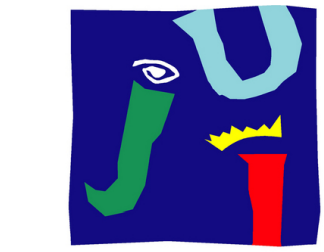

UNIVERSITAT JAUME·I

Development of an infiltration and heists videogame through interaction with the environment and intelligent NPCs

Ana Isabel Torner Ávalos

Final Degree Work

Bachelor's Degree in
Video Game Design and Development

Universitat Jaume I

June 20, 2024

Supervised by: Damaris Pascual González

To my family and friends.

Acknowledgments

First of all, I would like to thank my Degree Final Project supervisor, Damaris Pascual González for his help.

I also would like to thank my family for their support and my friends Enoc López and Adrián "Zero" Cruz for testing my game and for the support given.

Finally, 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.

Abstract

Currently, video games have a very important role in the entertainment industry. Video game production is constantly growing. The design and development of a videogame, is a process that integrate different tasks: graphic design, animation, characters, music, sound, definition of the game mechanic, environment in which the characters act, formalize rewards, etc; until achieve the final version of it.

In this document, is presented the design and implementation of a 3D and third-person videogame named Zero el Zorro, as the Final Degree Work report of Ana Isabel Torner Ávalos in Videogame Design and Development. This one, is about robberies by infiltrating a place in order to acquire the loot it has. The place is protected by security mechanisms and a NPC (Non Playable Character) with artificial intelligence. Also, the player can uses bonuses to improve his effectiveness in the robbery and ensure the victory. The videogame was created with Unity 3D engine, using C# programming code, and it is playable on PC.

Keywords

Document, Final Degree Work, Heist, Infiltration, Artificial Intelligence (AI), Security.

Contents

Acknowledgments **i**

Abstract **iii**

Contents **v**

1 Introduction **1**
1.1 Work Motivation 2
1.2 Objectives 2
1.3 Environment and Initial State 3

2 Planning and resources evaluation **5**
2.1 Planning 5
2.2 Resource Evaluation 8
2.3 Deviations from the initial Planning 8

3 System Analysis and Design **9**
3.1 Requirement Analysis 9
3.2 System Design 17
3.3 System Architecture 21
3.4 Interface Design 22

4 Work Development and Results **31**
4.1 Work Development 31
4.2 Results 39

5 Conclusions and Future Work **41**
5.1 Conclusions 41
5.2 Future work 42

Bibliography **43**

A Source code **45**

CHAPTER 1

INTRODUCTION

Contents

1.1 Work Motivation **2**
1.2 Objectives **2**
1.3 Environment and Initial State **3**

"A video game is an electronic game in which one or more people interact by means of a controller with a device that displays video images. This electronic device, known generically as a "platform", can be a computer, an arcade machine, a game console or a portable device, such as a mobile phone, smartphone or tablet." [14]

The history of videogames has its beginnings in the 1950s, after World War II. Since then, the videogames industry has not stopped growing, in 1972 *Pong* (Nolan Bushnell founded Atari company and published the game), in 1981 *Super Mario Bros* (one of the most famous characters in the world of videogames and Nintendo's mascot ever since it's a platform game in 2D), in 1985 *The Legend of Zelda* (2D adventure game), in 1987 *Final Fantasy* (role-playing game). Currently, in 2015 *Metal Gear Solid V: The Phantom Pain* (infiltration and stealth), in 2016 *Uncharted 4* (3D action-adventure game), in 2017 *Mario Kart 8* (kart racing), in 2023 *Thief Simulator 2* (infiltration and stealth) and so.

In recent years, the video game industry has experienced a rapid development, for example, the visual quality (before in 2D, now in 3D), the use of speakers has improved the sound, the interactivity of the players, mobile phones and tablets as gaming platforms too. It is important to mention that there are a great variety of videogames: adventure, arcade, sports, shooting, strategy, platforms, infiltration, etc.

Unity 3D is a powerful cross-platform to make videogames, both in 2D and 3D, with a pleasant environment. It has a library of assets, supports different resource formats,

uses the languages C# and JavaScript, and it is not so difficult the use of Visual Studio IDE.

Building a video game goes through several stages, starting with the idea itself, followed by the design: artistic (appearance) and mechanics of the game (interactions of the characters in the game) and also the planning, production and testing, among others.

The present work is about a 3D videogame, classified as an infiltration and stealth game, developed in Unity 3D engine, and programmed in C#. In this chapter, is explained the motivations, objectives, environment and initial state of this project.

1.1 Work Motivation

Throughout my life I have played a wide variety of heist, infiltration and stealth video games, being *Payday 2 (Overkill Software, 2013)* [7] one of the games that I have dedicated more time to and that I like the most. This game has been the main motivation and inspiration for my project.

Another of my motivations is to improve the knowledge that I have acquired during my studies, in particular: the use of Unity, C#, 3D modeling, animation and artificial intelligence in videogames.

1.2 Objectives

The main objective of this project is to create an infiltration and heist videogame with 3D low poly style for PC using Unity 3D engine. To achieve this, the following specific objectives are proposed:

- Build the main character in the way that its appearance will be related to the idea of the game (infiltration and heist).
- Implementing a non-playable character with artificial intelligence, specifically, implementing a state machine to reinforce and improve my knowledge acquired in the Artificial intelligence subject.
- To achieve a security system that communicates with the NPC in order to create an interactive environment.
- Adding a save system to the videogame to store the game progress, with the aim of the player continues the game where it was left, and most of all, learning about how to create and implement them in Unity.
- Design an attractive, intuitive and friendly interface to make the players' experience accessible and enjoyable.

1.3 Environment and Initial State

At the beginning of the course, just when I was looking for a theme for my project, the game *Payday 3 (Overkill Software, 2023)* [8] was released, with the release of this game I remembered its predecessor, the game *Payday 2 (Overkill Software, 2013)* [7], a game in which I had spent a lot of hours playing alone or with friends, this game was my inspiration and the source from which I got the idea to create a game of heists, infiltration and stealth. From that moment I had clear the theme of the game but I wanted to give a little twist to the menus. In the game that inspired me, the menus were managed the character and where it is chosen the place to go to do the heists was simply screens. I had the idea of creating a menu with scenery, that is to say, a place where the player could move around and access the different options without it being a simple screen, so I had the idea to create the lair where the player can manage the character and the game, and the map of the city from where the thief could access the place to steal. Another detail I wanted to include in the game was to create a charismatic main character, I didn't want it to be a generic character. Following the theme of the game, I remembered a videogame saga I played when I was a child, the *Sly Cooper* saga created by *Sanzaru Games* [2], the main character of this game was a thief raccoon and inspired by it I chose as the main character a fox.

Taking into account the above, I determined that the game would be developed using the Unity engine, the scenery design with 2D assets, the main character in 3D, and the rest of the 3D models would be purchased from the Synty Studios store [9], and all this work developed by myself. The work methodology I have followed has been to work as many hours as I could, especially at weekends, following as much as possible the established planning and combining the project with the external practices and the study of other subjects of the degree.

CHAPTER 2

Planning and resources evaluation

Contents

2.1 Planning 5
2.2 Resource Evaluation 8
2.3 Deviations from the initial Planning 8

This chapter is the most technical part of the work, it shows the planning followed for the development of the video game and the resources used.

2.1 Planning

This section shows the detailed time planning of the work, including all its tasks and subtasks.

- **Initial investigation (5 hours):** during this task, a brief research was carried out to collect references and to search for assets to be used in the project.
- **Art (75 hours):**
  - **2D Art (15 hours):** the creation of the 2D assets used in the videogame for the UI with the Krita programme.
  - **Modelling (15 hours):** the creation of the 3D model used for the main character, as well as adding the textures, all done with the Blender tool.
  - **Animation (15 hours):** search, configuration and assignment of animations for both the main character and NPCs.

  - **Stage Design (35 hours):** design of all the video game stages, the lair, the map and the stealable location, as well as making sure everything has collisions assigned to it.
  - **Interface Design (5 hours):** design of the entire interface, as well as the functionality of the buttons, trying to create an intuitive and effective design.

- **Development (170 hours):**
  - **Game Mechanics (70 hours):** programming of game mechanics: money system, shop, scene change, item collection and save system.
  - **Player (15 hours):** programming of character and camera movement.
  - **Bonuses (25 hours):** programming of available player bonuses: speed boost and temporary invisibility.
  - **Security Measures (45 hours):** programming of cameras used as a security mode in the stealable location that detect the player and communicate with NPCs.
  - **NPCs (15 hours):** programming of the NPCs patrolling in the stealable location, have been created by making use of a navMesh and a state machine.

