
- *subquestAction* - As mentioned before, the subquests are defined as an action. When a subquest is reached by the Monitor module, a new ”Issue” is created and assigned to the target NPC with the generated subquest. The action then defines the quest logs, setting the log text as, for example, ”It appears someone wants to have a word with you. Talk with HERO in SETTLEMENT and find out what he wants.”. When the subquest is completed, the *OnQuestCompletedEvent* is triggered and the action ends as successful.
- *takeAction* - the quest logs are defined as, for example, ”Take AMOUNT ITEM from HERO.”. The action ends as successful when the required amount of items as been taken. This is done by listening to the events *MapEventStarted* and *MapEventEnded*.
- *useAction* - the quest logs are defined as, for example, ”Go train and level up your SKILL by at least AMOUNT levels.”. Levelling up a skill triggers the *HeroGainedSkill* event, and the action ends as successful when the player has levelled up the chosen amount of levels.

In this module, the behavior tree is traversed more than once, on the contrary to the Instantiator module. Although it’s still not once every single frame, the behavior tree is traversed once when the quest is accepted and then once every time an action is completed. This is done so that the next action can be grabbed and its logs and dialogues, if applicable, can be initialized.

Figure 4.15: Quest on the quest screen

Figure 4.16: Quest state after completing a step

4.4.1 Save and Load system

Due to the limitations of Bannerlord's own save and load system, we had to implement our own system. A new XML file is created when the game creates a new ”Issue” and another file is created when the game creates a new ”Quest” from that ”Issue”. These files contain the quest behavior trees and are used to store strings of the objects used in the actions. For example, if a quest has the ”go to” action, then the file is going to contain that action stored with a string representative of the target settlement of the action. These files are also saved when the player saves the game.

When the player returns to the game, the files are loaded and each action brings back the original target objects using the stored strings.

5

Evaluation

Contents

To evaluate the quality and enjoyment of the mod we made, at first we decided to publicly publish the mod and later to perform user tests. Publishing the mod online allowed us to have players used to playing Bannerlord test out our mod, but it was harder to get more concrete results. Performing user tests meant the players were very unlikely to have experience with Bannerlord, however, we were able to get proper statistical results. Through this we were able to determine if the generator we made generated more varied quests with a more diverse structure and if the players got the idea that the quests were made by humans. This will allow seeing if we can reach our goal of generating quests in an automatic way that are considered as good as the quests from Bannerlord.

5.1 Metrics

To know how our quests compared with the Bannerlord quests, we evaluated two different metrics, Quest Quality and Quest enjoyment.

To evaluate Quest Quality, we created a set of questions that would help us better understand what the player thought. These were:

- Quest Length - using a Likert scale (from 1 to 5, in the form of 5 scalable options) we asked the players if they thought the length of the quests was suitable;

- Quest Cohesion - using three options (0 - The steps to complete the quest were very messy and didn't make sense; 1 - Most steps made sense, but a few of them didn't seem to fit in; 2 - Structurally sound, each step being believable and making sense for it to be) know what the players thought about the cohesion of the quests, meaning if they were consistent and made sense throughout their completion;

- Quest Step Clarity - using 4 options (0 - None of them were clear on what I needed to accomplish to complete them; 1 - Just a few of them were clear and most of them were confusing; 2 - Most of them were clear, but some of them were a bit confusing; 3 - Clear on what I needed to accomplish) to know the opinion of the players on the clarity of each step of the quest, this is, the clarity of every action the player needed to complete;

- Quest Description Clarity - using a Likert scale (from 1 to 7) to know the opinion of the players regarding the clarity of the provided quest descriptions and dialogues;

- Quest Dialogue Immersion - using a Likert scale (from 1 to 7) to See the player's opinion regarding how immersive the dialogues of each quest felt. This is, how much the players were drawn into the game world.

- Quests made by humans - using a Likert scale (from 1 to 7) to find out what the player thought about the origin of the quests, if they were made by humans or machine-generated.

To evaluate Quest Enjoyment we used an adaptation of the subscale ”enjoyment” from GUESS; Phan et al. 2016 [7], that specifically targeted the quests, so that we could know if the players enjoyed the quests. The questions were: ”I think the quests are fun”, ”I feel bored while completing the quests”, ”If given the chance, I want to complete more quests”, ”I enjoy completing the quests” and ”I feel like the quests are repetitive”.

5.2 Nexus Mods publication

Figure 5.1: Plot Lords logo on the Nexus Mods website

The mod was published on the most popular mod website for Bannerlord mods: Nexus mods [19]. It was published on the 22nd August 2021 [20] and in 60 days it reached over 2600 unique downloads, over 19700 views and over 35 endorsements (a way for users to show their appreciation for a mod). With this publishing, multiple bugs were found and fixed, and several feedbacks were made, the grand majority of them positive. The mod page contained a very vague description of the mod to not cause bias to the players when answering the questionnaire.

One of the goals with the publishing of the mod online was to have players with experience in the game to answer a questionnaire A.1 that evaluated:

- Demographics (Age, Gender, time spent playing games, if they ever played Bannerlord, etc.);
- Player enjoyment of the game (using the subscale ”enjoyment” from GUESS; Phan et al. 2016)

- Bannerlord quests quality (custom questions);
- Bannerlord quests enjoyment (questions derived from the subscale ”enjoyment” from GUESS; Phan et al. 2016);
- Plot Lords quests quality (custom questions);
- Plot Lords quests enjoyment (questions derived from the subscale ”enjoyment” from GUESS; Phan et al. 2016);

Even though we had many people downloading the mod, we, unfortunately, only ended up with 8 answers to the questionnaire, meaning that they are not statistically relevant enough for us to analyze, so we had to resort to in-person user tests to have results. This lack of answers might be due to the fact that there was no incentive for people to answer the questionnaire.

5.2.1 Mod advertisement

To advertise our mod, all the students working on a thesis related to Bannerlord joined together on a public discord server and invited people to join said server, in an attempt to boost each other's visibility on the mod. We also had a reference to each other's mods on our mod pages to further boots views. In total, over 110 people joined the server and a few of them interacted with us through the text channels.

We also got lucky and our mod was featured in the video of a somewhat popular YouTube channel [14]. This helped us gain a lot of attention and allowed us to have a high number of views and downloads.

