

**UNIVERSITAT
JAUME·I**

Development of a 3D asymmetric online multiplayer game in Godot

Daniel M. Pérez Maccari

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

June 28, 2024

Supervised by: José Vte. Martí Avilés



To my Family and Friends

ACKNOWLEDGMENTS

First of all, I would like to thank my Final Degree Work supervisor, José Vte. Martí Avilés, for his invaluable guidance and support throughout this project. His expertise and encouragement have been instrumental in the completion of my work.

Additionally I would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

I would also like to thank my parents and siblings for all their support and help over the years, allowing me to study what I was most passionate about and letting me dream.

Also thanks to all the people I met here, who have accompanied me over the years, and I hope they will continue to be part of my life in the future. I am especially grateful to those who shared unforgettable moments with me, from long nights of studying to breaks enjoying a good donut. Your support and friendship have made this experience truly special.

In addition, many thanks to the project organized by the UJI, *Pisos Solidarios*, which has given me the opportunity to stay in Castellón these past five years. It has shown me the kindness of people and how rewarding education can be.

Lastly, thank you, Margarita. Since I met you, the days fly by much faster. Your affection, patience, and time have meant the world to me. You have shown me just how beautiful life can be, and for that, I am forever grateful.

ABSTRACT

This document presents the project report of Daniel Miguel Pérez Maccari for the final project of the Video Games Design and Development degree, completed in collaboration with Margarita Gaya Hinsuk. The project details the development of a three-dimensional game using the open-source game engine Godot, with models created using Blender.

The game features real-time multiplayer communication set in a museum environment, employing an asymmetrical distribution of roles inspired by the popular title Among Us. Every aspect of the game, including code, models, UI assets, and rigs, was developed by the students. The animations used were sourced from a copyright-free site under the CC0 license.

This section of the project report focuses on the online architecture, communication, synchronization of computers, and the majority of the code implementation.

Keywords

Godot Engine, Asymmetrical Game, Real-Time Synchronization, PS1 Style, Security Guards, Vandals

CONTENTS

Contents	v
1 Introduction	1
1.1 Work Motivation	1
1.2 Objectives	3
1.3 Environment and Initial State	5
2 Planning and resources evaluation	9
2.1 Planning	9
2.2 Resource Evaluation	13
3 System Analysis and Design	23
3.1 Requirement Analysis	23
3.2 System Design	28
3.3 System Architecture	28
3.4 Interface Design	30
4 Work Development and Results	33
4.1 Game Design	33
4.2 Work Development	35
4.3 Programming	35
4.4 Results	47
5 Conclusions and Future Work	49
5.1 Conclusions	49
5.2 Future work	50
Bibliography	51
A Source code	53

INTRODUCTION

Contents

1.1	Work Motivation	1
1.2	Objectives	3
1.3	Environment and Initial State	5

This chapter outlines the planned activities and processes involved in the project's development. While stating the objectives of the work is essential, it is equally important to discuss the motivation driving the project, the initial concept, and the starting conditions.

The development of this project is rooted in a desire to explore innovative multiplayer gameplay mechanics and create a unique gaming experience. By leveraging our collective expertise and interests, we aim to address the growing demand for engaging and diverse multiplayer games. It also highlights the initial concepts and the foundational state from which we embarked on this creative journey.

1.1 Work Motivation

From a young age, one of the most fascinating aspects of video games has been their ability to bring people together. Regardless of age, these digital experiences unite a group of individuals in front of a screen to collaborate or compete, resulting in a highly enjoyable experience.

With increasing age and access to online connectivity through a computer, games such as *Minecraft* and *Club Penguin* (See figure 1.1) became popular choices for social interaction with friends. The inherent social nature of gaming has consistently sparked significant interest in networking and multiplayer game development.



Figure 1.1: Club Penguin (Disney Online Studios Canadá 2005)

Nonetheless, it seems like all the new technological advances keep pushing people further away. Social media platforms often foster superficial interactions, and certain video games tend to promote aggression and isolation. However, technology, particularly video games, also holds the potential to be an incredibly powerful tool for bringing people together.

Well-designed multiplayer games can facilitate genuine social interactions, allowing players to collaborate, compete, and connect in meaningful ways. By leveraging the interactive nature of games, it is possible to create experiences that not only entertain but also foster communication, teamwork, and a sense of community among players.

This potential makes video games a unique medium for bridging distances and enhancing social bonds, transforming them into a positive force in an increasingly digital world.

An investigation was conducted in this vast field of multiplayer video games, during which a relatively new genre gaining popularity was discovered. These are asymmetrical games, where players are divided into two or more teams, each with unique roles and abilities, competing for different objectives.

This unique distribution of power creates fascinating gameplay dynamics that encourage player interaction. Several subgenres have emerged within this genre. One of the games that inspired us the most was *Among Us* (See figure 1.2), due to its ability to create strategic and social tensions among participants. In this game roles are assigned secretly.

The primary objective in this game is to identify the "enemy" players, while they aim to eliminate the other players without being caught. This structure fosters intense interaction and strategic thinking, making it a compelling choice for the project.



Figure 1.2: Among Us (Innersloth, 2018)

This innovative play style and its increasing popularity lead us decide creating our own contribution to this genre and explore new possibilities of player interaction.

1.2 Objectives

The main objective of this project was to achieve a functional online connection between players and create a game experience that not only facilitates interaction but also fosters meaningful engagement and collaboration among them. The goal was to ensure seamless connectivity and smooth gameplay mechanics to enhance the overall multiplayer experience.

- **Explore and implement online video game architecture**

The first objective is to delve into the architecture of online video games, with a focus on understanding the intricacies of networking and multiplayer functionality.

By studying existing Godot Online Communication and implementing our own networking solutions in the engine, the aim is to gain a comprehensive under-

standing of the technical aspects involved in creating seamless online multiplayer experiences.

This includes implementing peer-to-peer communication to facilitate real-time synchronization, designing game logic that can handle multiple players efficiently, and ensuring that gameplay remains fair and balanced.

Special attention will be given to minimizing latency to provide a seamless multiplayer experience, allowing players to enjoy cooperation without technical issues.

- **Ensure Synchronization**

Using protocols to keep game states consistent and handle network delays effectively. The goal is to provide real-time updates and minimize discrepancies, ensuring a smooth and cohesive gameplay experience for all players.

This involves conducting comprehensive testing scenarios, including stress tests with multiple players, to identify and resolve potential issues.

The aim is to validate that all multiplayer features work correctly under various conditions. This testing process can be more complicated for this type of game compared to others, as it requires multiple participants.

- **Creating stylized 3D assets**

A key component of the project is the creation of stylized 3D assets to populate the game map. Our objective is to develop a cohesive and visually appealing art style that resembles the first 3D environments in videogames created for less capable machines.

The first step was to create a moodboard guideline for the art style to ensure its cohesiveness. Since the art was intended to have a retro aesthetic many references were gathered for this purpose. In the industry this type of visuals are denominated PSX (in honor to the graphics used in the consoles Play Station 1-2).

The model of the main character, game map and props, would also be following this same style, with a lower and a non-symmetric style. To keep the cohesion the textures of the models would need to have a lower resolution resembling a pixel style.

A PS1 shader and some post-processing effects will be added to ensure the visual aesthetics of the game remain true to the retro style.

- **Polishing and bringing an experience to the player**

Beyond the main development, one of the objectives is to focus on polishing and refining the overall player experience.

This includes optimizing gameplay mechanics, latency, optimizing network code, and improving the efficiency of data transfer. To ensure that players have a smooth and enjoyable experience from start to finish.

- **Emphasize Player Interaction and Engagement**

And finally emphasis on player interaction. Through features such as proximity voice chat and secret role assignments we aim to foster communication and teamwork.

The objective is to create a multiplayer experience that not only entertains but also encourages social interaction and player connections.

1.3 Environment and Initial State

In pursuit of this objective, an investigation into online video games took place, focusing on the current trends and successful elements within the genre.

This research highlighted the importance of an engaging environment and a well-defined initial state to captivate players from the outset. By analyzing various multiplayer games, it was found that an immersive setting combined with a clear starting point significantly enhances player interaction and enjoyment.

This understanding informed the design and development process, ensuring that the game's environment was meticulously crafted to be both visually appealing and functionally conducive to player engagement. Consequently, it was decided that the emerging new genre of **asymmetrical games** was particularly interesting, and we chose to focus our project within this innovative and dynamic category.

Some of the main leads in this genre are *Amond Us*, *Death By Daylight* (Figure 1.3) or *Back 4 Blood* [2]. Within this modern gaming genre, two primary groups have emerged, each with a distinct focus, centering the attention on different player emotions.

The method to archive this is mainly differentiated in how player roles are determined: the ability for players to choose their role versus roles being randomly assigned by the game.

The first group of games allows players to select their roles before the game starts. Giving the ability to select their preferred role aligned with their playstyle.

They enter the game with a clear understand of their objectives and abilities. This encourage strategic planning and specialization in an specific role. Players can discuss their roles and form strategies before the game begins, enhancing teamwork and cooperation.

On the contrary, games like *Among Us*, use random role assignment, adding unpredictability and surprise into each match. This ever-changing dynamic encourages adaptability and flexibility, with strategies needing to be adjusted to suit each unique



Figure 1.3: Dead By Daylight (by Behaviour Interactive)

match. Such games typically prioritize tension and excitement as players navigate the uncertainties of their roles and the actions of others.

Ultimately, the choice between allowing players to choose their roles or randomly assigning roles by the game has a significant impact on the overall gameplay experience and the emotions elicited from players. By exploring these two distinct approaches within the genre, a deeper understanding of player preferences and the design principles underlying successful multiplayer games can be attained.

Additionally, it's worth noting that a significant portion of games within this genre tend to adopt a horror theme. This preference for horror themes can be attributed to the inherent tension and suspense generated by the "killer" and "prey" dynamic commonly found in asymmetrical multiplayer games.