- **Documents (50 hours):**
  - **Technical Proposal (2 hours)**
  - **GDD (8 hours)**
  - **Memory (35 hours)**
  - **Presentation (5 hours)**

The image below shows the planning of the tasks and sub-tasks in a visual form with a Gantt chart (see Figure 2.1)

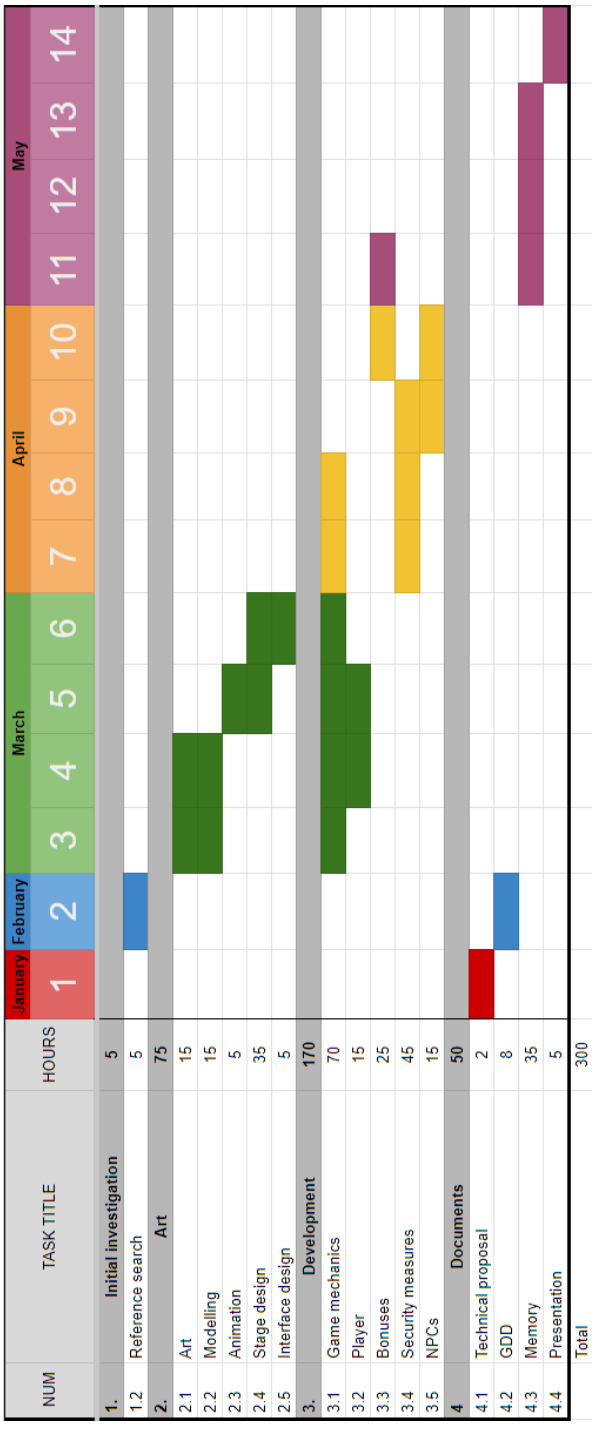

Figure 2.1: Gantt chart of the tasks (made with Google Sheets)

2.2 Resource Evaluation

The resources used for this project are:

- **Hardware:**
  - Laptop MSI Pulse GL76 11UEK-038XES Intel Core i7-11800H/32GB/1TB SSD/RTX 3060/17.3. Cost: 1700€.
- **Software:**
  - Unity 2022.3.20 version [13], used to create the project and work on it. Cost: free.
  - Visual Studio 2019 [5], used to program the project. Cost: free.
  - GitHub Desktop [4], to store the project and have control of the versions. Cost: free.
  - Krita [3], used to create some assets. Cost: free.
  - Blender [1], used to create the 3D model of the main character. Cost: free.
  - Mixamo [6], used to obtain 3D animations. Cost: free.
  - Synty Studios Store [9], a website used to purchase most of the assets used. Cost: 25€.
- **Human resources:**
  - Videogame designer salary [10]: 9,23€/h * 75h = 692,25€.
  - Videogame developer salary [11]: 20,51€/h * 170h = 3486,7€.

Concept	**Amount (€)**
Hardware	1700€
Software	25€
Human resources	4178,95€
Total	**5903,95€**

Table 2.1: Budget Breakdown

2.3 Deviations from the initial Planning

The Initial investigation and Art tasks were carried out in compliance with the planned hours (80). In the case of Development task, the sub-tasks Game mechanics, Player, Bonuses and Security players were executed in the planned time (155 h), but the beginning of the NPC creation was delayed due to problems with the Unity version.

Starting the NPC later than planned also had an impact on the initial moment of writing the memory. Also some reports was delayed.

The rest of sub-tasks: technical proposal and GDD were carried out in time (10 h).

CHAPTER 3

System Analysis and Design

Contents

3.1 Requirement Analysis **9**
3.2 System Design **17**
3.3 System Architecture **21**
3.4 Interface Design **22**

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

Zero el Zorro is a stealth game where the main character infiltrates places to steal their contents in order to get money. To achieve this, he must avoid the security measures and guards of these places, as well as, acquire upgrades to make better heists.

After opening the game, the main menu appears (see Figure 3.2), in this menu there are three options: *Jugar*, *Tutorial* and *Salir*. Pressing *Jugar* will start the gameplay of the game, pressing *Tutorial* will take you to the game tutorial where the game mechanics are explained in a general way (see Figure 3.3) and finally, pressing *Exit* will close the game.

Once inside the game, there are different menus and options. Inside the lair there are three menus: the safe box, the shop and the reset progress menu. In the safe box the player can check their money, in the shop the available character upgrades can be bought and the reset progress option allows to reset the money and the bonuses that the player have purchased to zero.

If the player goes to the stealable location they will see a guide on what to do to complete the level by pressing a button (see Figure 3.10 and 3.11). There will also be icons indicating the objects that have been collected. (see Figure 3.12). If the player is captured, a message will be displayed indicating this (see Figure 3.16) and if they manages to complete the level, a message of victory will be shown (see Figure 3.15).

3.1.1 Functional Requirements

A functional requirement defines a function of the system that is going to be developed. These are the functional requirements of the game:

- **R1**. The player can start the game (Table 3.1).
- **R2**. The player can access to the tutorial (Table 3.2).
- **R3**. The player return to the main menu from the tutorial (Table 3.3).
- **R4**. The player can close the game (Table 3.4).
- **R5**. The player can move through the lair. (Table 3.5).
- **R6**. The player can access to the safe box (Table 3.8).
- **R7**. The player can access to the shop (Table 3.9).
- **R8**. The player can buy bonuses (Table 3.10).
- **R9**. The player can use the invisibility bonus (Table 3.11).
- **R10**. The player velocity updates when they get the velocity bonus (Table 3.12).
- **R11**. The player can access to the restart progress menu (Table 3.13).
- **R12**. The player can restart the progress (Table 3.14).
- **R13**. The player can go from the lair to the city map (Table 3.15).
- **R14**. The player can move through the city map (Table 3.6).
- **R15**. The player can go from the city map to the lair (Table 3.16).
- **R16**. The player can go from the city map to the office (Table 3.17).
- **R17**. The player can move through the office (Table 3.7).
- **R18**. The player can display the guide (Table 3.19).
- **R19**. The cameras detect the player (Table 3.20).
- **R20**. The police officer detect and chase the player (Table 3.21).

- **R21**. The police officer caught the player (Table 3.22).
- **R22**. A caught message appears when the player is captured (Table 3.23).
- **R23**. The player can collect items (Table 3.25).
- **R24**. The card item appears in a random place among the available ones (Table 3.26).
- **R25**. The player can read the note (Table 3.27).
- **R26**. The code from the note is randomly generated (Table 3.28).
- **R27**. The player can use the key pad (Table 3.29).
- **R28**. The player can go from the office to the city map (Table 3.18).
- **R29**. A winning message appears when the player complete the level (Table 3.24).

Input:	Press *Jugar* button
Output:	Game initialized
When the user presses the *Jugar* button, the game must initialize all necessary components and transition to the first scenery, ready for user interaction.	

Table 3.1: Functional requirement «START1. Start Game»

Input:	Press *Tutorial* button
Output:	Tutorial screen displayed
When the user presses the *Tutorial* button, the game must display the tutorial screen, providing instructions and information on how to play the game.	

Table 3.2: Functional requirement «TUT1. Access Tutorial»

Input:	Press *Atrás* button
Output:	Main menu screen displayed
When the user presses the *Atrás* button, the game must display the main menu screen.	

Table 3.3: Functional requirement «TUT2. Return Main Menu»

Input:	Press *Salir* button
Output:	Game application closed
When the user presses the *Salir* button, the game must close the application safely, ensuring all necessary data is saved and resources are properly released.	

Table 3.4: Functional requirement «EXIT1. Exit Game»

Input:	Player movement input (keyboard)
Output:	Player position updated
The player must be able to move within the lair environment using appropriate input controls (W, A, S, D). The game should update the player's position accordingly, allowing exploration and interaction within the lair.	

Table 3.5: Functional requirement «MOVE1. Move Player in Lair»

Input:	Player movement input (keyboard)
Output:	Player position updated
The player must be able to move within the city map environment using appropriate input controls (W, A, S, D). The game should update the player's position accordingly, allowing exploration and interaction within the city map.	

Table 3.6: Functional requirement «MOVE2. Move Player in City Map»

Input:	Player movement input (keyboard)
Output:	Player position updated
The player must be able to move within the office environment using appropriate input controls (W, A, S, D). The game should update the player's position accordingly, allowing exploration and interaction within the office setting.	

Table 3.7: Functional requirement «MOVE3. Move Player in Office»

Input:	Press E key near safe
Output:	Display player's money amount
When the player presses the E key near the safe, the game must display a window with the amount of money the player currently has stored in the safe. This interaction allows the player to check their financial status within the game.	