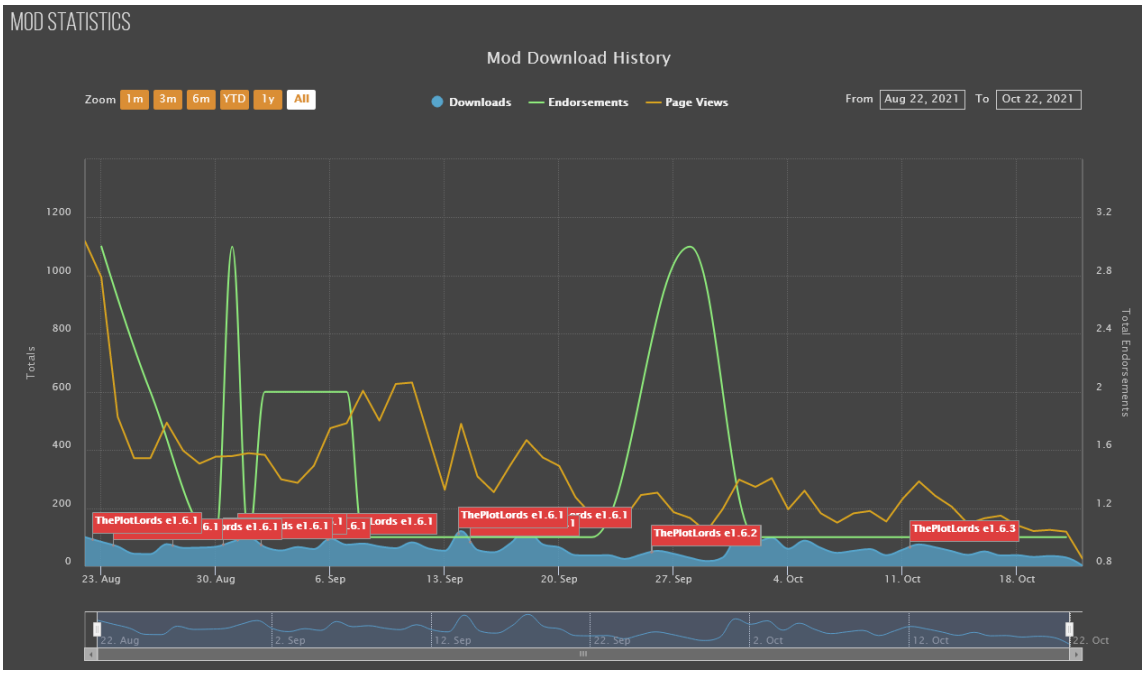

Figure 5.2: Plot Lords statistics in the Nexus mods website

5.2.2 Mod Community

Most of our interaction with the players that tried out our mod came down to bug fixing. This is, the player would communicate with us either through the comment section or bug report section of the mod page or the discord server. The discord server didn't have much activity, however, the comment section and bug report section of the mod page presented quite a lot of activity with over 60 comments with players leaving their feedback and asking questions and even suggestions.

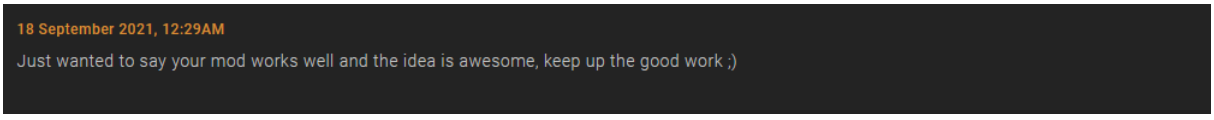

Figure 5.3: Example of a comment left as feedback

5.3 User Tests

The user tests were done in the span of two weeks, the participants were part of the Instituto Superior Técnico - TagusPark community, which is also where the tests were conducted. Each participant was given a 5€ gift card to use on the steam platform as incentive to participate. One of the reasons to conduct these user tests was due to the lack of responses to the online questionnaire.

5.3.1 Procedure

Accompanying the user tests, we had the players fill out a questionnaire. This test had a duration of between 30 and 40 minutes. The questionnaire consisted of a Demographics part and then two sections both used to evaluate the quest quality and quest enjoyment, one section for the first two quests and the other section for the last two quests (it could either be: First two quests $\rightarrow$ Bannerlord quests, Last two quests $\rightarrow$ Plot Lords quests or First two quests $\rightarrow$ Plot Lords quests, Last two quests $\rightarrow$ Bannerlord quests). The players were also accompanied during the test in case they had any questions regarding the game since, for most of them, it was their first time playing Bannerlord.

The performed user test was simple. Firstly, if the player hadn't played any of the Mount & Blade games before, they would play a small tutorial where its only goal would be to complete one of Bannerlord's quest entitled "HERO wants his daughter found". Afterwards, the player would fill out the Demographics part of the questionnaire and enter another already prepared save-file. Depending on their version, the player would either start out by completing two Bannerlord quests (with the titles of "Landowner needs manual laborers" and "Extortion by Deserters at SETTLEMENT") and responding to the related part of the questionnaire, or start by completing two Plot Lords quests (with the titles of "Check on HERO" and "Capture someone from the Looters") and responding to the related part of the questionnaire, following with whichever pair of quests they didn't start with. Unless the players knew the Bannerlord quests by heart, the two pairs of quests were indistinguishable and all the players played the same quests.

Our goal with the types of quests was to have them be of a similar type, so has to not cause bias. To choose the Bannerlord quests, we had to go through trial and error until we found two that were of similar Motivation and were somewhat close to each other on the map. To choose the Plot Lords quests, we generated two quests with a Motivation similar to the Bannerlord quests, in this case, either Protection or Serenity, and that were a bit small so that the time limit of the user test was kept at around 40 minutes. Afterwards, we just forced the game to assign those quests to settlements nearby to the Bannerlord quests.

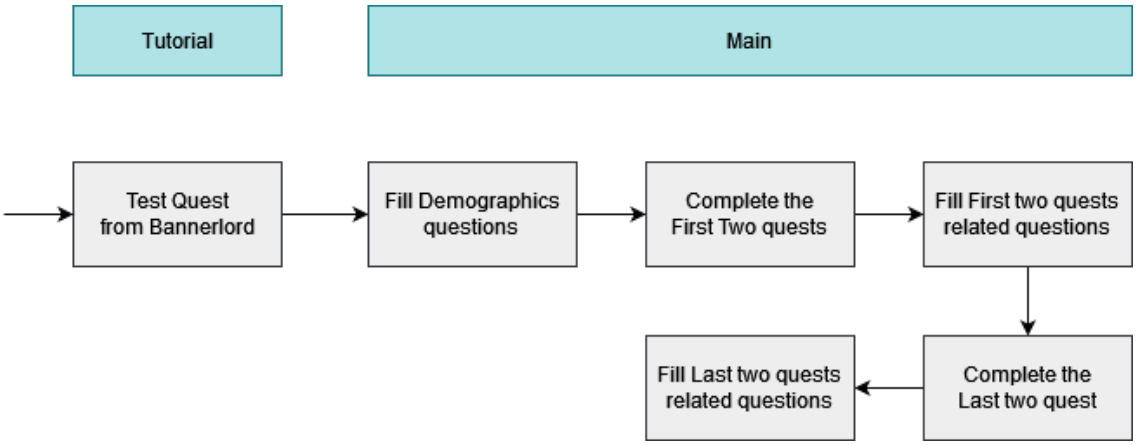

Figure 5.4: User Test Procedure

5.3.2 Results

We're now going to analyze the user tests results. Since each tester experienced both types of quests (Bannerlord and Plot Lords quests), we are dealing with a statistical scenario corresponding to repeated measures.