The horror genre lends itself well to creating an atmosphere of fear and anticipation, as players navigate environments fraught with danger and uncertainty. Furthermore, the asymmetrical nature of these games, where one player assumes the role of antagonist while others strive to survive.

However, we decided to diverge from the conventional horror concept to explore a different thematic direction. While the horror genre undeniably has its merits, we sought to create a game that would foster a unique social dynamic and broaden the appeal beyond just fear and suspense.

Inspired by the success of *Among Us* and its focus on deception and social interaction, the game was conceptualized to happen inside a museum during night time when there

is only security guards remaining. This setup allowed us to maintain the asymmetrical nature of gameplay while steering away from horror.

In our game, players are divided into two roles: vandals who aim to deface artwork in a museum and security guards tasked with apprehending them. This concept not only provides a fresh narrative but also encourages strategic thinking, teamwork, and deception in a non-horror setting.

By choosing this theme, the aim was to create a more accessible and engaging experience that appeals to a wider audience, emphasizing clever gameplay over fear-based thrills. And showcase the genre's potential for innovation and creativity across various themes and mechanics. By pushing the boundaries of what is conventionally expected from asymmetrical multiplayer games.

PLANNING AND RESOURCES EVALUATION

Contents

2.1	Planning	9
2.2	Resource Evaluation	13

In this chapter is going to be detailed the project’s planning process and outlines the necessary resources for its development.

2.1 Planning

Given the nature of the joint project, research and planning were essential components of a proper organization. The initial plan was drafted the design process started. Even though the fundamental concepts have remained unchanged, the project had minor modifications to adapt to design changes. Trello (see Figure 2.1) was used for planning purposes. All the task were later transferred to a Gantt chart (see Figure 2.2) The primary tasks include:

- **Research and Conceptualization - 20h**
Investigate and research various aspects of asymmetrical multiplayer game design, focusing on mechanics, player interactions, and the impact on player experience. Including how all this is implemented in Godot Engine and how to work with it.
- **Initial Design Phase - 20h**
Develop the Game Design Document (GDD) outlining the core concept of the

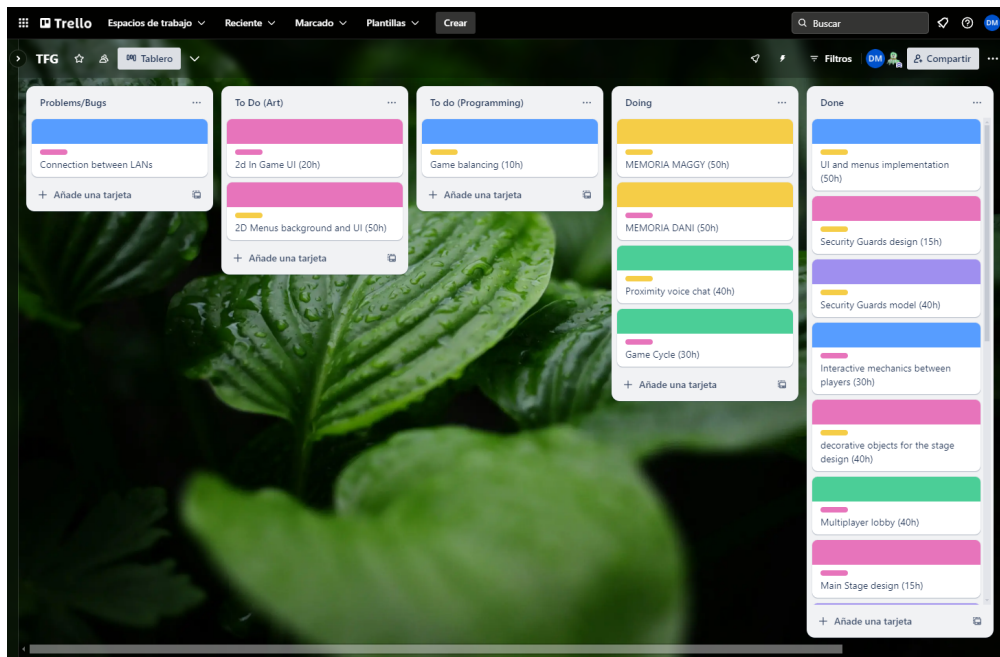


Figure 2.1: Trello Workspace

asymmetrical multiplayer game. Define game mechanics for each character role, objectives for each team, how to win the game, and the overall graphic style of the game.

- **3D Modeling and Asset Creation - 20h**

Most of the 3D assets were going to be made by the college Margarita. Using Blender (3D modeling software) to create models for characters, props and the main map. The player texture would be made by the student (See 4.3.6).

- **Programming - 150h**

This was going to be one of the parts where most time was invested in this report.

This project was started in Godot 3.8 but during development Godot 4.0 was realised with big optimizations and new functionalities in how multiplayer connections work. So it was decided to make a swift and start over in this new version of the engine.

It was a big step back in the process of having to remake some already working code. But it saved a lot of time in the long run.

- **Bug Fixing and Testing - 30h**

Bug fixing in video games is often an underrated aspect when planning ahead. In this project, due to its online nature, bug fixing would be particularly challenging. Initially, during the development phase the game was going to be tested on a single

machine. But it was needed to get a group of people to play-test it together and find potential error and design flows.

- **Memory and Presentation - 60h**

It includes the development of this report, the presentation and the preparation for the final assessment.

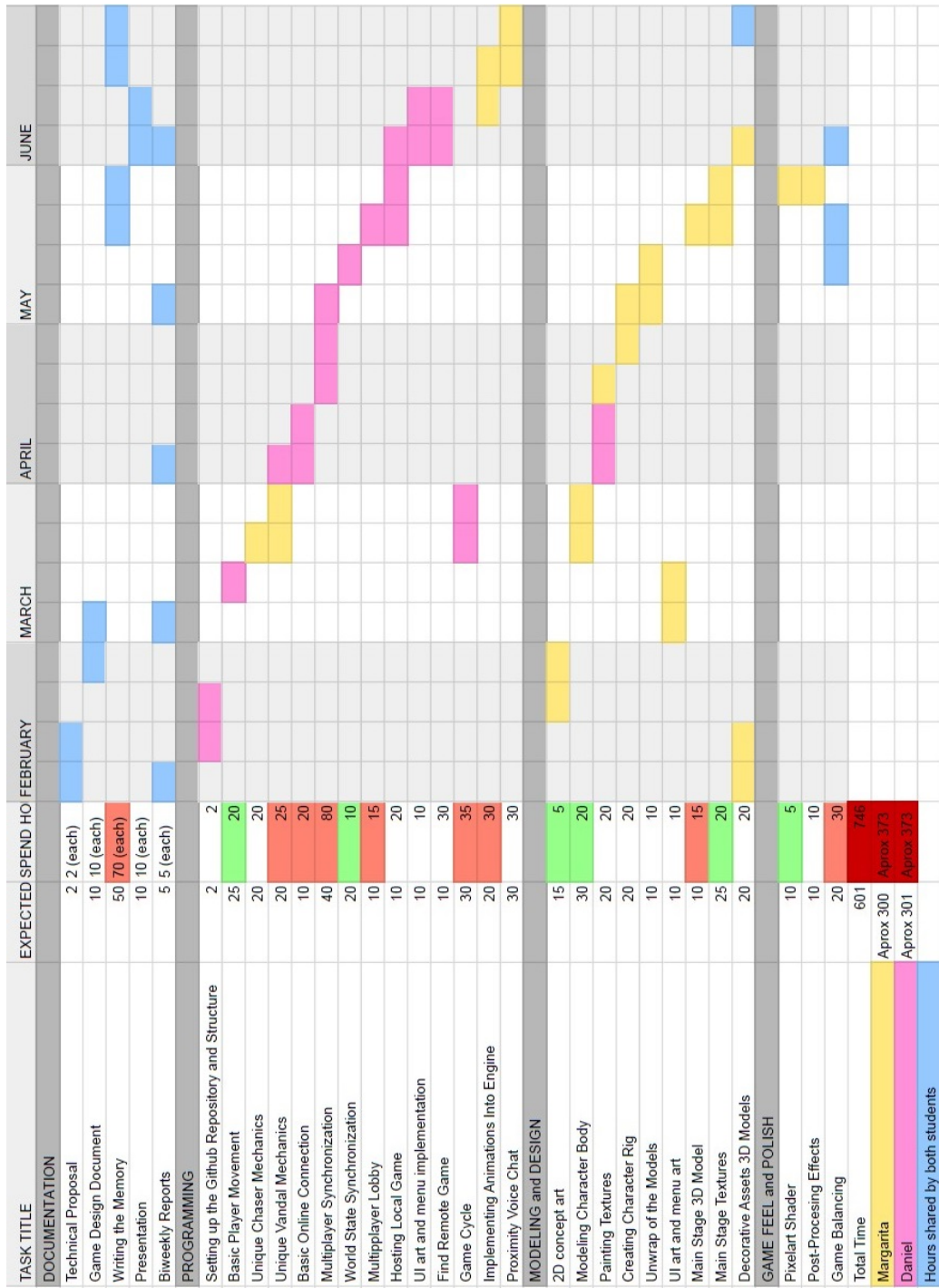


Figure 2.2: Gantt chart (made with Google Sheets)

2.2 Resource Evaluation

The goal of this project is to utilize open-source resources as much as possible. Both team members intentionally chose to use only open-source software throughout the development process. Most of the models were created by the team, but the few models sourced from the internet are also open source and appropriately credited at the end of this report.

This project was a collaborative effort between two people, eliminating the need for additional human resources. However, it's important to note that this is a demo, a complete working prototype of the product, but could eventually become a full game with extra features and polish. For a more extensive project, it would likely be necessary to involve more team members.

Here is an overview of all the hardware used by the team during the game's development. Including the authors and their colleges.