Table 3.8: Functional requirement «SAFE1. Access Safe to Check Money»

Input:	Press E key near store
Output:	Store interface displayed
When the player presses the E key near the store, the game must display the store interface.	

Table 3.9: Functional requirement «STORE1. Access Store»

Input:	Press the button to buy "+ Velocidad" or the button to buy "Invisibilidad"
Output:	Bonus acquired if sufficient money and not previously purchased
When the player presses the button to buy "+ Velocidad" or the button to buy "Invisibilidad", the game must check if the player has enough money and if the bonus has not already been acquired. If both conditions are met, the respective bonus is granted to the player, and the cost is deducted from the player's money.	

Table 3.10: Functional requirement «STORE2. Buy Bonuses»

Input:	Press F key if cooldown is not active
Output:	Player becomes undetectable
When the player presses the F key, the game must check if the invisibility bonus is available (i.e., the cooldown period is not active). If the bonus is available, the player becomes undetectable for five seconds, and the cooldown period is initiated.	

Table 3.11: Functional requirement «BONUS1. Use Invisibility Bonus»

Input:	Bonus purchased: "+ Velocidad"
Output:	Player speed increased by 15% permanently
When the player purchases the "+ Velocidad" bonus, the game must permanently increase the player's velocity by 15%. This increase should be applied immediately upon purchase and persist throughout the game.	

Table 3.12: Functional requirement «BONUS2. Permanent Velocity Upgrade»

Input:	Press E key near bed
Output:	Reset progress menu displayed
When the player presses the E key near the bed, the game must display the reset progress menu, allowing the player to choose options for resetting their game progress.	

Table 3.13: Functional requirement «RESET1. Access Reset Progress Menu»

Input:	Press "SI" button in reset progress menu
Output:	Player's money and bonuses reset
When the player presses the "SI" button in the reset progress menu, the game must reset the player's money and bonuses to their initial states, effectively restarting the player's progress.	

Table 3.14: Functional requirement «RESET2. Reset Progress Confirmation»

Input:	Press E key near lair door
Output:	Transition to city map scene
When the player presses the E key near the door of the lair, the game must transition from the lair scene to the city map scene, allowing the player to explore the city.	

Table 3.15: Functional requirement «SCENE1. Change from Lair to City Map»

Input:	Press E key near player's house
Output:	Transition to lair scene
When the player presses the E key near the player's house in the city map, the game must transition from the city map scene to the lair scene, allowing the player to enter the lair.	

Table 3.16: Functional requirement «SCENE2. Change from City Map to Lair»

Input:	Press E key near office building
Output:	Transition to office scene
When the player presses the E key near the office building on the city map, the game must transition from the city map scene to the office scene, allowing the player to enter and interact within the office environment.	

Table 3.17: Functional requirement «SCENE3. Change from City Map to Office»

Input:	Press E key near exit door
Output:	Transition to city map scene
When the player presses the E key near the exit door in the office map, the game must transition from the office map scene back to the city map scene, allowing the player to return to the city environment.	

Table 3.18: Functional requirement «SCENE4. Change from Office Map to City Map»

Input:	Press Tab key
Output:	Guide interface displayed
When the player presses the Tab key, the game must display a guide interface, providing a guide to complete the level. .	

Table 3.19: Functional requirement «GUIDE1. Display Guide on Tab Press»

Input:	Player enters camera detection area
Output:	Camera detects player presence
When the player enters the detection area of a camera, the camera must detect the player's presence and advice the NPC. This mechanic enhances the stealth aspects of gameplay.	

Table 3.20: Functional requirement «CAMERA1. Player Detection by Cameras»

Input:	Player enters police's line of sight
Output:	Police starts chasing player
When the player enters the police's line of sight, the police must detect the player and initiate a pursuit. The police will actively attempt to apprehend the player, adding challenge and consequences to gameplay scenarios involving law enforcement.	

Table 3.21: Functional requirement «POLICE1. Player Detection by Police»

Input:	Police collides with player
Output:	Player captured by police
When the police collides with the player during a pursuit, the player must be captured by the police..	

Table 3.22: Functional requirement «POLICE2. Police Captures Player on Collision»

Input:	Player captured event
Output:	Display a player captured message
When the player is captured, the game must display a message indicating that the player has been captured.	

Table 3.23: Functional requirement «MESSAGE1. Display Captured Message»

Input:	Player completes level
Output:	Display a victory message
When the player completes the level successfully, the game must display a message indicating that the player has won. This notification celebrates the player's achievement and signifies the completion of the gameplay objective.	

Table 3.24: Functional requirement «MESSAGE2. Display Victory Message»

Input:	Press E key near object
Output:	Object collected by player
When the player presses the E key near an object, the game must allow the player to collect the object. This interaction enables the player to gather items useful for gameplay progression.	

Table 3.25: Functional requirement «OBJECT1. Object Pickup»

Input:	Random appearance in available locations
Output:	Card object appears
The card object must appear randomly in one of the available locations within the office map. This randomness adds variability and difficulty to the gameplay.	

Table 3.26: Functional requirement «OBJECT2. Random Spawn of Card Object»

Input:	Press E key near note
Output:	Note content displayed
When the player presses the E key near a note, the game must display the content of the note. This interaction allows the player to read the code for the key pad.	

Table 3.27: Functional requirement «NOTE1. Read Note»

Input:	Note with randomly generated code
Output:	Random code displayed
The code from the note is generated randomly. This random generation adds variability and difficulty to the game.	

Table 3.28: Functional requirement «NOTE2. Randomly Generated Code on Note»

Input:	Press E key near keypad, enter numbers, press buttons
Output:	Display entered numbers, validate code, clear input
When the player presses the E key near the keypad, they can enter numbers using the keypad buttons. The entered numbers should appear on the keypad screen. Pressing the "M" button submits the entered code for validation. Pressing the "C" button clears the entered code. This interaction allows the player to input and validate the code to open manager's door.	

Table 3.29: Functional requirement «KEYPAD1. Use of Keypad»

3.1.2 Non-functional Requirements

- **R30**. The game can be played on PC.
- **R31**. The game will be made with low poly models.
- **R32**. The UI will be simple and clear.
- **R33**. The mechanics of the game will be simple to understand.
- **R34**. The main character will be original.
- **R35**. The game will be replayable.

3.2 System Design

In this section is presented the logical or operational design of the system to be carried out. These logical design is shown below with the use case diagram of the game (see Figure 3.1).

Requirement:	R1
Actor:	Player
Description:	The player can start the game
Preconditions:	1.The player is on the main menu
Normal sequence:	1.The player presses the button *Jugar* 2.The system load the lair stage
Alternative sequence:	None

Table 3.30: Case of use «Start game»

Requirement:	R2
Actor:	Player
Description:	The player can go to the tutorial
Preconditions:	1.The player is on the main menu
Normal sequence:	1.The player presses the button *Tutorial* 2.The tutorial is shown
Alternative sequence:	None

Table 3.31: Case of use «Go to tutorial»

Requirement:	R4
Actor:	Player
Description:	The player can close the game
Preconditions:	1.The player is on the main menu
Normal sequence:	1.The player presses the button *Salir* 2.The system close the game
Alternative sequence:	None

Table 3.32: Case of use «Quit the game»

Requirement:	R6
Actor:	Player
Description:	The player can check how much money they has
Preconditions:	1.The player is on the lair
Normal sequence:	1.The player is near the shop 2.The player press E key 3.A window opens showing the money
Alternative sequence:	None

Table 3.33: Case of use «Consult money»

Requirement:	R8
Actor:	Player
Description:	The player can buy bonuses
Preconditions:	1.The player is on the lair
Normal sequence:	1.The player is near the shop 2.The player press E key 3.The player press a button to buy a bonus 4.The bonus is bought
Alternative sequence:	4.1.The bonus is not bought because the player does not have enough money 4.2.The bonus is not bought because the player has already bought it

Table 3.34: Case of use «Buy bonus»

Requirement:	R5, R14, R17
Actor:	Player
Description:	The player can move through all the stages
Preconditions:	1.The player is on the lair/city/office
Normal sequence:	1.The player press W,A,S or D keys 2.The character moves
Alternative sequence:	None

Table 3.35: Case of use «Move»

Requirement:	R23
Actor:	Player
Description:	The player can take items
Preconditions:	1.The player is on the office stage 2.The player is near an object
Normal sequence:	1.The player press E key 2.The item is recollected
Alternative sequence:	None

Table 3.36: Case of use «Take item»

Requirement:	R15
Actor:	Player
Description:	The player can go to the lair
Preconditions:	1.The player is on the city map 2.The player is near the player's house
Normal sequence:	1.The player press E key 2.The system loads the lair stage
Alternative sequence:	None

Table 3.37: Case of use «Go to lair»

Requirement:	R13, R28
Actor:	Player
Description:	The player can go to the city map
Preconditions:	1.The player is on the lair/office 2.The player is near the exit door
Normal sequence:	1.The player press E key 2.The system loads the city map stage
Alternative sequence:	2.1.The system doesn't load the stage because the player hasn't yet completed the office level.