5.3.2.A Demographics

Regarding demographics. We had a total of 31 participants, where 87% were male and 13% were female. The overall age of our participants was between 18-25 years old, and a bit over 60% of them played games on a daily basis. The rest of them said they played weekly (13%), monthly (13%), once in 6 months (3%), once a year (3%) and either less than once a year or never (6%). Only one participant had played Bannerlord before, and only 3 participants played M&B Warband before. However, over 87% claimed to have played other RPGs/Open World games before.

Despite barely having someone with experience with Mount & Blade games, most of our participants were regular players and had experience with the same type of games as Bannerlord (this is, RPG and Open World games).

It's also worth noticing that over 70% of the players noticed that the two pairs of quests were different from one another.

5.3.2.B Quest Quality

A Cronbach Alpha test was conducted to measure the internal consistency of the questions. It was done for both the questions related to the Bannerlord quests and to the Plot Lord quests.

- Bannerlord quests - We got a result of 0.487, meaning the internal consistency was unacceptable.

- Plot Lord quests - We got a result of 0.638, meaning the internal consistency was questionable.

With these results, we decided to not make the average of the results and instead analyze and compare each question individually.

To do this analysis, we first needed to find out why kind of data we had, non-parametric or parametric. So we conducted Tests of Normality, namely the Shapiro-Wilks test. Having conducted this test we had all levels of significance (the *p*-value) lower than 0.05, so we conclude that the data does not fit the normal distribution, and therefore it's non-parametric.

According to [4], the test that should be done when analyzing repeated measures and non-normal data is the Wilcoxon's rank paired test.

5.3.2.C Quest Length

We got a significance value of 0.166, meaning that we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data.

From the box plot, we can conclude that most players thought the quests had the right size.

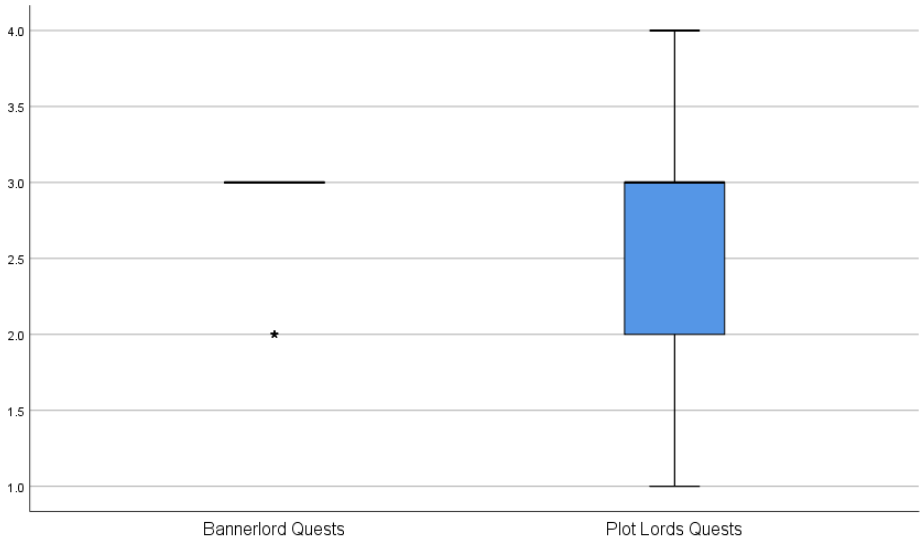

Figure 5.5: Quests Length opinion

5.3.2.D Quest Cohesion

We got a significance value of 0.046, meaning that we should reject the null hypothesis, this is, there is a statistical significance between the two sets of data. This shows that, in terms of Quest Cohesion our mod was a bit different from the Bannerlord quests.

From the histogram we can conclude that most players thought the quests were structurally sound, however there were a few that thought some steps didn't seem to fit in, mainly with the Plot lords quests.

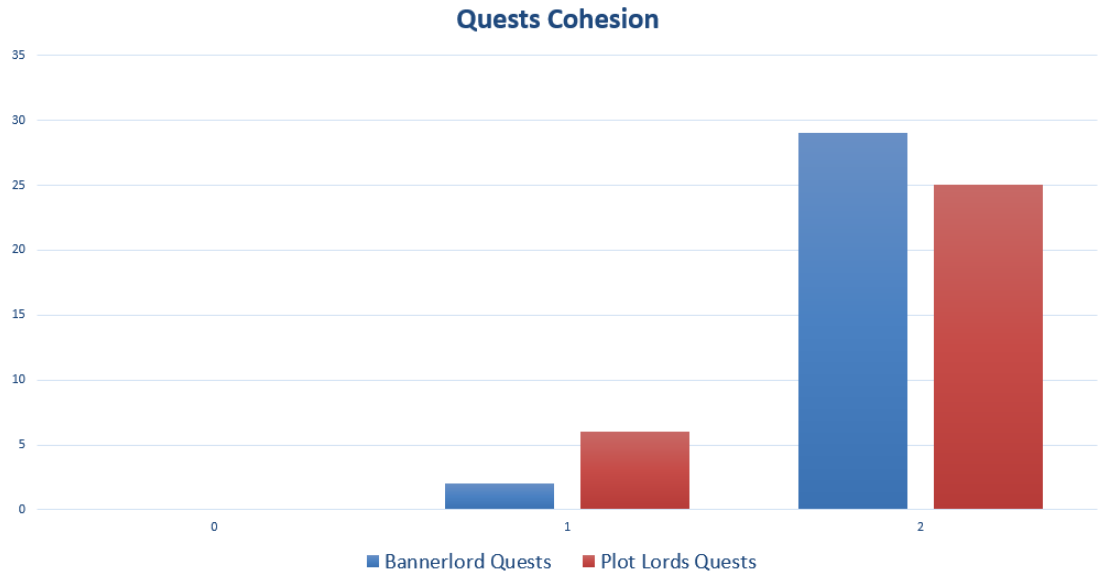

Figure 5.6: Quests Cohesion opinion: 0 - The steps to complete the quest were very messy and didn't make sense; 1 - Most steps made sense, but a few of them didn't seem to fit in; 2 - Structurally sound, each step being believable and making sense for it to be

5.3.2.E Quest Step Clarity

We got a significance value of 0.317, meaning that we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data. This shows that, in terms of Quest step clarity, our mod was similar to the Bannerlord quests.

From the histogram, we see that the players thought both pairs of quests were good in terms of Quest step Clarity.