2.2.1 Daniel's Hardware

- **Laptop PC.** This laptop was bought second hand, its original purchase cost 1079,70€, but it was purchased for **750€** with one year of use.
 - Computer model: MSI MS-17C5
 - CPU: Intel Core i5-12600K
 - GPU: GeForce® GTX 1060 with 6GB GDDR5
 - RAM: Max 32GB, DDR4-2400, 2 Slots
 - Memory: 1TB HDD

to this original setup it was implemented an addition 500GB SSD card and another 32GB RAM to the free slot, assuming an additional cost of around **150€**

- **Other components.**
 - Mouse: KATAR PRO Wireless Gaming Mouse (**49.99€**)
 - Headphones: RAZER KRAKEN V2 X (**29.99€**)
 - Monitor: HP M24fe Monitor 24" (**127.00€**)
 - Keyboard: Trust GXT 833 THADO (**17.99€**)

2.2.2 Margarita's Hardware

- **Laptop PC.** The cost of this laptop and all of the components when it was purchased was **1598,86€**.

- Computer model: MSI GL75 Leopard 10SEK-040XES
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- GPU: GeForce® RTX 2060
- RAM: DDR IV 8GB*2 (3200MHz)
- Memory: 1TB HDD 256GB SSD

- **Other components.**

- Mouse: Corsair HARPOON PRO RGB (**174.24€**)
- Headphones: EDIFIER W820NB Bluetooth-Headset - Kabellose (**49.99€**)
- Secondary monitor: MSI PRO MP243W - Monitor 23.8" (**168.95€**)
- Ipad: Ipad Pro 2020 (**869€**)
- Apple pencil 2nd Gen (**119€**)

2.2.3 Software

All the software used is intended to be free, with a preference for open source whenever possible.

2.2.4 Game Engine

One of the doubts encountered was to decide which game engine would be used for this project, the main free game engines on the market are Unity, Unreal Engine, and Godot. Each of these engines has its own strengths and weaknesses, making the decision process crucial for the project's success.

UNITY

Unity is a powerful and versatile engine widely used in the game development industry. It offers several advantages:

- **Extensive Asset Store:** Unity's asset store provides a vast array of assets, plugins, and tools that can accelerate development.
- **Strong Community and Documentation:** Unity boasts a large community and extensive documentation, which can be invaluable for troubleshooting and learning.

However, Unity also has some drawbacks:

- **Resource-Intensive:** Unity is a heavy engine that requires substantial system resources, which can slow down development and testing, especially on less powerful hardware.
- **Workflow Complexity:** Despite having spent many hours working with Unity, I found its workflow to be cumbersome and less intuitive than other engines.

UNREAL ENGINE

Unreal Engine is renowned for its high-fidelity graphics and powerful capabilities:

- **Superior Graphics:** Unreal Engine excels in producing realistic graphics and is often used for AAA games and high-end visual projects.
- **Blueprint System:** Unreal's Blueprint visual scripting system is user-friendly and allows for rapid prototyping without deep programming knowledge.

Despite these strengths, Unreal Engine was not the ideal choice for this project due to the student's lack of familiarity and experience with it. Undertaking a large-scale project while simultaneously learning the intricacies of the engine would have been impractical.

GODOT

Godot emerged as the optimal choice for several reasons:

- **Lightweight and Efficient:** Godot is a lightweight engine that runs efficiently on a variety of hardware, making the development process smoother and faster.
- **3D Pipeline Optimization:** With the release of Godot 4.0, the 3D pipeline has been significantly optimized, ensuring high performance and quick rendering times.
- **Familiarity and Ease of Use:** Having prior experience with Godot from small personal projects, I found its workflow intuitive and enjoyable. Godot's GDScript, a Python-like language, is easy to learn and use.
- **Enhanced Online Capabilities:** Preliminary research into Godot's online capabilities showed promising results, and I was eager to explore and implement these features in the project.

- **Open Source Nature:** Being open source, Godot allows for greater flexibility and control over the project, without the concerns associated with licensing fees or restrictions.

In conclusion, while Unity and Unreal Engine have their distinct advantages, Godot was selected for this project due to its lightweight nature, improved 3D performance, familiarity, and promising online features. This combination of factors made Godot the most suitable engine to meet the project's requirements and goals effectively.

Godot is an open-source game engine known for its comprehensive toolset tailored for both 2D and 3D game development. One of its standout features is the fully integrated editor, providing developers with powerful capabilities to design and build game worlds, create intricate animations for characters and objects, and design intuitive user interfaces through a visual editor interface.

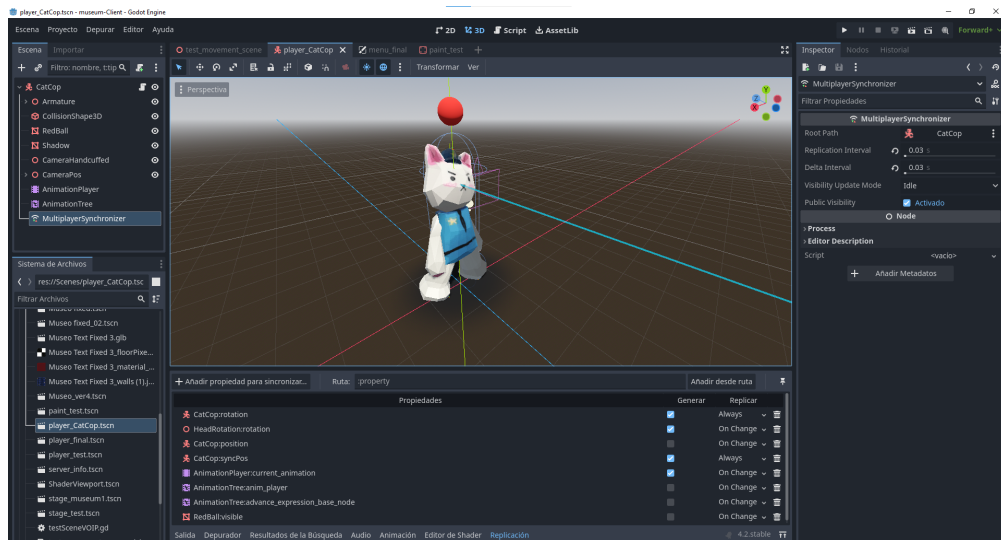


Figure 2.3: Godot Framework

Developers working with Godot benefit from its extensive cross-platform support, enabling seamless deployment across Windows, macOS, Linux, Android, iOS, and web platforms.

This versatility in deployment is complemented by the engine's flexibility in programming languages. While GDScript serves as the primary scripting language, developers also have the option to use C sharp or C++ for projects requiring additional performance or integration capabilities.

Godot's IDE includes a feature-rich code editor with syntax highlighting, auto-

completion, and debugging tools, enhancing productivity and facilitating rapid iteration during development. Moreover, the engine's open-source nature encourages community collaboration and continuous improvement, ensuring ongoing enhancements and support for a diverse range of game development needs.

In summary, Godot stands out as a robust and accessible game engine choice, offering a blend of powerful features, flexible programming options, and extensive platform compatibility suitable for both indie developers and professional studios alike.

2.2.5 Graphical Comparison

The following table [9] compares key features of the previous game engines: Unity, Unreal Engine, and Godot. It evaluates aspects such as source availability, usability, platform support, programming languages, and export capabilities.

Each engine is assessed based on its suitability for different development needs, ranging from 2D and 3D game creation to platform-specific deployment and mobile app integration. This comparison aims to assist developers in choosing the most appropriate engine for their projects.




-  The engine has robust support for this feature.
-  The engine does not support this feature.
-  The engine can potentially support this feature, but it can be in form of beta packages, or some extra effort might be involved.

Figure 2.4: table legend

2.2.6 Other Software

Other important software tools used in the project where:

- **Windows 11 Pro**

The operating system used. The basic digital licence (**9.90€**)

- **Blender 4.1** [12]

When deciding on the tool for creating 3D models for this project, the options considered were Blender and Maya. Both tools have their own advantages and considerations and the experience the students had with both of them were quite similar.


	Unity	Unreal	Godot
Engine			
Source available	 (1)		
Free to use	 (2)	 (2)	
Learning curve	 (3)	 (3)	 (3)
3D			 (4)
2D		 (6)	
Editor			
Windows			
Mac			
Linux	 (7)	 (8)	
Coding			
Language	C#, C++	Blueprints, C++	GDScript, C#, C++ (9)
Visual scripting			
Built-in editor		 (10)	
Export			
Windows			
Mac			
Linux			
Android			
iOS			
Web			
XBox			 (11)
Playstation			 (11)
Nintendo Switch			 (11)

Figure 2.5: Table Comparing Important Points of Each Engine

Blender was chosen for this project due to its cost-effectiveness, powerful features, supportive community, and the opportunity to build proficiency with an increasingly popular and capable open-source tool. This decision aligns with the project's goals of minimizing costs and preparing for future production needs. **(Open Source)**

- **Mixamo**
A web-based service provided by Adobe, offers a vast collection of high-quality 3D character models and animations. All the animations used in this project were taken from this site and applied to the skeleton created by the student. **(Free)**
- **Krita**
This tool was used to paint over the player's model unwrap to create a texture then used inside the engine. **(Free)**
- **GitHub**
The version control system used for storing the different stages of the game. **(Free)**
- **GitKraken** (free version)
For a good visualization of the project's version control and better coordination between the members this tool to manage the Git actions was used. (see Figure 2.6)
- **Trello**
A web-based project management tool that uses cards to organize tasks. It was used to assign tasks, organize, and track progress visually. **(Free)**
- **Overleaf**
An online tool useful for the development of the thesis in LaTeX. **(Free)**