Table 3.38: Case of use «Go to city map»

Requirement:	R16
Actor:	Player
Description:	The player can go to the office
Preconditions:	1.The player is on the city map 2.The player is near the office
Normal sequence:	1.The player press E key 2.The system loads the office stage
Alternative sequence:	None

Table 3.39: Case of use «Go to office»

Requirement:	R19, R20, R21
Actor:	Security systems / NPC
Description:	The player is detected by the security systems / NPC
Preconditions:	1.The player is on the office
Normal sequence:	1.The player collide with the detection area of the security systems / NPC
Alternative sequence:	None

Table 3.40: Case of use «Detect player»

3.3 System Architecture

According to the Unity documentation[12], the minimum requirements to run games made with this game engine are:

- A minimum operating system of Windows 7 SP1 or newer.
- A CPU with x86 or x64 architecture, including SSE2 instruction set support.
- A graphics card (GPU) supporting DX10 or higher.
- A keyboard and mouse, or alternatively, a touch panel.

I cannot guarantee that the game will run on all devices that fulfill these requirements, but the game has been played successfully on a computer with the following characteristics:

- Intel Celeron N5100
- 8GB RAM
- Intel UHD Graphics 600, 250-750MHz

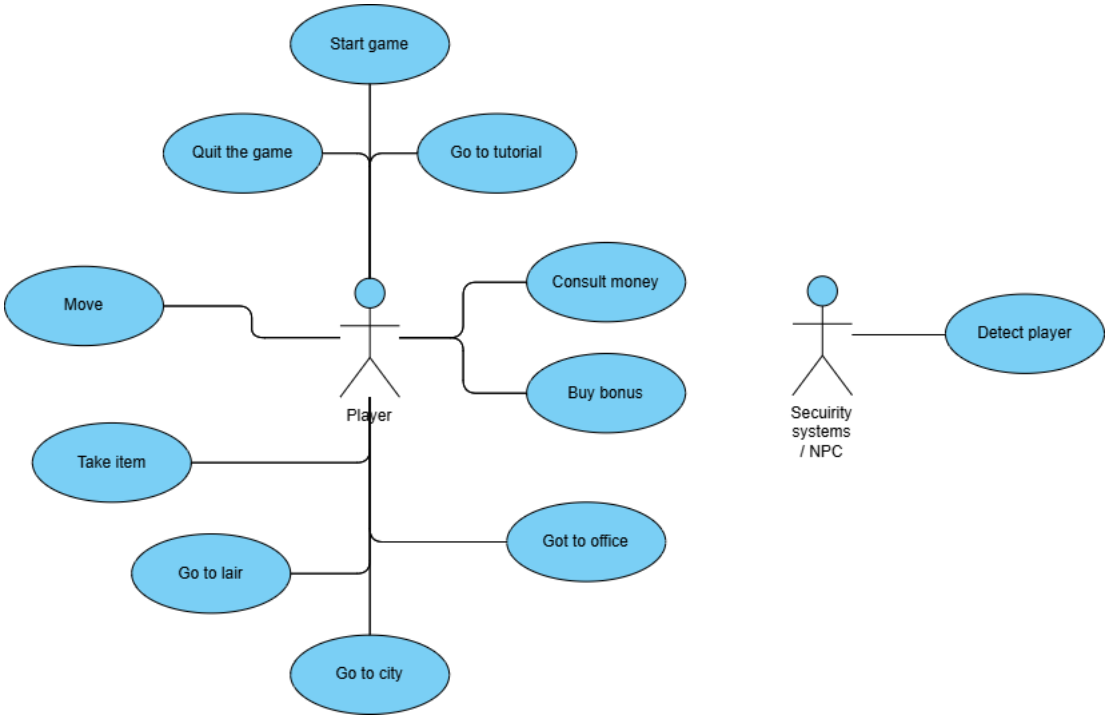

Figure 3.1: Case use diagram (made with `https://online.visual-paradigm.com`)

3.4 Interface Design

The game user interface (GUI) is simple and non-invasive. The interface, which appears during the game, is mostly hidden and only appears if the player requires it. The first interface the player sees when starting the game is the main menu. This menu has a background image, a game title/logo and three buttons, the button to start the game, the button to view the tutorial and the button to exit (see Figure 3.2).

By clicking on the tutorial button, the player is taken to the tutorial and can see the interface with images and advice texts, also the menu can be returned to with the button "Atrás". (see Figure 3.3).

Once stated the game a variety of interfaces and menus can be seen:

When the player approaches a door or interactive object, an icon appears in the centre of the screen with the key to press to interact (see Figure 3.4).

Inside the lair the player can find three windows/menus. The first one would be the safe. Here the player can see the current money being updated (see Figure 3.5). The second one would be the shop which has two buttons, they will be red if they cannot be pressed and green if they can be pressed, they will also have a tick next to them if the bonus has already been bought (see Figure 3.6). The third one would be the reset menu, it has two buttons, one to close the menu and one to confirm the reset (see Figure 3.7).

If the player purchases the invisibility bonus, an icon will appear on the screen with

the key to press, if activated, two counters will appear, one for the time the bonus is active and one for the cooldown (see Figures 3.8 and 3.9).

Within the stealable site there is a wide variety of UIs:

The first thing the player will see is an icon indicating a button to press. Pressing the button displays a guide (see Figures 3.10 and 3.11).

While playing through the level the player must collect two items, once collected they will be displayed on the screen indicating that they are in their possession (see Figure 3.12).

The player will have to obtain a code, which will be displayed in a note (see Figure 3.13), and once obtained, the player will have to use a key pad to enter it (see Figure 3.14).

If the player completes the level and escapes to the city map a victory message will be displayed (see Figure 3.15) otherwise if they are captured by the security guard a defeat message will appear (see Figure 3.16).

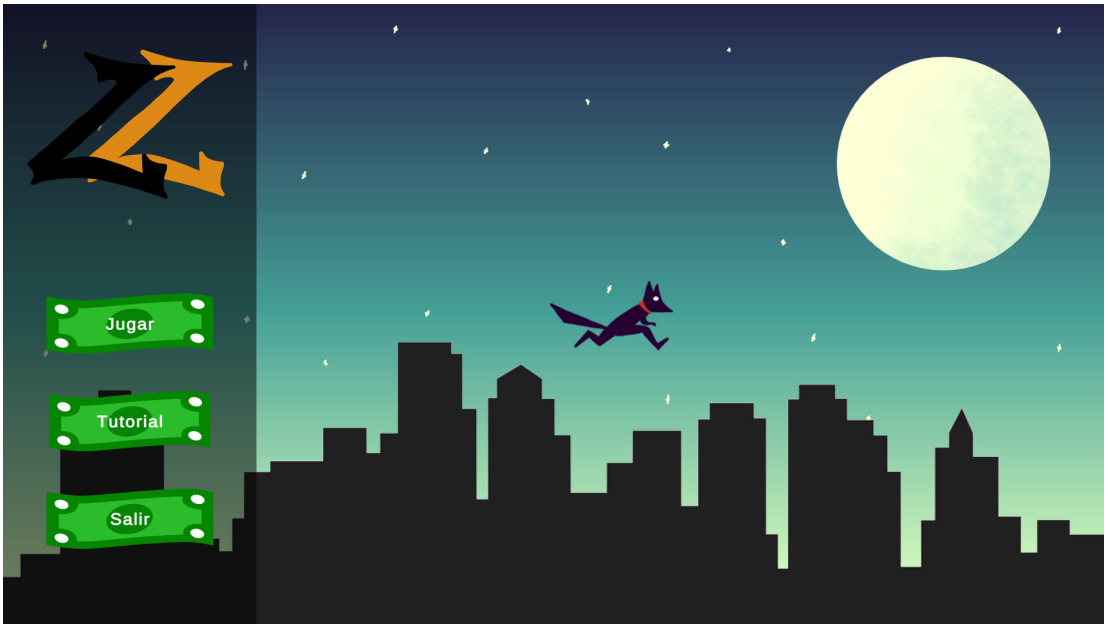

Figure 3.2: Main menu of the game (image taken in game)

Figure 3.3: Game tutorial in the main menu (image taken in game)

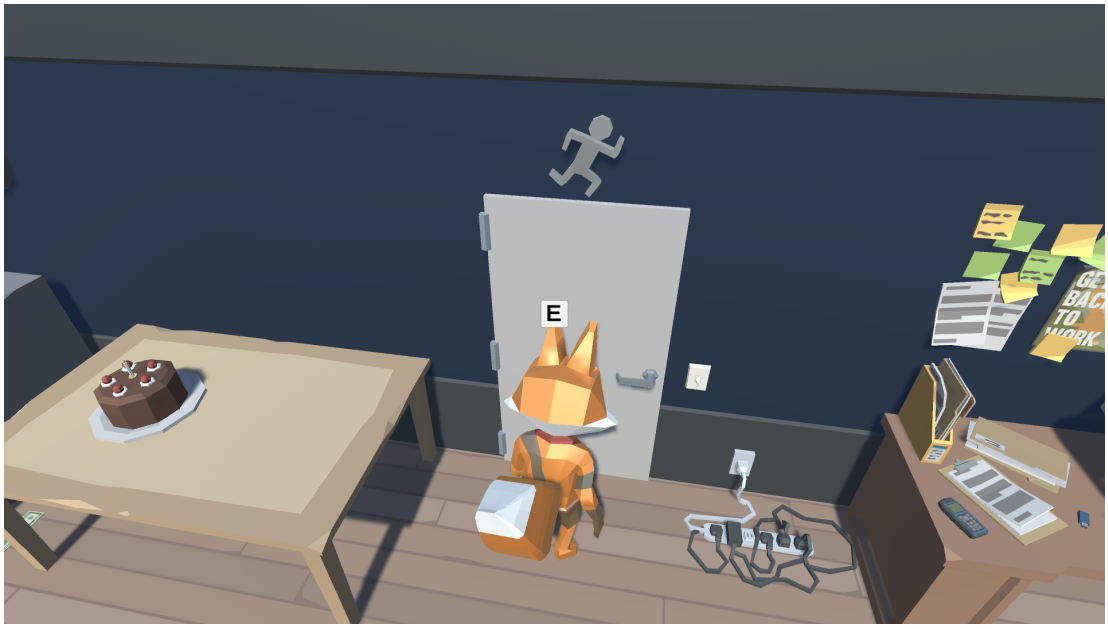

Figure 3.4: Icon of the key to press to interact (image taken in game)