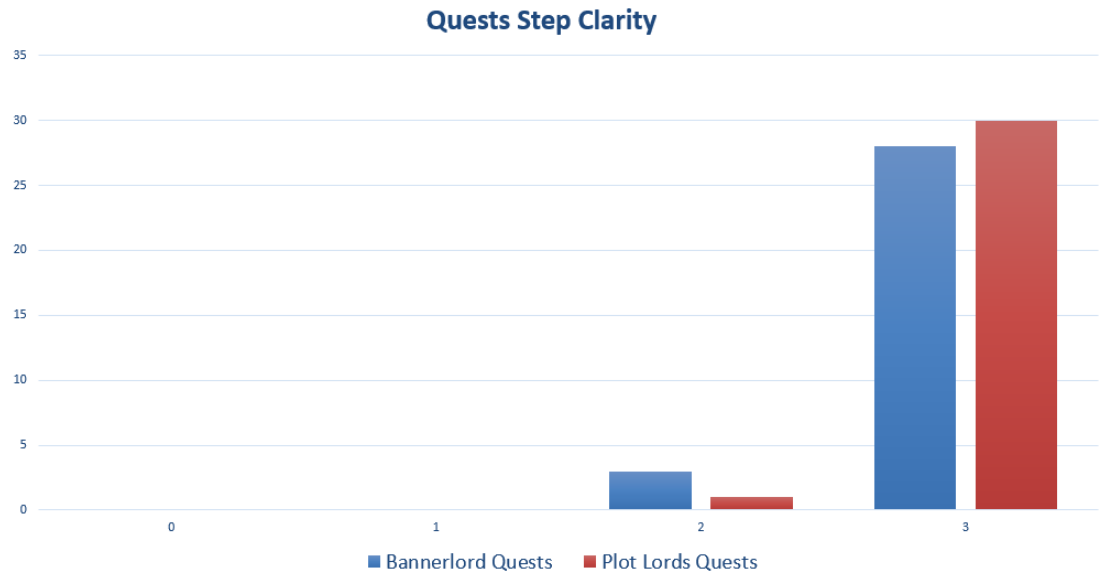

Figure 5.7: Quest Step Clarity opinion: 0 - None of them were clear on what I needed to accomplish to complete them; 1 - Just a few of them were clear and most of them were confusing; 2 - Most of them were clear, but some of them were a bit confusing; 3 - Clear on what I needed to accomplish

5.3.2.F Quest Description Clarity

We got a significance value of 0.154, meaning that we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data. This shows that, in terms of Quest Description clarity, our mod was similar to the Bannerlord quests.

From the box plot, we see that the players thought both pairs of quests were good in terms of Quest Description Clarity.

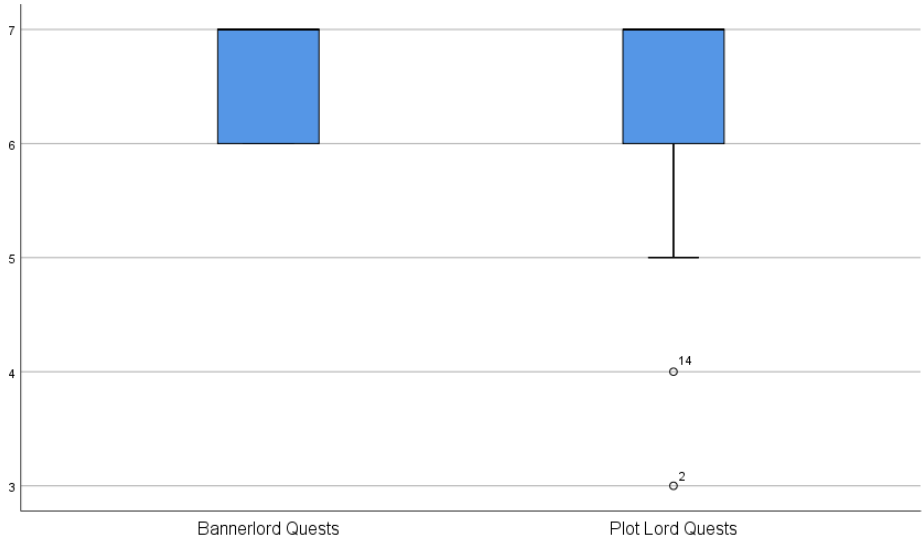

Figure 5.8: Quests Description Clarity opinion

5.3.2.G Quest Dialogue Immersion

We got a significance value of 0.120, meaning that we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data. This shows that, in terms of Quest Dialogue Immersion, our mod was similar to the Bannerlord quests.

From the box plot, we see that the players thought both pairs of quests were good in terms of Quest Dialogue Immersion, however, the Plot lords got a few negative outliers. Still, these are not enough for us to consider there is a statistical significance, therefore we can't say that there is a difference between both sets of quests.

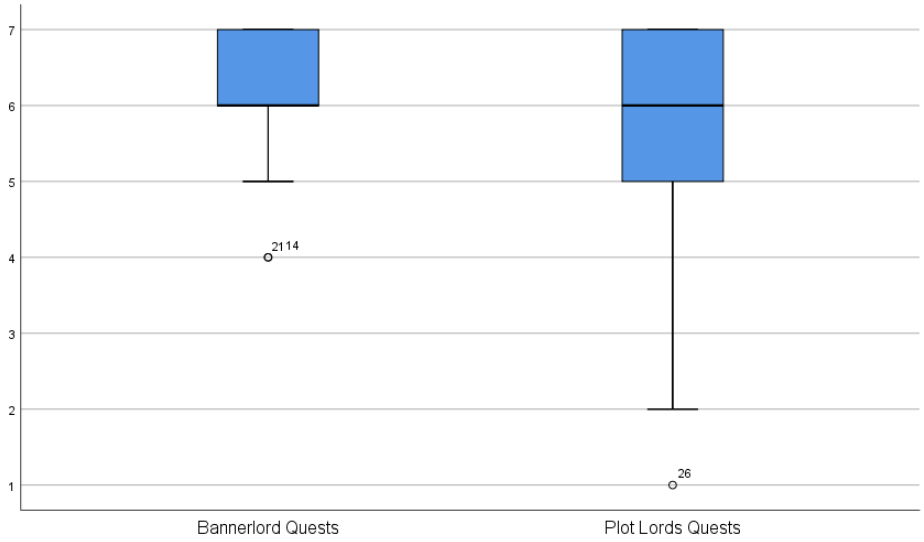

Figure 5.9: Quest Dialogue Immersion opinion

5.3.2.H Quests made by humans

We got a significance value of 0.773, meaning that we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data. This shows that, in terms of thinking the quests were made by humans, our mod was similar to the Bannerlord quests.

From the box plot, we see that there are no statistical differences between the Plot lords quests and the Bannerlord quests when it comes to thinking the quests were made by humans.