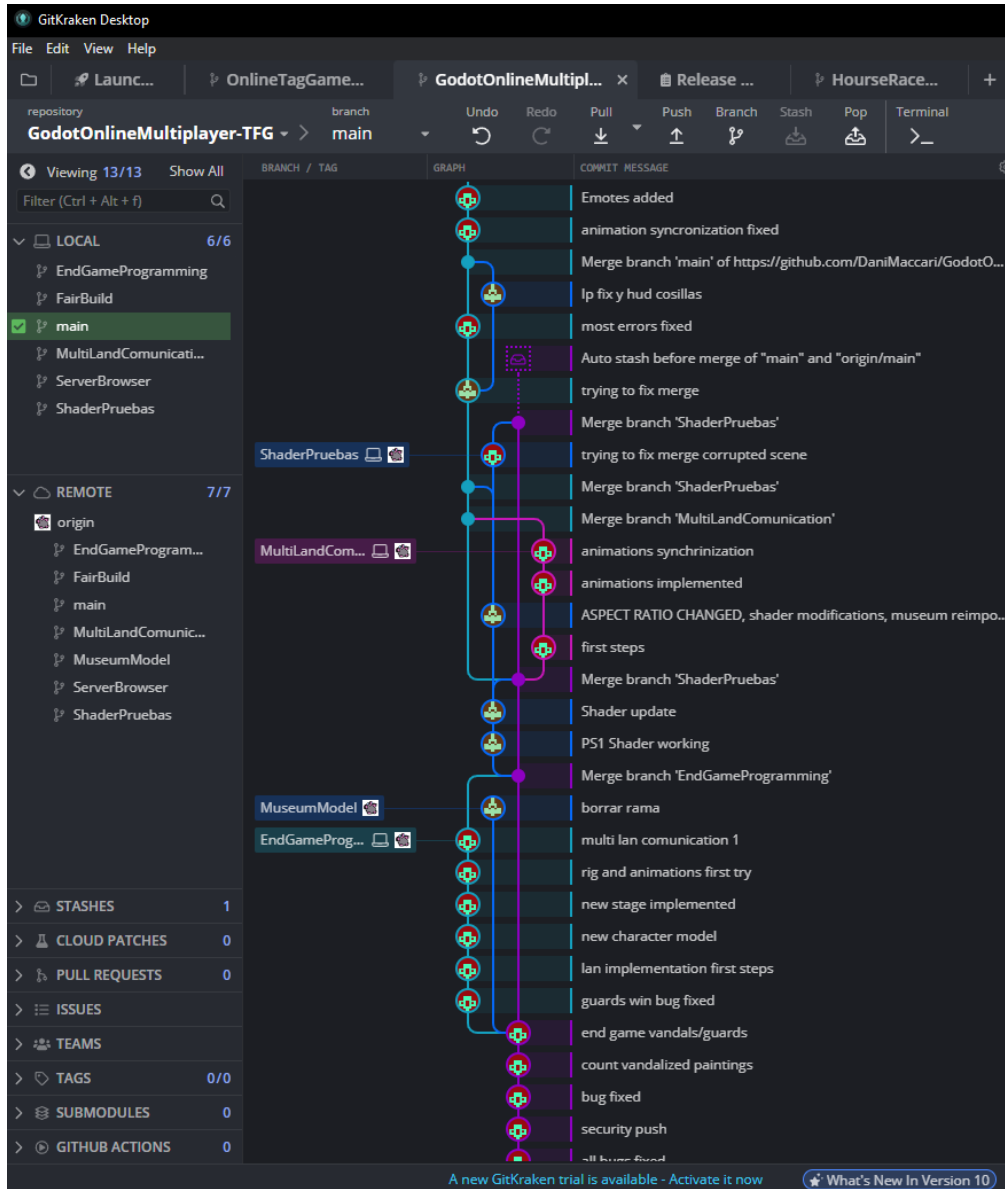


Figure 2.6: GitKraken, project commit timeline

2.2.7 Other Resources

If this project was developed by a company (or self-employed), the cost of human resources would be measured in person/month. If the project costs approximately 650 hours with two person working in it and the workweek is around 37,5 hours, the project will be completed in 8.66 weeks.

And a month consists of approximately 4,35 weeks, this would be 2 months work. The average salary of a Junior developer in Spain is €1990.16/month [10], that is a cost of around €2400. In conclusion, human resources would be $2(\text{months}) \times €2.400 \times 2(\text{people}) = €9600$.

In Castellón de la Plana, the rent of a co-working local is around €220 per month. (€440 for 2 months).

Adding electricity, water and internet connection would increase the monthly cost about €120. A total of extra €240.

On the other hand, the software is mostly free. The hardware and software specified before costs around €4104.97. All the hardware has an estimated useful life of around 5 years. So for the 2-month period it would be approximately €137 equipment.

In conclusion, the total development cost would be $€4800 \times 2 + €440 + €240 + €137 = €10417$. If the contingency costs are taken into account, which are typically 10% of the total cost, it would be $€10417 + €1041.7 = €11458.7$.

A 15% profit margin is taken into account. The total would add up to €13177.5 .

Considering 12% IVA stipulated for selling online media such as videogames. The total amount would be €14758.8 .

We have a sales target of 2500 copies. Therefore, the price at which the game should be launched would be €5.9 per copy of the game.

SYSTEM ANALYSIS AND DESIGN

Contents

3.1	Requirement Analysis	23
3.2	System Design	28
3.3	System Architecture	28
3.4	Interface Design	30

This chapter presents the requirements analysis, design and architecture of the proposed work, as well as, where appropriate, its interface design.

3.1 Requirement Analysis

3.1.1 Functional Requirements

The following tables describe the functional requirements as independent actions, each with its respective reaction in the game. These actions are integral to the game’s mechanics, defining how different elements interact within the system.

By breaking down the requirements into discrete actions and reactions, it becomes easier to understand and implement the necessary functionalities for a seamless gaming experience. Each action triggers a specific response, ensuring that the game logic remains consistent and predictable.

These interactions are meticulously collected and organized in the graph, providing a visual representation of the workflow and dependencies (see Figure 3.1). This graphical

depiction aids in identifying potential bottlenecks, optimizing performance, and ensuring that all requirements are met systematically.

The proposed system should allow the purchase of discs using credits. The functional requirements related to the query and purchase of credits are shown in Tables 3.1 and 3.2.

Input:	A player can create a new party
Output:	A new lobby is created to
Each player can decide to host a new game, creating a party where other players can join.	

Table 3.1: Functional requirement «CRED1. Create party»

Input:	Search for existing games
Output:	A list with all current games
The user decides to search for existing lobbies, it gets a list of existing parties that hasn't started the game. each lobby has a different name and the number of current players in it.	

Table 3.2: Functional requirement «CRED2. Join Party»

Input:	The host player can start a match
Output:	Match starts
The player who is hosting the game has the ability to start the game. When this button is clicked the roles are assigned and the match starts for all the players that joined this party.	

Table 3.3: Functional requirement «CRED3. Start Game»

Input: WASD keys

Output: Move throught the game map

Once the match starts, each player can move freely through the game world. Navigate using arrow keys for movement in any direction. The key direction is relative to the camera position. "W key" will move forward, "S" backwards, "A" left and "D" right..

Table 3.4: Functional requirement «CRED3. Basic movement»

Input: Mouse Movement Input

Output: Look around the game world

The player has the ability of using the mouse input to move the camera. The key direction is relative to the camera position. Forward will be the z axe in the camera coordinates.

Table 3.5: Functional requirement «CRED4. Camera Movement»

Input: Left mouse button

Output: Handcuff a player

When the player is looking at another player, they have the possibility to handcuff them using the left mouse button. Once this action is used the player can no longer use it.

Table 3.6: Functional requirement «CRED5. Handcuff Another Player»

Input: E key

Output: Paint over a painting

The players that are given the vandal role at the start of the game, can click the E key on the keyboard when standing in front of a painting and it will be vandalized. The players movement will be blocked a short time while the spray animation is active.

Table 3.7: Functional requirement «CRED6. Vandalize A Painting»

Input: 1 key

Output: Wave Animation

The player can hold the 1 key to do a waving animation.

Table 3.8: Functional requirement «CRED6. Dance 1»

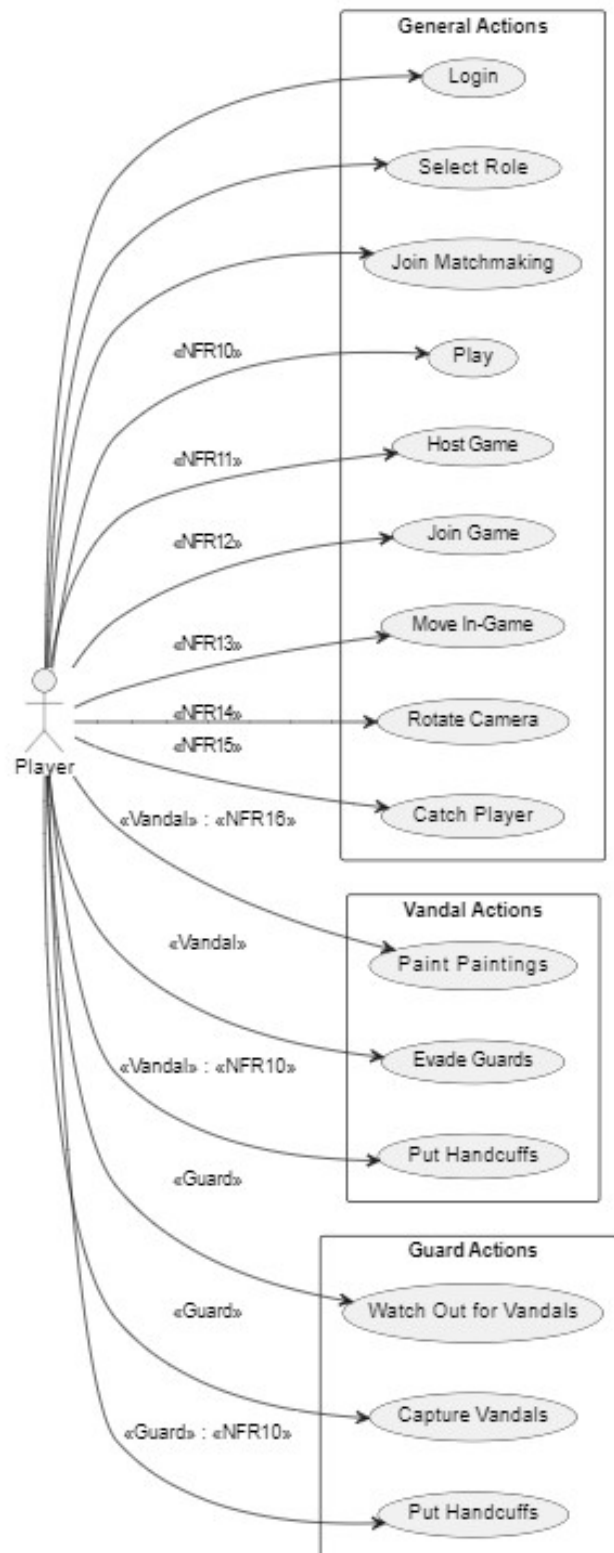


Figure 3.1: Case of Use

Input:	2 key
Output:	Point Animation
The player can hold the 1 key to do a pointing animation.	