Figure 3.5: Menu of the safe box (image taken in game)

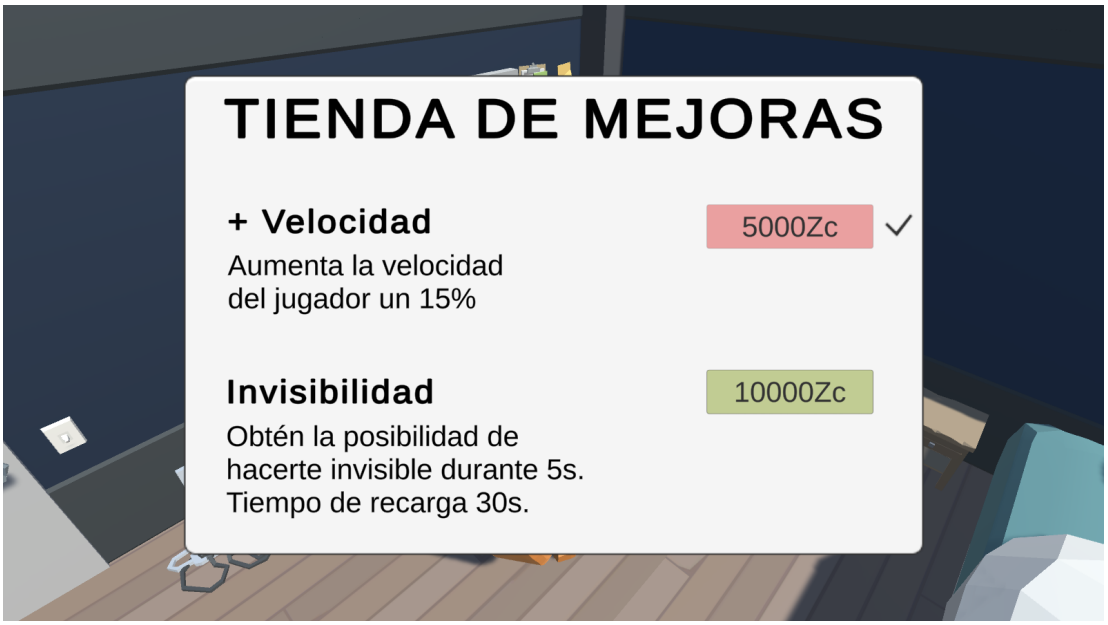

Figure 3.6: Menu of the shop (image taken in game)

Figure 3.7: Menu to restart the game (image taken in game)

Figure 3.8: Invisibility bonus deactivated (image taken in game)

Figure 3.9: Invisibility bonus activated (image taken in game)

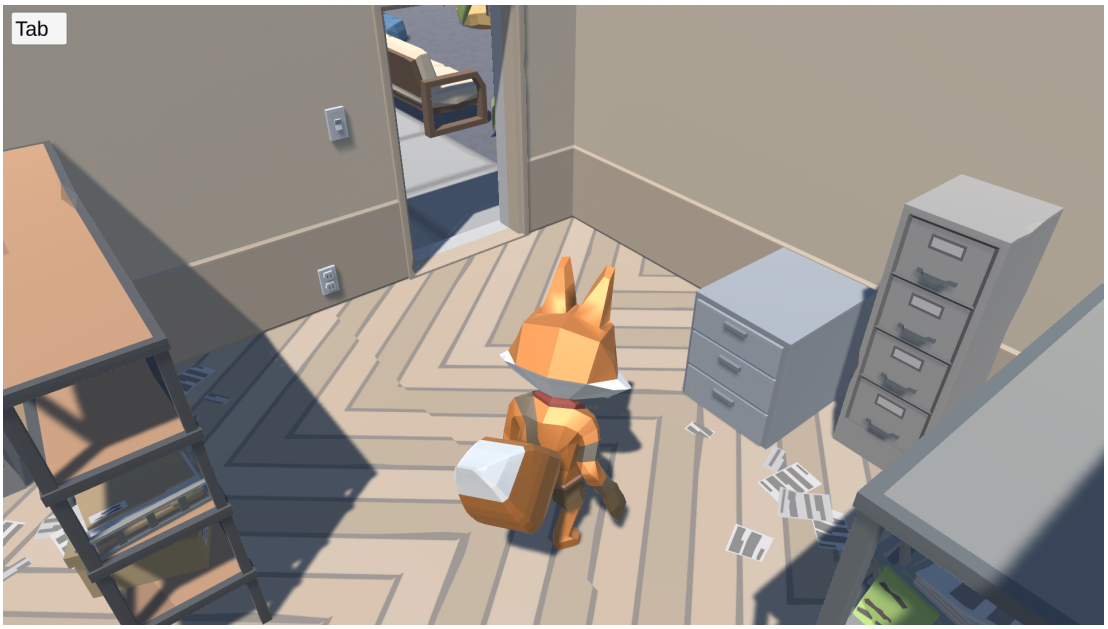

Figure 3.10: Button to display the guide (image taken in game)

Figure 3.11: Guide displayed (image taken in game)

Figure 3.12: USB and key card icons (image taken in game)

Figure 3.13: Note with the code (image taken in game)

Figure 3.14: Key pad (image taken in game)

Figure 3.15: Winning message (image taken in game)

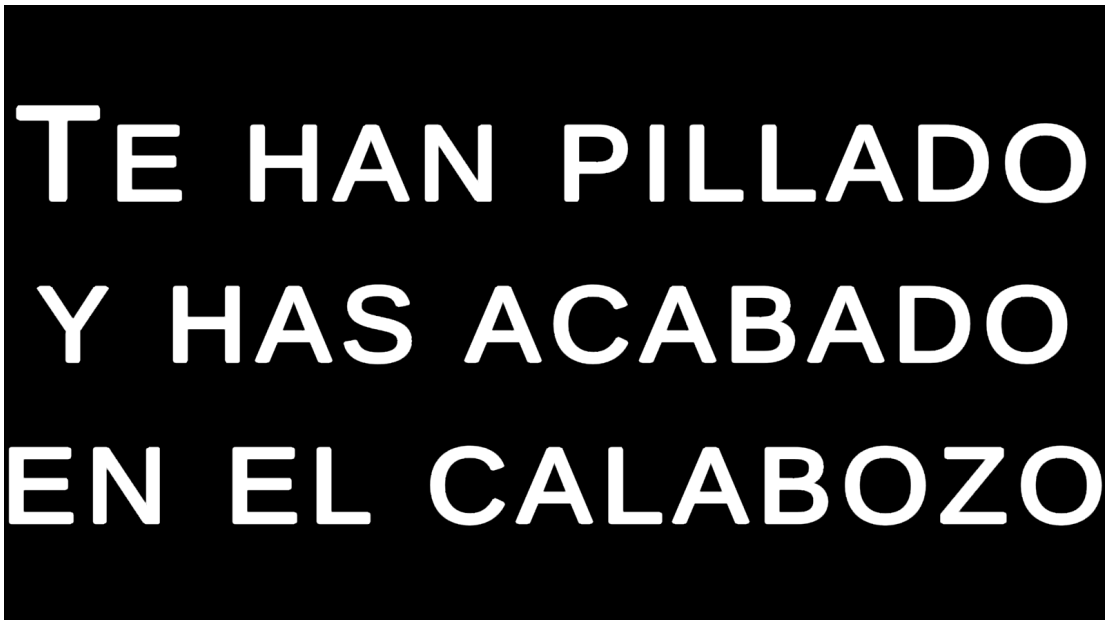

Figure 3.16: Caught message (image taken in game)

CHAPTER 4

Work Development and Results

Contents

4.1 Work Development **31**
4.2 Results **39**

In this chapter it is shown the game development work and the results.

4.1 Work Development

The work done will be explained following the order of the planning. Therefore, the explanation will be divided into three parts: Initial investigation, art and development.

Initial investigation

In this part I was busy researching information to lay the base for what the game was going to be. I researched a variety of stealth and heist games to learn more about their mechanics and gameplay. I also looked for assets to use in the game, in this case 3D models, taking into account that I had made the decision to make a low poly game.

Art

This part refers to everything to do with the artistic design of the game and has five subsections.

The first one would be *Art*, which refers to the 2D assets used during the game. The assets created by me are the background of the main menu (see Figure 4.1), the buttons used in the main menu (see Figure 4.2), the game logo (see Figure 4.3), the item icons

(see Figure 4.4), the invisibility bonus icon (see Figure 4.5) and all the key pad elements (see Figure 4.6). All of this was made with *Krita*[3].