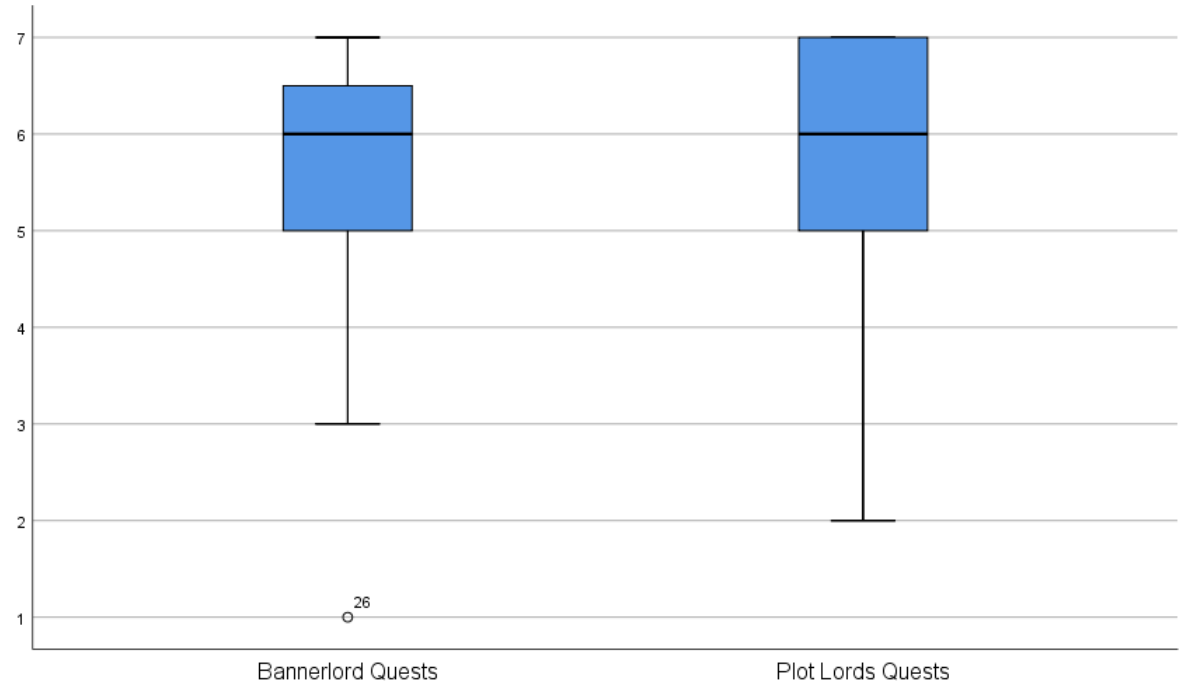

Figure 5.10: Opinion on ”Were the quests made by humans?”

5.3.2.I Quest Enjoyment

A Cronbach Alpha test was conducted to measure the internal consistency of the questions. It was done for both the questions related to the Bannerlord quests and to the Plot Lord quests.

- Bannerlord quests - We got a result of 0.840, meaning the internal consistency was good.
- Plot Lord quests - We got a result of 0.832, meaning the internal consistency was good.

With these results, we decided to make the mean of the results and analyze those means.

To do this analysis, we first needed to find out why kind of data we had, non-parametric or parametric. So we conducted Tests of Normality, namely the Shapiro-Wilks test. Having conducted this test, we had all levels of significance (the *p*-value) lower than 0.05, so we concluded that the data does not fit the normal distribution, and therefore it's non-parametric.

According to [4], the test that should be done when analyzing repeated measures and non-normal data is the Wilcoxon's rank paired test.

The Wilcoxon rank paired test gave a significance of 0.011, which means we should reject the null hypothesis, this is, there is a statistical significance between the two sets of data. We got an effect size of -0.457, meaning the effect is average.

From the box plot, we see that the players enjoyed the Bannerlord quests slightly more than the Plot lords quests.

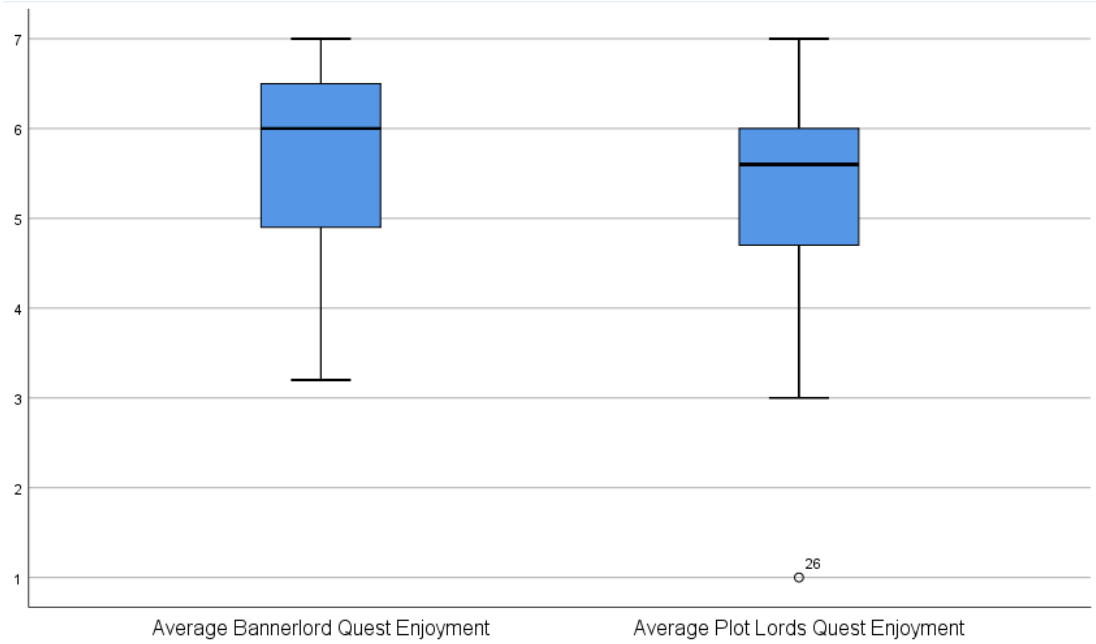

Figure 5.11: Opinion on Quest Enjoyment without order

To better know the cause for this difference we analyzed each question individually:

- ”I think the quests are fun” - Significance level of 0.026, which means we should reject the null hypothesis, this is, there is a statistical significance between the two sets of data.

- ”I feel bored while completing the quests” - Significance level of 0.067, which means we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data.

- ”If given the chance, I want to complete more quests” - Significance level of 0.806, which means we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data.

- ”I enjoy completing the quests” - Significance level of 0.166, which means we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data.

- ”I feel like the quests are repetitive” - Significance level of 0.074, which means we should retain the null hypothesis, this is, there is no statistical significance between the two sets of data.

From this, we can see that the players thought the Bannerlord quests were simply slightly more fun than the Plot Lords quests.

5.4 Discussion

Discussing now the results, in general, we got what we expected since the testing conditions weren't perfect. The ideal tests would have been done with players already experienced with Bannerlord and capable of testing more than two quests from Plot Lords so that they could experience the variety of quests the mod is capable of providing. With only two quests, this was an impossible metric to measure.

Even though most of these users had experience with RPG/Open World games (the same type as Bannerlord), which was an advantage when conducting the tests, most of them had never played Bannerlord. Since they only played between 30 and 40 minutes, there wasn't enough time for them to experience both the base game and the quests of Plot Lords, so it's possible the test results would change given more time and experience with the game and the mod. It's also important to notice that everything from the mod was automatically generated, and all the NPC dialogues and step descriptions were written by us, so some English from the descriptions and such could not be perfect, something that could affect user experience. All these nuisances might have contributed for the Plot lords quest to have very slightly lower results when in comparison with the Bannerlord quests.