Table 3.9: Functional requirement «CRED6. Dance 2»

Input:	3 key
Output:	Dance Animation
The player can hold the 1 key to do a dancing animation.	

Table 3.10: Functional requirement «CRED6. Dance 1»

3.1.2 Non-functional Requirements

Non-functional requirements specify conditions that a system must meet beyond its core functionality, such as performance, safety, or reliability, as defined by industry standards [13]. They describe how a system should behave or perform. These requirements guide the system's design and implementation to ensure it meets specified criteria, which will be detailed in the system architecture.

- **NFR01.** The game is playable on the Windows PC platform, ensuring broad accessibility for users of this operating system.
- **NFR02.** The game has robust network capabilities, facilitating multiplayer functionality and online interactions essential for its gameplay mechanics.
- **NFR03.** The artistic style emulates the nostalgic aesthetics of old PlayStation games, featuring low-poly models, vibrant colors, and stylized textures to evoke a sense of retro gaming charm.
- **NFR04.** The user interface (UI) is designed to be minimalist and unobtrusive, prioritizing a clear view of the game world and essential player information, such as character health and mission progress.
- **NFR05.** The controls and gameplay mechanics are intuitively designed, ensuring ease of learning and smooth gameplay experience for players of all skill levels.
- **NFR06.** The game maintains a cohesive ambience and atmosphere throughout, aligning with its distinctive artistic style and thematic elements to enhance immersion and storytelling.

- **NFR07.** The game features a 3D art style that is simplistic yet visually appealing, using stylized visuals rather than realistic graphics to create a unique and charming game world.
- **NFR08.** The game is structured around short-duration matches or gameplay sessions, catering to modern gaming preferences for quick, engaging experiences that fit into varying time constraints.

3.2 System Design

The best way to present the logical design of the game is through a flow chart. (see Figure 3.2)

3.3 System Architecture

This video game is developed using the Godot engine, version 4.2. The minimum system requirements [4] for running games with this engine are as follows:

- Operating System: Windows 7/8/10 (64-bit), macOS 10.11+, or Linux Ubuntu 16.04+
- Processor: Intel Core 2 Duo E8400
- Graphics: NVIDIA GeForce 6200 or higher

These requirements are based on information from the Godot engine documentation.

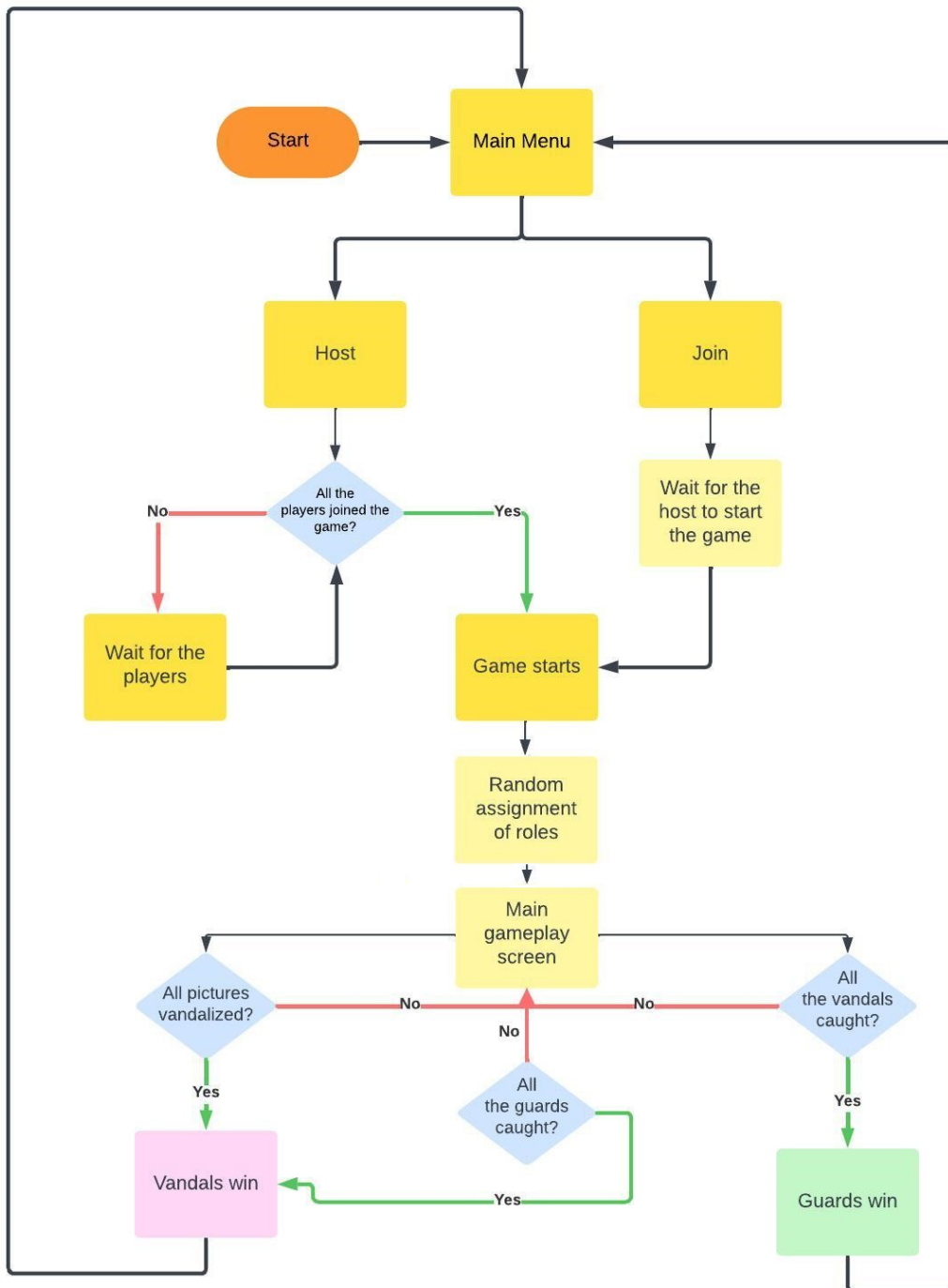


Figure 3.2: Game Flow Chart

3.4 Interface Design

The interface design in this game emphasizes simplicity and functionality to enhance the player experience. During gameplay, important status indicators are strategically placed for quick reference. For instance, players can easily check the bottom left corner of the screen to see if they have access to handcuffs, which are essential for capturing opponents.

In addition to these real-time gameplay elements, the game begins with initial screens that promptly inform each player of their assigned role as either a security guard or a vandal. This early notification sets the tone for their objectives and responsibilities throughout the match, ensuring clarity from the start.

At the conclusion of each game session, a dedicated end-game screen provides a comprehensive summary of how the match unfolded. This screen details whether the vandals successfully achieved their goals by painting designated artworks or if the guards effectively thwarted their efforts by apprehending them. This feedback not only informs players of the outcome but also reinforces the game's competitive dynamics.

Due to the multiplayer nature of the game, there is deliberately no pause screen available. Implementing such a feature would disrupt the flow of gameplay for all participants simultaneously. This decision ensures that the game maintains its competitive

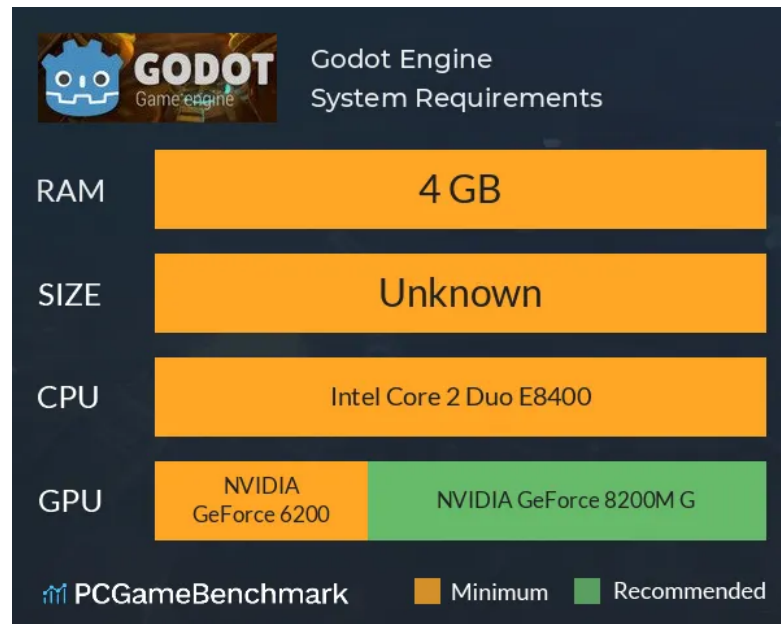


Figure 3.3: Steam Minimum Godot Requirements

integrity and allows all players to remain engaged without interruptions.