The second subsection is *Modelling*. This part was dedicated to creating the model of the main character (see Figure 4.7) and texturing it, adapting it to the style of the game and especially to the look of the NPC (see Figure 4.15). This tasks was made using *Blender*[1].

The next part is *Animation*. To make the animations of the characters I used *Mixamo*[6], a software that includes a large library of animations and allows to adjust them for own models and export these animations to be used in different projects.

The fourth subsection is *Stage design*. In this section I designed all the stages of the game: the lair (see Figure 4.8), the city (see Figure 4.9) and the office (see Figure 4.10).

Finally, the section Interface design, where I created the whole interface of the game, both the main menu and the one that appears throughout the gameplay (see section 3.4).

Figure 4.1: Background of the main menu

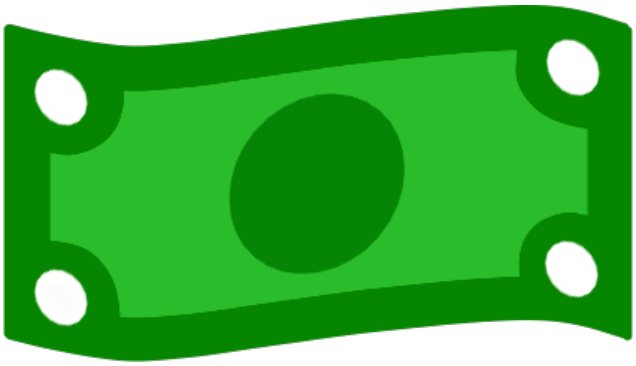

Figure 4.2: Main menu buttons icon

Figure 4.3: Game logo

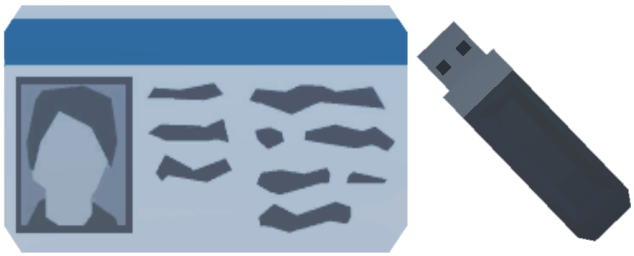

Figure 4.4: Card and USB items icons

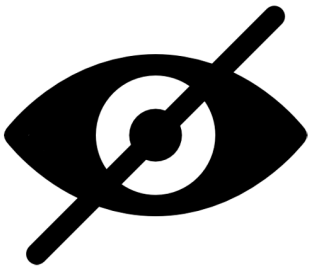

Figure 4.5: Invisibility icon

Figure 4.6: Key pad

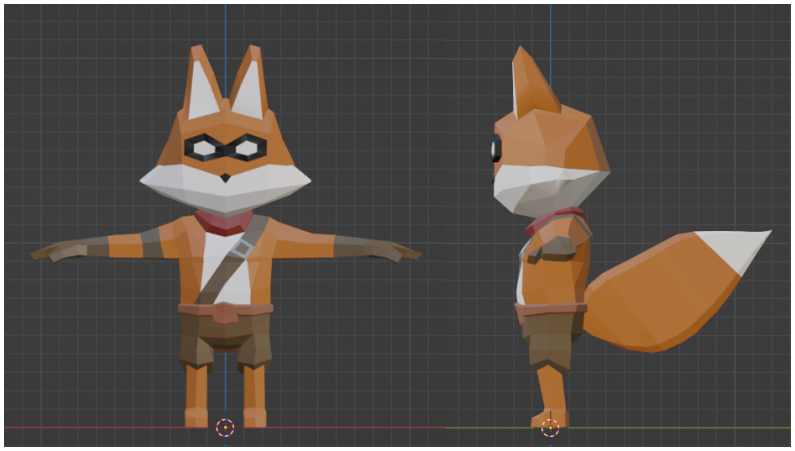

Figure 4.7: Main character 3D model

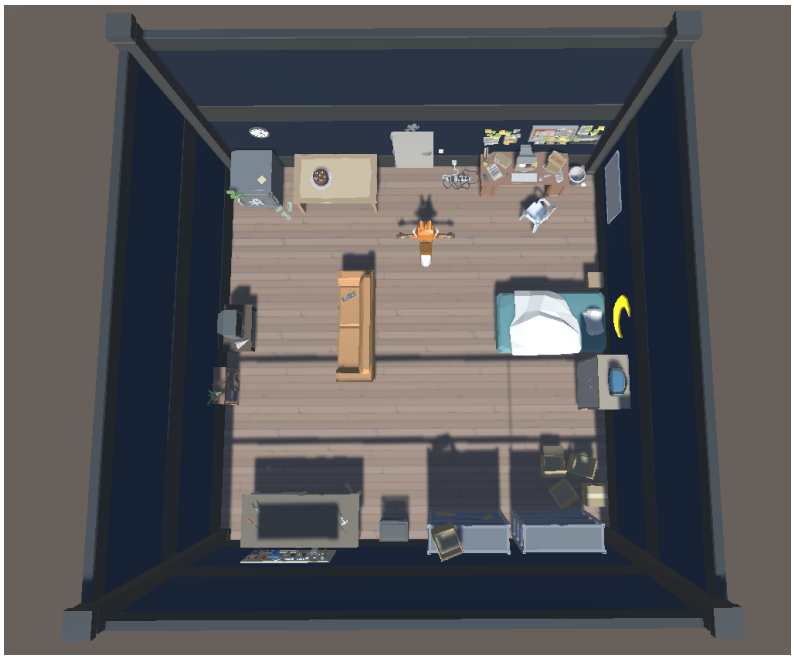

Figure 4.8: Lair stage

Figure 4.9: City stage

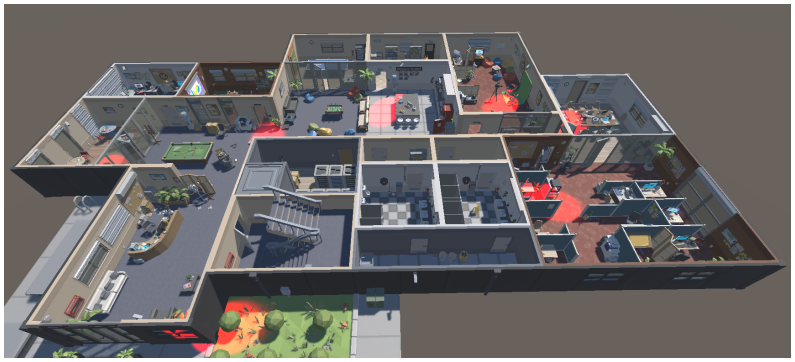

Figure 4.10: Office stage

Development

This section refers to all programming aspects of the game. Again, it is divided into five subsections.

The first subsection is *Game mechanics*, it is related to the general mechanics of the game.

When the player is close to an interacting object, an icon with the key to press for the interaction will appear on the screen (*floatingImage*).

Changes of scene (*SceneManagement*). You can go from the lair to the city, from the city to the lair, from the city to the office and from the office to the city. To go from one location to another, press the E key near the accesses to these locations (when the icon appears on the screen).

The functioning of the monetary system (*MoneyManager*) and the safe (*SafeBox*). The player gets money when the level is completed, this money can be consulted in the SafeBox inside the lair (see Figure 4.11) by pressing the E key near the lair (when the icon appears on the screen). The money earned is saved and is retained even if the game is restarted thanks to the save system implemented.

Objects in the office. The cards that appear randomly in one of the available locations on the map (*RandomObjectVisibility*), when collected, they will open the human resources door where the player needs to enter to continue advancing. A USB which is the object that the user needs to get to unlock the exit and thus complete the level, this will appear in a fixed location. These items (see Figure 4.12), when the icon appears on screen, are collected by pressing E key near them (*ItemCollector*).

The code to access the manager's office. Once the player get the card, as it is said before, the human resources door opens and inside there is a note that can be interacted with by pressing the E key near it (when the icon appears on the screen). In this note will appear a code that is randomly generated every time the level is accessed (*CodeScript*). Once the code is obtained, the Player has to enter it in the numerical keyboard that there is in the door of the manager's office (*KeyPadScript*), this keyboard is composed of buttons with numerical keys, a button to delete the code and another one to enter it, if it is correct the door will be opened. As the player types in numbers, this will be

displayed on the small screen on the keypad, the maximum number of numbers that can be typed in is four (see Figures 3.14 and 4.6).

As soon as the USB is obtained and the exit is unlocked, if the player goes to the exit and presses the E key nearby (when the icon appears on the screen) the city scene (*exitManagment*) will be loaded, taking into account that the player has completed the level, therefore, the player will appear in front of the office and a message will emerge indicating that the money has been obtained (*CitySpawnManager*).

Inside the office, the player will be able to see a small guide to the steps to complete the level (*GuideScript*). The guide will be displayed by holding down the Tab key and hidden when the key is released (see Figures 3.10 and 3.11).

If the player wishes, all the progress can be restarted from the lair bed (see Figure 4.11), resetting will affect both money and bonuses.

The second subsection is *Player*. This subsection concerns to the movement of the player (*PlayerMovement*) and the camera (*CameraController*). The character can move and rotate using the movement keys (W, A, S, D), and the camera will rotate with the character, always behind the character.