Nevertheless, we think we accomplished our goals of providing the user with an experience almost as good as the base one, while at the same time we generated quests more varied in an automated way.

In terms of quest quality, since each question targets a different metric to measure, we have to look at them individually.

Regarding the Quest Length, most players though the quests had the right size and there was no statistical difference between the two sets of data, there were, however, a few that though the Plot lords quests were a bit on the small side. This is a great result, since the chosen quests for Plot Lords were purposely chosen because they were on the smaller side. This was done so that the time spent in the user test would not go beyond the 40 minutes. Due to being automatically generated, the quest sizes can vary a lot, and some of them could have a tendency to become a bit time-consuming, especially if they had sub-quests.

Looking at Quest Cohesion, there was a statistical difference, meaning the players thought the Bannerlord quests were a bit more structurally sound than the Plot Lords quests. This was also expected, because there was more time spent to refine each one of Bannerlord quests, while with the Plot lords quest, since they were automatically generated, it's possible some steps weren't perfectly fit into the quest generation.

Taking a look at the Quest step Clarity. There was no statistical difference between the two sets of data, so both quest pairs had good step clarity, this is, each step in the quests was easy for the player to understand, and they knew what they needed to do.

Next, Quest Description Clarity. Again, there is no statistical difference between the two sets of data, so both quest pairs had good description clarity, this is, the dialogue and descriptions provided by the

quest giver to the player were good and understandable.

Afterwards, the Quest Description Immersion. Once again, there is no statistical difference between the two sets of data, so both quest pairs had good dialogue immersion, this is, the dialogues proved to be immersive to the players. It's important to notice the negative outliers on the plot lords quests. They might be due to the automated generation of text that might have some problems that affected the user experience. Still, they are not enough for us to consider there is a statistical significance.

Finally, the opinion on the quest origin, this is, if they were made by humans. Once again, there is no statistical difference between the two sets of data, so the players didn't notice any difference between the quests, and they thought both of them were made by humans. This result is great to see, since this means that there's a possibility that a system like this could be implemented in another game, and players might not even notice that the quests from the game are being procedurally generated.

Concerning the quest enjoyment. We didn't get the expected results, but they were still positive considering the challenge we had to face, since the quests were automatically generated and from what we saw above there are a few problems with the quest cohesion that might have affected the user experience. There was a statistical significance between the two sets of data, so the players enjoyed the Bannerlord quests slightly more than the Plot lords quests. The main reason for this, from the analysis we did afterwards of each individual question related to the enjoyment, seems to be that the players thought the Bannerlord quests were slightly more fun than the Plot Lords quests. The reason for this might be because the chosen Bannerlord quests involved more fighting (one of the most attractive features in Bannerlord) than the Plot lords quests that made the player walk a bit more around the map (a feature some players might find less fun). Since for the vast majority of the players it was the first time they played Bannerlord they might have become more captivated with the combat, that was more present in the Bannerlord quests, so they found the Bannerlord quests a bit more fun.

6

Conclusion and Future work

Contents

The main goal of this thesis was to implement a generator capable of creating quests that could match human-made quests and even be mistaken with them, with more diversity and a much more varied structure.

To achieve this we adjusted the generator envisioned by Jonathan Doran and Ian Parberry with the modifications done by António Machado to fit a game with a quest system already in place, Mount & Blade II: Bannerlord. These adjustments consist of: an adaptation of the number of actions to fit Bannerlord, a change to the algorithm that assigns the world objects to the actions, the addition of a behavior tree system and the addition of alternative ways to complete the quest.

We believe these changes made the structure of the generated quests more varied, improving the player experience.

The results we got were overall quite positive. We were able to produce quests that, even though were slightly inferior to the Bannerlord quests in terms of enjoyment, were comparable in terms of quality, only noticeably less good when it came to quest cohesion. The automatically generated quests proved to have a good overall quality and were enjoyed by the players. They also proved to be indistinguishable from human-made quests which is our most positive result and, as said before, demonstrates the system could be applied to another game and the players could not even notice the quests were being procedurally generated.

With these results, we believe the Bannerlord Quest Generator was able to create quests with great diversity and varied structure that players enjoyed. We think this is a step forward in the gaming industry, since the generated quests helped in improving the replayability and enjoyment of Bannerlord, and much like Bannerlord, this Generator could be applied to other games (with or without a questing system) to help improve their replayability and reducing the time and financial costs associated with quest creation.

6.1 Future Work

The best way to continue the work we've done is to finish implementing all the actions proposed by Jonathan Doran and Ian Parberry with the modifications done by António Machado. We believe the best environment to accomplish this would be in a game that was done from the ground up with a procedural quest generation system.

The system currently has a few issues with making sure all the actions in the quests are coherent, and all the world objects are correctly assigned when there are chained actions and rules. Our work already made a few changes to better assure these issues are corrected (with the exceptions implemented in the "BindParameters" algorithm 4.2.1.E). However, we believe the algorithm and the flag system used (flag 0 for parent restriction, 1 for no restriction (also known as new case) and 2 for sibling restriction), need to be reviewed and receive an overhaul, especially if more actions are added since more actions

mean more potential problems. We think, for example, the addition of more flags could help.

The text generation part of the quests is also not perfect. So further work on this component would also allow for even greater quests to be generated.

Bibliography

[1] Breault, V.: Multi-agent planning for interactive emergent narrative systemss. Master's thesis, Carleton University (2015)

[2] Broekema, C.: Drama management implementation for interactive digital storytelling (2013)

[3] Doran, J., Parberry, I.: A prototype quest generator based on a structural analysis of quests from four mmorpgs. In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games. PCGames '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2000919.2000920, https://doi.org/10.1145/2000919.2000920, last accessed 26 October 2021

[4] Flowchart to choose a statistical test, https://www.intro2r.info/unit3/which-test.html, last accessed 26 October 2021

[5] Machado, A.: A procedural quest generator for Conan Exiles. Master's thesis, Universidade de Lisboa - Instituto Superior Técnico (2017)

[6] Machado, A., Santos, P.A., Dias, J.: Towards a procedurally generated experience : A structural analysis of quests. Universidade de Lisboa - Instituto Superior Técnico (2017)

[7] Mikki, P., Keebler, J.R., S. Chaparro, B.: The development and validation of the game user experience satisfaction scale (guess). Human Factors The Journal of the Human Factors and Ergonomics Society (September 2016). https://doi.org/10.1177/0018720816669646

[8] Millington, I., Funge, J.: Artificial Intelligence for Games, Second Edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edn. (2009)

[9] Mimesis aquarium image. 21 December 2020, https://www.researchgate.net/figure/A-screenshot-of-the-Mimesis-Virtual-Aquarium-a-simulation-of-a-portion-of-the-Monterey_fig1_270960091/, last accessed 26 October 2021