Overall, the interface features in this game are designed to be intuitive and informative, contributing to a seamless multiplayer experience while maintaining gameplay integrity and clarity throughout.

WORK DEVELOPMENT AND RESULTS

Contents

4.1	Game Design	33
4.2	Work Development	35
4.3	Programming	35
4.4	Results	47

This chapter presents a comprehensive overview of the project’s development from its inception to completion. It evaluates the outcomes achieved and discusses the evolution of original concepts throughout the process, as well as addressing various issues encountered along the way.

4.1 Game Design

During the brainstorming process, several game models were considered. The goal was to achieve gameplay where all players felt they had an active role and to minimize downtime. Therefore, it was decided to introduce player-to-player interactions, providing them with tools to create interesting situations.

Various game models were explored to ensure dynamic and engaging gameplay [11]. The emphasis was on creating an environment where each player’s actions influenced the overall experience. This approach aimed to foster collaboration, strategic thinking, and meaningful player interactions.

The implementation focused on mechanics that encouraged players to engage with each other actively. This included systems for communication, cooperation, and competition among players, ensuring that every moment in the game contributed to its overall excitement and depth.

By enabling players to influence and shape the game environment through their interactions, the design aimed to create a dynamic narrative driven by player agency. This approach not only enhanced the game's replayability but also encouraged creativity and emergent gameplay scenarios.

The game is a multiplayer experience set in a museum environment, where players assume dual roles as both guards and vandals. At the start of the game, each player is randomly assigned one of these roles. Vandals' objective is to covertly paint artworks displayed in the museum, evading detection by guards who must discern their identities from a crowd of identical-looking characters.

The museum setting is richly detailed, featuring various exhibits and valuable artworks ripe for artistic alteration. Vandals must strategically choose their targets and plan their movements to avoid the limited number of guards patrolling the halls. Meanwhile, guards must rely on observational skills and deductive reasoning to apprehend vandals without mistakenly targeting innocent players.

Win conditions are clear-cut: Vandals achieve victory by successfully painting all designated artworks before being captured, showcasing their skill in stealth and creativity. Conversely, guards secure a win by identifying and handcuffing all vandals before they can complete their artistic endeavors, demonstrating keen judgment and teamwork.

This mechanic of capturing other players by handcuffing them and limiting their movement serves as an alternative to the common use of violent objects. Additionally, it allows us to keep these captured players engaged in the game as spectators, enabling them to share information about what they observe. This ensures that being captured doesn't mean the game is over for them, maintaining their involvement and strategic impact throughout the match.

The gameplay thrives on suspense and strategic depth, encouraging players to adapt their tactics dynamically based on unfolding events and the evolving interactions between vandals and guards. Each match promises a unique and engaging experience, driven by the intricate balance between deception, observation, and decisive action in the pursuit of victory.

4.2 Work Development

It will be explained the development in chronological order to maintain continuity in the project. This structured approach allows for a clear understanding of how each component builds upon the previous one. As the project progresses, you'll notice that many of the mechanics introduced early on serve as foundational elements for those introduced later. This sequential presentation ensures a coherent expansion of the game and leads to a clear understanding of each new feature.

The game is set in a museum during nighttime when there are no visitors. Only the security guards, also known as CatCops, remain, whom our players will impersonate. Among them, vandals hide in secrecy, pretending to be one of the guards. These vandals plan to graffiti and paint over the paintings. The guards' mission is to identify and arrest the graffiti artists before they manage to deface all the museum's paintings. The game will end once the vandals complete their plan or are captured.

To execute this strategy, players are afforded the opportunity to traverse the museum's halls, carefully scrutinizing each painting while engaging in discussions with their fellow players to piece together any clues. Every participant, whether they be a guard or a vandal, will be equipped with a pair of handcuffs, a crucial tool in their arsenal. These handcuffs can be strategically deployed on a chosen player, effectively immobilizing them for the duration of the game. The pivotal decision of which player to restrain with handcuffs holds significant weight, as it can ultimately decide the outcome of the entire match.

However, vandals possess an additional layer of complexity. In addition to their standard abilities, they are endowed with a unique mechanic that grants them the ability to deface paintings with graffiti. If they manage to vandalize all of them before being caught, they will win the game.

4.3 Programming

4.3.1 Starting Over

When the project started, despite the improved optimization for 3D projects promised by the release of Godot 4.0, the decision was made to utilize Godot 3.8, its predecessor. This choice stemmed from the limited availability of information and tutorials for the new 4.0 version at the time. However, as development unfolded, significant challenges arose, particularly in getting to work the real-time peer-to-peer communication.

Efforts to address these challenges consumed considerable time, spanning several weeks without yielding viable solutions. Faced with this setback, a decision was reached

to explore alternative approaches, prompting research into the capabilities of Godot's 4.0 version. It was discovered that many of the encountered issues had been addressed in this updated iteration, along with the introduction of new features aimed at streamlining the networking system within the engine.

Following extensive deliberations, the challenging yet essential decision was made to discard the current progress and begin with Godot 4.2. Despite the setback this decision entailed, it promised a more streamlined development process. Fortunately, the initial tests were conducted very early on, so we didn't encounter any timing issues.

4.3.2 Player Movement

One of the early implementations and foundations for the player is the character movement. A 3D first-person movement [7] served as the starting point. Thanks to previous work with the Godot Engine and familiarity with its GDScript language, which is based on Python, it wasn't difficult to achieve a movement system that felt satisfactory to use. Introducing this first-person camera movement was easier than expected, given the robust support and documentation available.

By implementing this movement system early on, a solid foundation was established for the player characters. This primary movement system enabled a natural and intuitive control scheme for the players, which is crucial for ensuring an immersive gaming experience. Furthermore, having the main character movement implemented allowed future mechanics and gameplay elements to be built upon this foundation smoothly.

```
1 func _physics_process(delta):
2
3
4     if $MultiplayerSynchronizer.get_multiplayer_authority() ==
5         multiplayer.get_unique_id():
6
7         isLabel.text = ("is_handcuffed_" + str(isHandcuffed) )
8         hasLabel.text = ("has_handcuffs_" + str(hasHandcuffs) )
9         #$Camera3D/Control/HandcuffsTexture.visible = hasHandcuffs
10
11        if isHandcuffed || !canMove:
12            return
13
14        headRotation.rotation.x = -camera.rotation.x #head movement
15
16        syncPos = global_position
17
18        # Get the input direction and handle the movement/deceleration.
19        var input_dir =
```

```
20     Input.get_vector("ui_left", "ui_right", "ui_up", "ui_down")
21     var direction =
22     (transform.basis * Vector3(input_dir.x, 0, input_dir.y)).normalized()
23
24     if input_dir.y < 0 || input_dir.x != 0:
25         playAnim.rpc("Walk")
26     elif input_dir.y > 0:
27         playAnim.rpc("WalkBack")
28     elif !emote:
29         playAnim.rpc("Idle")
30
31     if direction: #is moving
32         velocity.x = direction.x * SPEED
33         velocity.z = direction.z * SPEED
34     else:
35         velocity.x = move_toward(velocity.x, 0, SPEED)
36         velocity.z = move_toward(velocity.z, 0, SPEED)
37
38     move_and_slide()
39
40     #handle interpolation to minimize data
41     else:
42         syncPos = global_position.lerp(syncPos, 0.5)
```

4.3.3 Initial Player Synchronization

Due to this progress, a preliminary test was conducted to assess the functionality of multiplayer synchronization in the latest version update. After carefully reviewing the official documentation available about this online feature [1], a comprehensive understanding was gained of how the engine manages shared scenes between different players.

This preliminary test was crucial in determining the effectiveness of the multiplayer synchronization features provided by Godot. The documentation had detailed insights into the intricacies of scene replication, node synchronization, and network communication protocols supported by the engine. Also provided a demo project and a step by step guide to follow its creating. Were all the basic capabilities were explained.

In this preliminary version, testing was conducted locally with multiple instances of the same project running on a single computer. A button was provided to instantiate a new *character* object onto the scene. Once instantiated, the character could be moved using the arrow keys. The movement exhibited some lag, and synchronization skipped numerous frames, but the core functionality was operational.

This local testing setup allowed for an initial assessment of the multiplayer system's basic mechanics. Despite the performance issues, the foundational aspects were success-

fully demonstrated. Overall, the preliminary version established a working prototype of the multiplayer system, highlighting both its potential and the challenges to be addressed in subsequent iterations.

After that preliminary test, it was decided to start working in a proper multiplayer communication [5]. Establishing one of the players as the creator or host of the game. This player would be the one connecting the other players, acting as a server that also is involved in the game.

The concept involved the first player creating a new scene (the map) and instantiating a player character for themselves. Subsequent players would then search for an existing scene and spawn a new character within it, leveraging the local machine's IP address. Once it was decided to implement a more complex synchronization, it was time to start using the dedicated nodes for this work. The new 4.2 update implemented to the engine a relatively easy way to manage the sensed information to mince bandwidth cost. This unique node is called *MultiplayerSynchronizer* and it needed the use of a *MultiplayerSpawner* in the scene intended to work.

Each object that needed to share and synchronize information was equipped with a this child node, which transmitted the selected character data to the host. The host subsequently relayed all player positions to connected players. Every object instantiated in this manner within the scene listened for incoming updates and adjusted accordingly upon receiving new data.