The next subsection is *Bonuses*. In this part, everything related to character upgrades was programmed.

On the one hand, there is the shop which can be accessed from the computer in the lair (see Figure 4.11). In this shop the player can buy the two bonuses that are available, the bought bonuses will be kept as well as money (*BonusManager*). To buy them, the player must click on the buttons, which will only be available if they have enough money and the upgrade has not been bought yet, if the upgrade has already been bought it will be marked with a tick next to the button (see Figure 3.6) (*ShopScript*).

The first bonus is *+Velocity*. This bonus will permanently improve the player's speed by 15% (*VelocityBonus*).

The other bonus is *Invisibility*. This bonus allows the player to become invisible for five seconds and has a cooldown of thirty seconds (see Figures 3.8 and 3.9). If the bonus is active the player will become undetectable to the cameras and to the NPC.

The next subsection, *Security measures*, is about the security cameras in the office stage.

Throughout the stage there are several security cameras (see Figure 4.13), these will rotate between set angles (*SecurityCameras*) and have a detection area marked with a red light, if the player enters this detection area the camera invokes the player's position and sends it to the NPC to investigate the area (*CameraDetection*).

The last subsection is *NPCs*. This involves all aspects of NPC creation.

The first thing was to create a navigation mesh (see Figure 4.14) to mark where the NPC (see Figure 4.15) should move and place waypoints to use for their route later on. The NPC will change state/task as needed (*StateMachine*).

At first it will be in the *Normal* state, in this state the NPC will wander between the waypoints in the scene. If a camera detects the player it will switch to *Investigate* state, in this state it will go to the location where the player has been detected and switch to *Wait* state and stay there for a few seconds, then return to *Normal* state. If

the player enters the NPC's detection area the NPC will enter the *Chase* state and start chasing the player. If he manages to catch the player (collide) the game will be over, a message will appear warning that the player has been captured (see Figure 3.16) and it will be indicated that the player has been captured (*GameEnding*), this way the player will appear in the city in front of the police station (*CitySpawnManager*).

Figure 4.11: Safe, PC and Bed from the lair

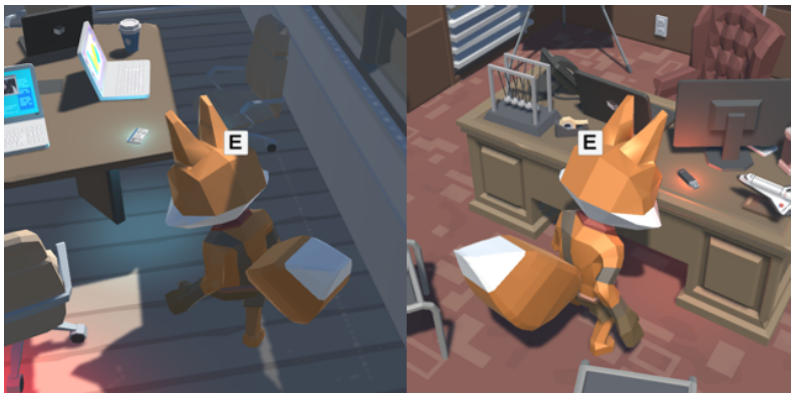

Figure 4.12: Items from the office

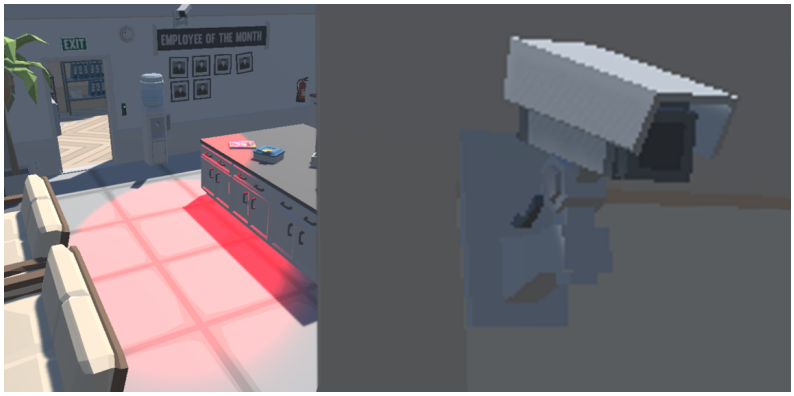

Figure 4.13: Office cameras

Figure 4.14: Navigation mesh

4.2 Results

The first objective was achieved because the main character is a fox. In addition to the considerations mentioned at the beginning of this work, in many animal stories the fox is characterised as a clever, petty thief, deceitful, stealthy, which is combined with the fact that it is a game of heist and infiltration.

The second goal was achieved too. This was to implement an NPC with artificial intelligence using a state machine. I did this by programming a state machine with four states: Normal, Investigate, Wait and Chase, the NPC switches between these states when the condition of these ones occurs. This objective, although it took me more time than expected and delayed the project a bit, was achieved as I had planned.

The third one was to create security measures that communicate with the NPC

Figure 4.15: NPC

to create an interactive environment. This objective was achieved, I have created the security cameras that have a detection area and when the player enters they invoke the character's location for the NPC to go there.

Another outcome is the creation of a save system for the game, in correspondence with the fourth objective. This system was to save the money the player owned and the bonuses he had acquired. This objective was thought mainly because I wanted to learn how to create a save system even if it was a simple version. The goal was successfully achieved, I was able to implement it without any problems and I learned how to do it.

With regard to the fifth objective, the interface was made in a simple way, intuitive, friendly to provide the user with a pleasant experience.

Finally, a 3D style infiltration and heist videogame for PC was created using the Unity 3D engine, with a coherent game flow and usability, achieving the overall goal.

The GitHub repository can be accessed here:

`https://github.com/AnabelTorner/TFG`

And you can also access the build of the project for Windows here:

`https://drive.google.com/drive/folders/1bkqokA99D464Gfq-7k-4wU4v3deKgjE_?usp=sharing`

CHAPTER 5

Conclusions and Future Work

Contents

5.1 Conclusions **41**
5.2 Future work **42**

In this chapter, the conclusions of the work, as well as its future extensions are shown.

5.1 Conclusions

In this work I have mainly shown my ability to design and develop a 3D videogame, I have been able to learn the different phases to create a videogame, the planning, the difficulties and the constraints presented during the work. Knowing more about Unity and C# gave me the desired result. The video game presented here fulfils the conditions I was looking for, the design of the main character is appropriate according to the idea of the game and the NPC, the use of AI to program the NPC and the cameras make the game more interesting by incorporating bonuses, collectable objects, etc. to provide a pleasant interface and make a story of robbery and stealth around the fox.

In conclusion, after finishing this project I can say that I have enjoyed developing it and improving my skills, but I also have to say that it has been months of hard work to complete it. I have had to combine the development of the project with the external practices, with the study and delivery of work of other subjects. It has been a difficult task but a great experience. I have to say that I would have liked to have more time for the development of this project, and to include more aspects in the game to make it more playable, but unfortunately it has been impossible for me.

Nevertheless, I am satisfied with the final result of the game and with having been able to improve my skills in videogame design and development and to learn new concepts.

5.2 Future work

The created game has no endgame, it is conceived as a replayable game, so it has a lot of possibilities for expansion. New bonuses, more stealable locations, more security measures, as well as more NPCs can be added. It is possible that, in the future, I will try to improve these aspects of the game in order to continue practising and learning.

Bibliography

[1] Blender. Blender software. https://www.blender.org/.

[2] Sanzaru Games. Sly cooper. https://es.wikipedia.org/wiki/Sly$_{C}ooper_{(}serie)$, 2002.

[3] Krita. Krita software. https://krita.org/es/.

[4] Microsoft. Github. https://github.com/.

[5] Microsoft. Microsoft visual studio. https://visualstudio.microsoft.com/es/downloads/, 2019.

[6] Mixamo. Mixamo web. https://www.mixamo.com/#/.

[7] Overkill Software. Payday 2. https://www.paydaythegame.com/payday2/, 2013.

[8] Overkill Software. Payday 3. https://www.paydaythegame.com/payday3/, 2023.

[9] Synty Studios. Synty store. https://syntystore.com/.

[10] Talent. Videogame designer salary. https://es.talent.com/salary?job=dise%C3%B1ador+videojuegos.

[11] Talent. Videogame developer salary. https://es.talent.com/salary?job=desarrollador+videojuegos.

[12] Unity. Unity documentation. https://docs.unity3d.com/Manual/UnityManual.html.

[13] Unity. Unity technologies. https://unity.com/es, 2022.

[14] Wikipedia. Videojuego. https://es.wikipedia.org/wiki/Videojuego.

APPENDIX A

Source code

This appendix contains all the scripts used in the development of the project. All the code can be accessed and downloaded from the link to the repository in section 4.2.