[10] Poor, N.: Computer game modders' motivations and sense of community: A mixed-methods approach. New Media & Society **16**, 1249–1267 (11 2013). https://doi.org/10.1177/1461444813504266

[11] Teutenberg, J., Porteous, J.: Efficient intent-based narrative generation using multiple planning agents (05 2013)

[12] Young, R.: An overview of the mimesis architecture: Integrating intelligent narrative control into an existing gaming environment (2001)

[13] Young, R., O. Riedl, M., Branly, M., Jhala, A., Martin, R.J., Saretto, C.J.: An architecture for integrating plan-based behavior generation with interactive game environments. J. Game Dev. **1**, 1–29 (2004)

Game References

[14] Artem: Mount blade 2 bannerlord mods 26: Banners, torches, custom troops more!, https://youtu.be/bkC4nKkeTaE?t=44, last accessed 26 October 2021

[15] Mount and blade ii: Bannerlord game. 21 December 2020, https://store.steampowered.com/app/261550/Mount__Blade_II_Bannerlord/, last accessed 26 October 2021

[16] Bannerlord documentation. 21 December 2020, https://docs.bannerlordmodding.com/_intro/getting-started.html#tools, last accessed 26 October 2021

[17] Conan exiles game. 21 December 2020, https://store.playstation.com/pt-pt/product/EP1833-CUSA10486_00-CONANEXILES00001

[18] Harmony library, https://harmony.pardeike.net/articles/intro.html, last accessed 26 October 2021

[19] Nexus mods website, https://www.nexusmods.com, last accessed 26 October 2021

[20] The plot lords mod. 22 August 2021, https://www.nexusmods.com/mountandblade2bannerlord/mods/3243, last accessed 26 October 2021

[21] The witcher iii: Wild hunt. 4 January 2021, https://store.steampowered.com/app/292030/The_Witcher_3_Wild_Hunt/, last accessed 26 October 2021

A

Questionnaires

A.1 Nexus Mod Questionnaire

M&B II Bannelord mod - The Plot Lords

This form is part of a Master thesis by Marco Cabral, a student of Instituto Superior Técnico, University of Lisbon and is to be filled out after playing with the Plot Lords mod. This mod mainly afects the quest system, mainly by adding new quests.

In this form we hope to measure the player's game experience when playing with our mod.

A few key points:
- participation is voluntary and you can withdraw at any time;
- at no point throughout the study will you be identifiable, and individual results will not be disclosed;
- there are no physical or psychological hazards in your participation;
- you are free to ask any inquiry about the experiment at any time (email: marco.cabral@tecnico.ulisboa.pt, discord: DezLezz#8307 or in the project's discord https://discord.gg/eysf5TNagj)

By proceeding to the questionnaire you are giving your consent to the above conditions

* Required

Demographics

1. Gender *

Mark only one oval.

- ◯ Female
- ◯ Male
- ◯ Prefer not to say
- ◯ Other: _____

2. Age *

Mark only one oval.

- ◯ Younger than 18 years old
- ◯ 18-25 years old
- ◯ 26-35 years old
- ◯ 36-45 years old
- ◯ older than 45 years old
- ◯ Prefer not to say

3. How often (approximately) do you currently play video games? *

Mark only one oval.

- ◯ Daily
- ◯ Weekly
- ◯ Once a month
- ◯ Once in 6 months
- ◯ Once a year
- ◯ Less than once a year or never

4. Have you played any other RPG/Open World games? *

Mark only one oval.

- ◯ Yes
- ◯ No

5. If you answered yes to the previous question, name other RPG/Open World games you have played.

_____

6. Have you ever played Mount & Blade II: Bannerlord? *

Mark only one oval.

- Yes
- No

M&B: Bannerlord

M&B: Bannerlord will be henceforth referred to has Bannerlord.

7. Have you played Bannerlord recently? *

Mark only one oval.

- Yes
- No

8. How many hours have you played Bannerlord (an approximation is enough) *

_____

Now you will be asked about your enjoyment regarding your experience with vanilla Bannerlord without the Plot Lords mod

Chose from the scale the number that best describes how you feel about the question where: 1 = Strongly Disagree and 7 = Strongly Agree

9. I think the game is fun *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

10. I feel bored while playing the game *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

11. If given the chance, I want to play this game again *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

12. I am likely to recommend this game to others *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

13. I enjoy playing the game *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

Now you will be asked about your enjoyment regarding your experience with the quests/missions presented in vanilla Bannerlord without the Plot Lords mod

A quest/mission is any form of task/tasks given by an NPC that the player follows in order to receive some sort of reward.Chose from the scale the number that best describes how you feel about the question where:
1 = Strongly
Disagree and 7 = Strongly Agree

14. About the lenght of the quests, do you think they were *

Mark only one oval.

- ◯ Just the right size
- ◯ A bit too long
- ◯ A bit too short
- ◯ Definitely too long
- ◯ Definitely too short
- ◯ Other: _____

15. About the cohesion/structure of the quests, do you think they were *

Mark only one oval.

- ◯ Structurally sound, each step being believable and making sense for it to be needed
- ◯ Most steps made sense, but a few of them didn't seem to fit in
- ◯ The steps to complete the quest were very messy and didn't make sence
- ◯ Other: _____

16. About the clarity of each step in a quest, do you think they were *

Mark only one oval.

- ◯ Clear on what I needed to accomplish
- ◯ Most of them were clear, but some of them were a bit confusing
- ◯ Just a few of them were clear and most of them were confusing
- ◯ None of them were clear on what I needed to accomplish to complete them
- ◯ Other: _____

17. I felt like the dialog of the quests (including quest descriptions) was clear *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

18. I felt like the dialog of the quests (including quest descriptions) was immersive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

19. I think the quests are fun *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

20. I feel bored while completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

21. If given the chance, I want to complete more quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

22. I enjoy completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

23. I feel like the quests are repetitive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

24. I feel like the quests always start and end the same way *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

25. I feel like the quests were made by humans *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

The Plot Lords mod

Now we're going to ask some questions regarding your general opinion on the quests added by the Plot Lords mod and your enjoyment while completing the quests.

26. How many hours have you played the mod "Plot Lords" for? *

Mark only one oval.

- ◯ less than 5 hours
- ◯ 5-10 hours
- ◯ 11-20 hours
- ◯ 21-30 hours
- ◯ more than 30 hours
- ◯ Prefer not to say

27. I realized that there were different quests in the game when playing with the Plot Lords mod *

Mark only one oval.

- ◯ Yes
- ◯ No

28. About the lenght of the quests, do you think they were *

Mark only one oval.

- ◯ Just the right size
- ◯ A bit too long
- ◯ A bit too short
- ◯ Definitely too long
- ◯ Definitely too short
- ◯ Other: _____

29. About the cohesion/structure of the quests, do you think they were *

Mark only one oval.

- ◯ Structurally sound, each step being believable and making sense for it to be needed
- ◯ Most steps made sense, but a few of them didn't seem to fit in
- ◯ The steps to complete the quest were very messy and didn't make sence
- ◯ Other: _____

30. About the clarity of each step in a quest, do you think they were *

Mark only one oval.