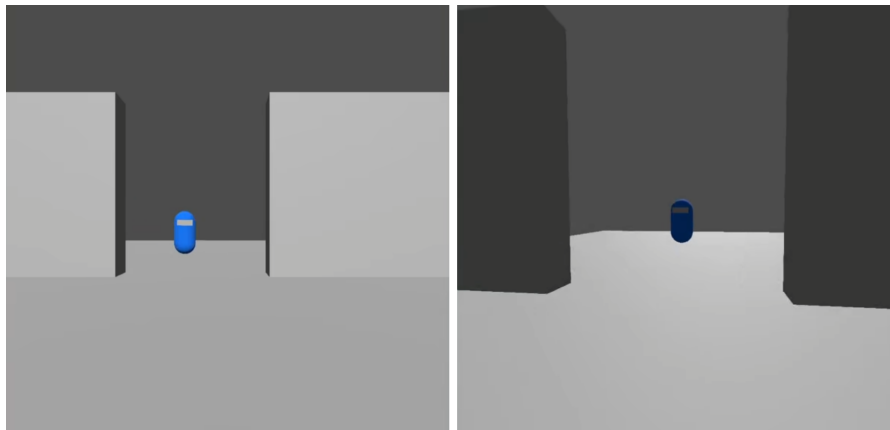


Figure 4.1: project state after successful Player Synchronization

4.3.4 Peer-to-Peer Application

Peer-to-Peer (P2P) Connection [14] is a network model where individual devices (referred to as "peers") act as both clients and servers, enabling direct sharing of resources between them without the need for a centralized server. In such a network, each node

has the capability to send, receive, and share data and resources with other nodes, facilitating decentralization and equitable distribution of load among participants.

Godot provides a set of functions within the engine that facilitate the implementation of this system, up to a certain extent. The following steps are required to achieve this:

1. A user starts by choosing the server option, providing the IP address to be distributed to others for connection.
2. Other users receive a list of active hosts and choose to join one of them.
3. Once a sufficient number of players are present, the button to start the game session becomes active.

These steps outline the process for setting up a peer-to-peer connection using Godot, leveraging its built-in functionalities to enable multiplayer interactions without relying on a central server.

```
1 func _on_host_pressed():
2     HostGame()
3     playerNick = $LineEdit.text
4     SendplayerInformation(playerNick, multiplayer.get_unique_id())
5     $ServerBrowser.SetUpBroadCast(playerNick)
6     pass
7
8 func HostGame():
9     peer = ENetMultiplayerPeer.new()
10    var error = peer.create_server(PORT, max_clients)
11    if error != OK:
12        print("couldnt_host:_", error)
13        return
14
15    peer.get_host().compress(ENetConnection.COMPRESS_RANGE_CODER)
16
17    multiplayer.set_multiplayer_peer(peer)
18    print("waiting_for_players")
19
20 func _on_join_pressed():
21    JoinByIp(ADDRESS)
22    pass # Replace with function body.
23
24 func JoinByIp(ip):
25    peer = ENetMultiplayerPeer.new()
```

```
26 | peer.create_client(ip, PORT)
27 | peer.get_host().compress(ENetConnection.COMPRESS_RANGE_CODER)
28 | multiplayer.set_multiplayer_peer(peer)
```

4.3.5 Player Interaction

Video games are a lot of the time described as a virtual interactive environment. Without this key feature there would just be an exploitable environment with nothing else.

Typically, in video games with a first-person view, as in this one, a mark known as a cross-hair is drawn in the center of the screen. This cross-hair serves as an indicator for the player, pinpointing the exact point that will activate interactions.

The presence of the cross-hair is crucial for providing a precise aiming mechanism, ensuring that players can accurately target objects or areas within the 3D game environment. By aligning the cross-hair with interactive elements, players can effectively engage with the game world.

To enable this functionality, a ray-cast is aligned with the player's camera, ensuring that the cross-hair position overlaps with the ray-cast direction. This alignment allows the game to accurately determine which objects or elements the player is aiming at, facilitating precise interactions within the game environment [8].

This ray-cast implementation and use to get the actions is discussed in more detail in the colleague's report.



Figure 4.2: Game Screenshot

Once this was completed (handcuffing and vandalizing the paintings), it became necessary to synchronize these actions across different players to ensure a seamless multiplayer experience.

The main concept involved sending a signal when a player handcuffed another player [3], prompting the handcuffed player to activate a script that would immobilize them for the remainder of the game.

This was implemented by identifying the name of the object that collided with the raycast, which would be the ID of the handcuffed player, and sending an RPC call to this IP to execute a `GetHandcuffed` function.

Initially, this appeared to work correctly; however, further testing revealed that the functionality only operated as intended when the host player was being arrested. In all other cases, the system failed. Diagnosing this issue required a considerable amount of time, as it was the first of such problems encountered in the project. The root cause of the problem was traced to **RPC Labels**.

In Godot, these labels are critical for identifying an RPC call and determining how it will be accessed and executed. The available labels are:

- `any_peer`
- `authority`
- `call_local`
- `call_remote`
- `reliable`
- `unreliable`

For our function, we needed it to be callable by any peer external to the current game instance. Therefore, it was essential to use the “`any_peer`” and “`call_remote`” labels. This configuration ensured that the code could be executed by any player, regardless of their location in the network relative to the game instance.

Through this approach, the synchronization issue was resolved, allowing all players to properly execute the handcuffing action, thus enhancing the overall multiplayer experience. This solution underscored the importance of understanding and correctly applying RPC labels within Godot to manage networked multiplayer interactions effectively.

4.3.6 Character Textures

Textures play a crucial role in achieving the authentic PS1 aesthetic for game assets. Given that these models are deliberately designed with very low polygon counts, textures are responsible for conveying details, colors, and shading.

Ensuring that these textures look good on the models requires precise UV mapping to accurately connect the model's polygons with the texture. Once my project partner provided the UV unwrap, the texturing process could commence.

These are the steps that were followed:

- The unwrap was loaded onto krita, that image was locked up in a base layer.
- Another layer was created above, that would be the texture layer for the cat texture.
- Taking into account the different parts and margins of the unwrap, the texture layer was painted above.



Figure 4.3: Character Model With Texture

4.3.7 Role selection

At the beginning of each game, the roles are assigned randomly to ensure a balanced and unpredictable gameplay experience.

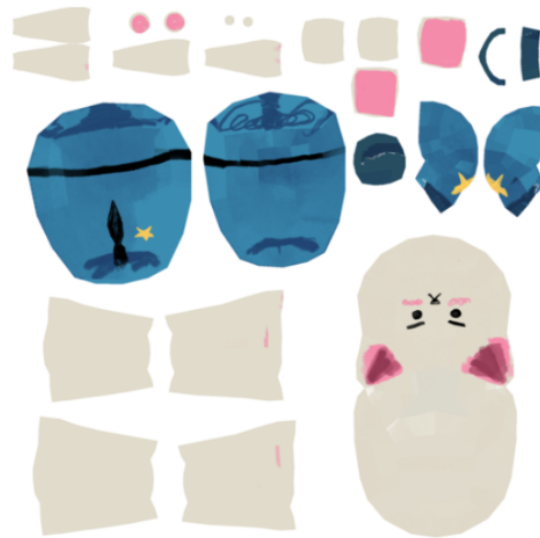


Figure 4.4: Character Texture

Depending on the total number of players, a different number of vandals will be selected to maintain the game's difficulty and excitement. Typically, the ratio of vandals to security guards is adjusted to create a challenging environment for both teams.

The role assignment process is managed by the host player at the `GameManager` script `??`, the function `selectBadGuys()` designed to allocate roles fairly and randomly.

Once the roles are assigned, players are notified of their roles through a discreet text message displayed on their screens. This notification system ensures that the role assignment remains confidential, with only each player knowing their own role.

The secrecy of role assignments adds a layer of intrigue and strategy to the game. Since only the individual players know their roles, both vandals and security guards must carefully navigate their actions to avoid revealing their identities prematurely.

Vandals must blend in with the guards and act inconspicuously, while the guards must be vigilant and observant to detect any suspicious behavior.

```
1 func selectBadGuys():
2     #selec number of badguys
3     var numPlayers = Players.size()
4     var numBadGuys = 2
```

```
5     var badGuysPos = [-1, -1, -1]
6     if numPlayers <= 4:
7         numBadGuys -= 1
8
9     var rng = RandomNumberGenerator.new()
10    while numBadGuys > 0:
11        var setRandom = rng.randi_range(0, numPlayers -1)
12
13        if badGuysPos[0] == -1:
14            badGuysPos[0] = setRandom
15            numBadGuys -= 1
16        elif badGuysPos[0] != setRandom:
17            badGuysPos[1] = setRandom
18            numBadGuys -=1
19    print(badGuysPos)
20    var playerCounter = 0
21    #send info to all players
22    for player in Players:
23        if playerCounter == badGuysPos[numBadGuys]:
24            setBadGuyDictionary.rpc(player)
25            numBadGuys += 1
26            playerCounter += 1
27
28    print(Players)
```

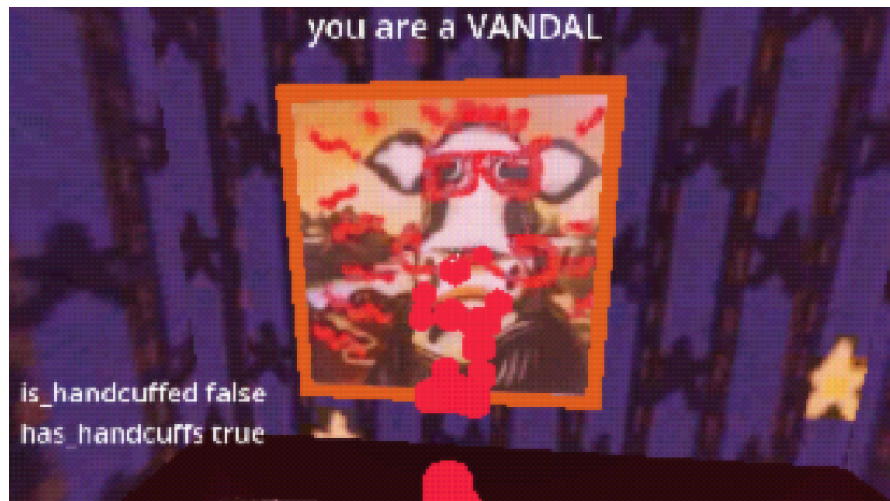


Figure 4.5: Painting Animation

4.3.8 Synchronize Painting State

The paintings are treated as independent objects within the museum to manage their state and synchronize it between players.