- ◯ Clear on what I needed to accomplish
- ◯ Most of them were clear, but some of them were a bit confusing
- ◯ Just a few of them were clear and most of them were confusing
- ◯ None of them were clear on what I needed to accomplish to complete them
- ◯ Other: _____

31. I felt like the dialog of the quests (including quest descriptions) was clear *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

32. I felt like the dialog of the quests (including quest descriptions) was immersive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

33. I think the quests are fun *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

34. I feel bored while completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

35. If given the chance, I want to complete more quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

36. I enjoy completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

37. I feel like the quests are repetitive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

38. I feel like the quests were made by humans *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

39. If you have any opinion or suggestion regarding the mod you would like to say feel free

Player Data

In this section we would just like to collect a bit of data stored by the mod for analytical purposes. The collected data it's a randomly generated player ID and the number of quests you have completed given by the mod. To share this data we just need for you to go to the mod folder, then the PlayerData folder, open the txt file and copy the contents to the spot bellow. If possible also write the id assigned to you bellow (should be the file name).

40. Given Player ID

41. Player Data file

Thank you

You have now completed the form. Thank you so much for your help, any further questions feel free to contact the mod developer Marco Cabral at: email: marco.cabral@tecnico.ulisboa.pt, discord: DezLezz#8307 or in the project's discord https://discord.gg/eysf5TNagj

A.2 User Test Questionnaire

Plot Lords User Test

This questionnaire is part of a Master's Thesis by Marco Cabral a student of Instituto Superior Técnico, University of Lisbon.

In this questionnaire, we seek to understand how the user perceives the new quests added by the Plot Lords mod in the game Mount & Blade II: Bannelord.

The experiment will take place as follows: you will start in an already created save file with a considerable army and will have to follow the instruction on the instruction sheet provided; two quests from the base game will first have to be completed followed by two quests added by the plot lords mod. Afterwards you will respond to the question from this questionnaire. The full experiment should take about 30 to 40 mins.

We would also like to remind you that:
- participation is voluntary and you can withdraw at any time;
- at no point throughout the study will you be identifiable, and individual results will not be disclosed;
- there are no physical or psychological hazards in your participation;
- you are free to ask any inquiry about the experiment at any time (email: marco.cabral@tecnico.ulisboa.pt | discord: DezLezz#8307), and you can also join the project's discord server if you have any further questions (https://discord.gg/34NXMeZR4Q).

By proceeding to the questionnaire you are giving your consent to the above conditions.

* Required

Demographics

1. Gender *

Mark only one oval.

- ◯ Female
- ◯ Male
- ◯ Prefer not to say
- ◯ Other: _____

2. Age *

Mark only one oval.

- ◯ Younger than 18 years old
- ◯ 18-25 years old
- ◯ 26-35 years old
- ◯ 36-45 years old
- ◯ older than 45 years old
- ◯ Prefer not to say

3. How often (approximately) do you currently play video games? *

Mark only one oval.

- ◯ Daily
- ◯ Weekly
- ◯ Once a month
- ◯ Once in 6 months
- ◯ Once a year
- ◯ Less than once a year or never

4. Have you played any other RPG/Open World games? *

Mark only one oval.

- ◯ Yes
- ◯ No

5. If you answered yes to the previous question, name other RPG/Open World games you have played.

_____

6. Did you ever play Mount & Blade II: Bannerlord before the experiment? *

Mark only one oval.

- ◯ Yes
- ◯ No

7. What about Mount & Blade: Warband? *

Mark only one oval.

- ◯ Yes
- ◯ No

First Two quests

Now you will be asked about your enjoyment regarding your experience with the first two quests you completed.
A quest/mission is any form of task/tasks given by an NPC that the player follows in order to receive some sort of reward.Chose from the scale the number that best describes how you feel about the question where: 1 = Strongly Disagree and 7 = Strongly Agree

8. About the lenght of the quests, do you think they were *

Mark only one oval.

- ◯ Definitely too short
- ◯ A bit too short
- ◯ Just the right size
- ◯ A bit too long
- ◯ Definitely too long
- ◯ Other: _____

9. About the cohesion/structure of the quests, do you think they were *

Mark only one oval.

- ◯ Structurally sound, each step being believable and making sense for it to be needed
- ◯ Most steps made sense, but a few of them didn't seem to fit in
- ◯ The steps to complete the quest were very messy and didn't make sence
- ◯ Other: _____

10. About the clarity of each step in a quest, do you think they were *

Mark only one oval.

- ◯ Clear on what I needed to accomplish
- ◯ Most of them were clear, but some of them were a bit confusing
- ◯ Just a few of them were clear and most of them were confusing
- ◯ None of them were clear on what I needed to accomplish to complete them
- ◯ Other: _____

11. I felt like the dialog of the quests (including quest descriptions) was clear *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

12. I felt like the dialog of the quests (including quest descriptions) was immersive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

13. I think the quests are fun *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

14. I feel bored while completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

15. If given the chance, I want to complete more quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

16. I enjoy completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree								Strongly Agree

17. I feel like the quests are repetitive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

18. I feel like the quests were made by humans *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

Last Two quests

Now we're going to ask some questions regarding your general opinion on the third and fourth quest you completed

19. I realized that the quests were different *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Other: _____

20. About the lenght of the quests, do you think they were *

Mark only one oval.

- ◯ Just the right size
- ◯ A bit too long
- ◯ A bit too short
- ◯ Definitely too long
- ◯ Definitely too short
- ◯ Other: _____

21. About the cohesion/structure of the quests, do you think they were *

Mark only one oval.

- ◯ Structurally sound, each step being believable and making sense for it to be needed
- ◯ Most steps made sense, but a few of them didn't seem to fit in
- ◯ The steps to complete the quest were very messy and didn't make sence
- ◯ Other: _____

22. About the clarity of each step in a quest, do you think they were *

Mark only one oval.

- ◯ Clear on what I needed to accomplish
- ◯ Most of them were clear, but some of them were a bit confusing
- ◯ Just a few of them were clear and most of them were confusing
- ◯ None of them were clear on what I needed to accomplish to complete them
- ◯ Other: _____

23. I felt like the dialog of the quests (including quest descriptions) was clear *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

24. I felt like the dialog of the quests (including quest descriptions) was immersive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

25. I think the quests are fun *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

26. I feel bored while completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

27. If given the chance, I want to complete more quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

28. I enjoy completing the quests *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

29. I feel like the quests are repetitive *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

30. I feel like the quests were made by humans *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	◯	◯	◯	◯	◯	◯	◯	Strongly Agree

Thank you

You have now completed the form. Thank you so much for your help, any further questions feel free to contact the mod developer Marco Cabral at: email: marco.cabral@tecnico.ulisboa.pt, discord: DezLezz#8307 or in the project's discord https://discord.gg/eysf5TNagj

31. If you have any opinion or suggestion regarding the mod you would like to say feel free