Initially, the implementation was considered with a single painting in the scene. Upon receiving a signal from a player, the painting itself would make an RPC (Remote Procedure Call) named `VandalizePainting`, which would notify the painting instance in each player's process that its state had changed and needed to be updated. This approach encountered some issues regarding the permissions of the call, but the solution was relatively straightforward.

Once this was completed, attention turned to implementing multiple identical objects within a single scene. It was essential to be able to distinguish which specific painting had been vandalized to ensure consistency among the different players, allowing everyone to see the same painting defaced in the same position. The challenge that arose was that these objects did not possess their own unique identifiers, such as an IP address, which made it difficult to pinpoint and apply the necessary changes.

To address this issue, a system was devised where all relevant information regarding the paintings' states would be stored on the host player. The host would then be responsible for relaying this information back to the other players. This centralized approach ensured that all players received consistent and accurate updates about the state of each painting, maintaining the integrity of the game's synchronization mechanics.

The implementation of this solution involved creating a method within the "GameManager" script. At the start of the game, this method would search for all objects tagged as "Painting" within the scene. Each discovered painting was then stored in a vector, and the object's name in the scene was changed to reflect its position in the vector to facilitate later usage. For instance, the first painting found would be stored at position 0 in the vector, an attribute `isVandalized` would be initialized to `false`, and the object's name would be changed to "0".

This systematic approach allowed for easy identification and management of each painting within the game. By renaming the objects according to their vector positions, the game could efficiently reference and update specific paintings as needed. This method also ensured that the status of each painting, such as whether it had been vandalized, could be tracked consistently and accurately across all players.

Once a painting was detected as having been vandalized, this information was communicated to the host. The host would then update the painting's state and subsequently share these changes with the rest of the players. By simply sending the position of the

vandalized painting, the system could directly access it by its name and update its state in each of the game instances.

As a consequence of implementing this method, it became easier to track how many paintings had been vandalized and to end the game once all had been defaced. The centralized system allowed for a straightforward check of the vandalized status of each painting. By maintaining a clear and organized record of each painting's state, the game could efficiently determine when all paintings had been vandalized and subsequently trigger the end of the game.

4.3.9 Game Cycle

At the start of each match, roles are randomly assigned. Depending on the number of players, certain players are designated as vandals, while the rest act as guards. The game can end in two ways:

- Guards succeed in discovering and apprehending the vandals, leaving them handcuffed.
- Vandals end the game if they manage to paint all squares on the map before being captured.

These events are monitored by the GameManager, which creates two vectors: one for squares and another for characters, and checks their information each time any of these data are updated during the match. When the game ends, an RPC signal is sent based on the outcome to notify all players with an on-screen message. Afterward, the game concludes, and players return to the initial screen where they can start another match.

```
1 func CountPlayersLeft():
2     var badGuysLeft = false
3     for i in GameManager.Players:
4         #badguy without handcuffs
5         if GameManager.Players[i].badguy && !GameManager.Players[i].handcuffed:
6             badGuysLeft = true
7
8     if !badGuysLeft:
9         GuardsWin.rpc()
10
11 func CountVandalized():
12     var countVandalized = 0
13     for i in GameManager.Paintings:
```



```
14         if GameManager.Paintings[i].isVandalized == true:
15             countVandalized += 1
16
17         #contar cauntos cuadros han sido vandalizados
18         if countVandalized >= GameManager.Paintings.size():
19             return true
20         return false
21
22 @rpc("any_peer", "call_local", "reliable")
23 func VandalsWin():
24     endLabel.text = ("VANDALS_WIN!_/n_all_paintings_were_vandalized")
25
26 @rpc("any_peer","call_local", "reliable")
27 func GuardsWin():
28     endLabel.text = ("GUARDS_WIN_/n_all_vandals_were_cought")
```

4.4 Results

The main purpose of this project was to investigate and understand the mechanics of multiplayer video games, an area I aim to specialize in and continue researching in the future.

Initially, the project aimed to achieve a server-peer communication setup. However, upon researching the topic, the decision was made to first focus on establishing a functional peer-to-peer (P2P) system. The project successfully developed from its initial connection to gameplay and conclusion, creating an interactive player experience. This achievement sets a foundation that can be extended to future projects. It does not preclude further research into achieving a seamless server-peer connection.

As the project stands, it is ready for beta testing by users to gather feedback for potential improvements or changes. The planned platform for release is itch.io, known for hosting a variety of similar caliber indie games.

Due to limited information available on this topic in Godot at present, the entire project is available on GitHub for reference and verification of specific mechanics.

CONCLUSIONS AND FUTURE WORK

Contents

5.1	Conclusions	49
5.2	Future work	50

This chapter presents the conclusions of the project and outlines possible future developments.

5.1 Conclusions

Upon completing this project, which is the first full-scale game development project undertaken from start to finish, the complexity involved has become evident. The challenge of working with multiplayer games has been particularly significant, especially in implementing all the necessary details for proper game control.

The synchronization between players, latency management, and balancing performance with graphical quality required meticulous and constant attention. A version control procedure, akin to that used in a professional environment, was applied throughout the development process.

This approach facilitated tracking and managing code changes and was crucial for team collaboration and the integration of various game components. Tools like Git were essential in maintaining a detailed history of modifications, enabling code reviews, and efficiently resolving conflicts.

In summary, this project has not only been a technical exercise but also a valuable learning experience in project management, teamwork, and the application of professional practices in software development. The experience gained during this process will be invaluable for future projects and has laid a solid foundation for a career in game design and programming.

5.2 Future work

Upon completing the project, a game was developed that allows for testing the final production idea and conducting matches with multiple players from start to finish. However, there are several areas where further work could be pursued.

5.2.1 Introduction of New Roles

One major area for expansion involves the introduction of new roles, beyond the initial ones of vandal and security guard. Adding more roles would enhance gameplay variety and provide players with a broader range of strategies and experiences.

5.2.2 Implementation of a Remote Server

Another significant improvement would be the implementation of a remote server. Establishing a server-peer connection where matches are conducted would minimize the potential for cheating and provide a more seamless experience for all players. This approach would ensure greater fairness and stability in multiplayer sessions.

5.2.3 AI-Controlled Virtual Players

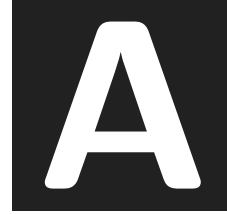
Lastly, incorporating AI-controlled virtual players would be beneficial. This feature would allow the game to be played even without internet access, ensuring that players can enjoy the game at any time. AI players could simulate human behavior, providing a challenging and engaging experience in the absence of real human opponents.

In conclusion, while the current project has successfully met its primary objectives, these enhancements could significantly improve the overall gameplay experience and broaden the game's appeal. Future work in these areas promises to deliver a more robust and versatile gaming experience.

BIBLIOGRAPHY

- [1] Fabio Alessandrelli. Multiplayer in godot 4.0: Scene replication. <https://godotengine.org/article/multiplayer-in-godot-4-0-scene-replication/>. Accessed: 2023-02-23.
- [2] Warner Bros. Back 4 blood. <https://back4blood.com/es-es>.
- [3] DevLogLogan. Basics of multiplayer in godot 4! <https://www.youtube.com/watch?v=e0JLO5UgQolist> = *PLA8H2FkJ25vTNf8cFW7WYJzPn2Jw94ZWdindex* = 112. Accessed : 2023 - 22 - 01.
- [4] Godot Engine. Godot engine system requirements. https://store.steampowered.com/app/404790/Godot_Engine/?l=spanish.
- [5] FinePointCGI. Basics of multiplayer in godot 4! <https://www.youtube.com/watch?v=e0JLO5UgQot> = 3730s. Accessed : 2023 - 08 - 10.
- [6] Daniel M. Pérez Margarita Gaya. Godotonlinemultiplayer-tfg. <https://github.com/DaniMaccari/GodotOnlineMultiplayer-TFG.git>.
- [7] GDQuest. 3d godot movement. <https://www.youtube.com/watch?v=UpF7wm0186Q>. Accessed: 2022-10-30.
- [8] Nagi. How to add interaction in godot 3. <https://www.youtube.com/watch?v=e0JLO5UgQolist> = *PLA8H2FkJ25vTNf8cFW7WYJzPn2Jw94ZWdindex* = 113. Accessed : 2022 - 10 - 10.
- [9] David Serrano. Comparing unreal unity godot. Accessed: 2023-05-15.
- [10] SpainAVS. Libro blanco del videojuego español 2022. <https://spinaudiovisualhub.mineco.gob.es/es/actualidad/libro-blanco-del-desarrollo-espanol-de-videojuegos-2022>. Accessed: 2023-22-07.
- [11] Dustin Tyler. What makes a good game so much fun? <https://www.gamedesigning.org/gaming/great-games/>. Accessed: 2023-10-10.
- [12] Wikipedia. Blender. <https://es.wikipedia.org/wiki/Blender>.

- [13] Wikipedia. Non-functional requirement. https://en.wikipedia.org/wiki/Non-functional_requirement. Accessed : 2024 - 12 - 06.
- [14] Wikipedia. Peer-to-peer. <https://es.wikipedia.org/wiki/Peer-to-peer>. Accessed: 2024-26-03.



SOURCE CODE

All the code for this project was written in `GDScript` aside from the shaders. This programming language is based on Python and shares many similarities with it. The syntax and structure of `GDScript` allow for efficient and readable code, which is particularly beneficial for game development. The most relevant parts of the code are shared in this report, along with detailed explanations of their functionality to provide a clear understanding of how the game's mechanics and systems were implemented.

Additionally, the entire project, including all the scripts, is available on **GitHub**, where it can be publicly reviewed and examined [6]. This open-access approach not only encourages transparency and collaboration but also allows others to learn from and build upon the work that has been done. The repository serves as a comprehensive resource for anyone interested in the development process, the challenges faced, and the solutions implemented.