SceneManagement script

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneManagement : MonoBehaviour
{

    public string SceneName;

    public void CerrarJuego()
    {
        Application.Quit(); // Cierra el juego
        Debug.Log("Salir"); // Comprobación por consola para Unity
    }

    public void CambioEscena()
    {
        SceneManager.LoadScene(SceneName); // Carga la escena indicada
    }

    private void OnTriggerStay()
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            CambioEscena(); // Carga la escena indicada
        }
    }
```

```
}
```

PlayerMovement script

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float moveSpeed = 3f; // Velocidad de movimiento del jugador
    public float rotationSpeed = 3f; // Velocidad de rotación del jugador
    public Transform cameraTransform; // Transform de la cámara

    private Rigidbody rb;
    private Animator animator;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();
        rb.freezeRotation = true; // Evitar rotaciones debido a colisiones
    }

    void Update()
    {
        // Obtener la entrada de teclado para el movimiento horizontal y vertical
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        // Calcular la dirección de movimiento relativa a la cámara
        Vector3 forward = cameraTransform.forward;
        Vector3 right = cameraTransform.right;

        // Asegurarse de que forward y right no tengan componente y
        forward.y = 0f;
        right.y = 0f;
        forward.Normalize();
        right.Normalize();

        // Calcular el vector de movimiento basado en la entrada del teclado y la cámara
        Vector3 movement = forward * moveVertical + right * moveHorizontal;

        // Aplicar el movimiento al Rigidbody
        rb.velocity = movement * moveSpeed;

        // Actualizar el parámetro IsWalking del Animator basado en el movimiento
        bool isWalking = movement != Vector3.zero;
        animator.SetBool("IsWalking", isWalking);

        // Rotar el personaje hacia la dirección del movimiento
        if (isWalking)
        {
            Quaternion toRotation = Quaternion.LookRotation(movement, Vector3.up);
```

```
            transform.rotation = Quaternion.Slerp(transform.rotation, toRotation, rotationSpeed
            * Time.deltaTime);
        }
    }
}
```

CameraController script

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
    public Transform target; // Referencia al jugador
    public float smoothSpeed = 0.125f; // Velocidad de suavizado
    public Vector3 locationOffset; // Desplazamiento de posición
    public Vector3 rotationOffset; // Desplazamiento de rotación

    void FixedUpdate()
    {
        // Calcula la posición deseada de la cámara
        Vector3 desiredPosition = target.position + target.rotation * locationOffset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition,
        smoothSpeed);
        transform.position = smoothedPosition;

        // Calcula la rotación deseada de la cámara
        Quaternion desiredRotation = target.rotation * Quaternion.Euler(rotationOffset);
        Quaternion smoothedRotation = Quaternion.Lerp(transform.rotation, desiredRotation,
        smoothSpeed);
        transform.rotation = smoothedRotation;
    }
}
```

BonusManager script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BonusManager : MonoBehaviour
{
    public static BonusManager instance;
    public VelocityBonus velocityBonus;

    public GameObject invisibilityBonusCanvas;

    private string velocityKey = "VelocityBonus";
    private string invisibilityKey = "InvisibilityBonus";

    public bool velocidadComprada = false;
```

```
16      public bool invisibilidadComprada = false;
17
18      private void Awake()
19      {
20          // Verifica si ya existe una instancia del BonusManager
21          if (instance == null)
22          {
23              // Si no existe, establece esta instancia como la instancia única
24              instance = this;
25              // Marca este GameObject como no destruible al cargar una nueva escena
26              DontDestroyOnLoad(gameObject);
27          }
28          else
29          {
30              // Si ya existe una instancia, destruye este GameObject para evitar duplicados
31              Destroy(gameObject);
32          }
33      }
34
35      private void Start()
36      {
37          LoadPlayerBonus();
38          BonusInvisibilityActive();
39      }
40
41      private void SavePlayerBonus()
42      {
43          PlayerPrefs.SetInt(velocityKey, velocidadComprada ? 1 : 0);
44          PlayerPrefs.SetInt(invisibilityKey, invisibilidadComprada ? 1 : 0);
45          PlayerPrefs.Save();
46      }
47
48      private void LoadPlayerBonus()
49      {
50          velocidadComprada = PlayerPrefs.GetInt(velocityKey, 0) == 1;
51          invisibilidadComprada = PlayerPrefs.GetInt(invisibilityKey, 0) == 1;
52      }
53
54      // Método para acceder a la variable velocidadComprada desde otras clases si es necesario
55      public bool GetVelocidadComprada()
56      {
57          return velocidadComprada;
58      }
59
60      // Método para acceder a la variable invisibilidadComprada desde otras clases si es necesario
61      public bool GetInvisibilidadComprada()
62      {
63          return invisibilidadComprada;
64      }
65
66      public void VelocidadComprada()
67      {
68          velocidadComprada = true;
69          velocityBonus.UpdateVelocity();
```

```
70          SavePlayerBonus();
71      }
72
73      public void InvisibilidadComprada()
74      {
75          invisibilidadComprada = true;
76          BonusInvisibilityActive();
77          SavePlayerBonus();
78      }
79
80      private void BonusInvisibilityActive()
81      {
82          if (GetInvisibilidadComprada())
83          {
84              invisibilityBonusCanvas.SetActive(true);
85          }
86          else
87          {
88              invisibilityBonusCanvas.SetActive(false);
89          }
90      }
91
92      public void RestartBonus()
93      {
94          velocidadComprada = false;
95          invisibilidadComprada = false;
96          SavePlayerBonus();
97      }
98  }
```

MoneyManager script

```
1   using UnityEngine;
2
3   public class MoneyManager : MonoBehaviour
4   {
5       public static MoneyManager instance;
6
7       private int playerMoney = 0;
8       private string moneyKey = "PlayerMoney";
9
10      private void Awake()
11      {
12          // Verifica si ya existe una instancia del MoneyManager
13          if (instance == null)
14          {
15              // Si no existe, establece esta instancia como la instancia única
16              instance = this;
17              // Marca este GameObject como no destruible al cargar una nueva escena
18              DontDestroyOnLoad(gameObject);
19          }
20          else
```

```
        {
            // Si ya existe una instancia, destruye este GameObject para evitar duplicados
            Destroy(gameObject);
        }
    }

    private void Start()
    {
        LoadPlayerMoney();
    }

    public void AddMoney(int amount)
    {
        playerMoney += amount;
        SavePlayerMoney();
    }

    public void SubstractMoney(int amount)
    {
        playerMoney -= amount;
        SavePlayerMoney();
    }

    public void RestartMoney()
    {
        playerMoney = 0;
        SavePlayerMoney();
    }

    private void SavePlayerMoney()
    {
        PlayerPrefs.SetInt(moneyKey, playerMoney);
    }

    private void LoadPlayerMoney()
    {
        if (PlayerPrefs.HasKey(moneyKey))
        {
            playerMoney = PlayerPrefs.GetInt(moneyKey);
        }
    }

    public int GetPlayerMoney()
    {
        return playerMoney;
    }
}
```

restartScript script

```
using UnityEngine;

```

```
3  public class restartScript : MonoBehaviour
4  {
5      public MoneyManager moneyManager;
6      public BonusManager bonusManager;
7      public VelocityBonus velocityBonus;
8      public GameObject cuadroAviso;
9      public GameObject invisibilityBonusCanvas;
10
11     private void Start()
12     {
13         moneyManager = FindObjectOfType<MoneyManager>();
14         bonusManager = FindObjectOfType<BonusManager>();
15         velocityBonus = FindObjectOfType<VelocityBonus>();
16
17         Transform foundTransform = bonusManager.transform.Find("invisibilityBonusCanvas");
18         invisibilityBonusCanvas = foundTransform.gameObject;
19     }
20
21     private void OnTriggerStay()
22     {
23         if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
24         {
25             cuadroAviso.SetActive(true);
26         }
27     }
28
29     public void restartAll()
30     {
31         moneyManager.RestartMoney();
32         bonusManager.RestartBonus();
33         cuadroAviso.SetActive(false);
34         invisibilityBonusCanvas.SetActive(false);
35         velocityBonus.restartVelocity();
36     }
37
38     public void OnTriggerExit()
39     {
40         cuadroAviso.SetActive(false);
41     }
42 }
```

SafeBox script

```
1  using UnityEngine;
2  using TMPro;
3
4  public class SafeBox : MonoBehaviour
5  {
6      public GameObject safeInventory;
7      public TextMeshProUGUI money;
8      public MoneyManager moneyManager;
9
```

```
    void Start()
    {
        safeInventory.SetActive(false);
        moneyManager = FindObjectOfType<MoneyManager>();
    }

    private void OnTriggerStay()
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            safeInventory.SetActive(true);
            money.text = moneyManager.GetPlayerMoney().ToString() + "Zc";
        }
    }

    public void OnTriggerExit()
    {
        safeInventory.SetActive(false);
    }
}
```

ShopScript script

```
using UnityEngine;
using UnityEngine.UI;

public class ShopScript : MonoBehaviour
{
    public GameObject pcShop;
    public MoneyManager moneyManager;
    public BonusManager bonusManager;

    public Button velocidadButton;
    public GameObject checkMarkVelocidad;
    public int velocidadPrecio = 5000;

    public Button invisibilidadButton;
    public GameObject checkMarkInvisibilidad;
    public int invisibilidadPrecio = 10000;

    void Start()
    {
        pcShop.SetActive(false);
    }

    private void OnTriggerStay()
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            pcShop.SetActive(true);
            buttonState();
            if (bonusManager.GetVelocidadComprada()) checkMarkVelocidad.SetActive(true);
```

```
30              else checkMarkVelocidad.SetActive(false);
31              if (bonusManager.GetInvisibilidadComprada()) checkMarkInvisibilidad.SetActive(true);
32              else checkMarkInvisibilidad.SetActive(false);
33          }
34      }
35
36      public void OnTriggerExit()
37      {
38          pcShop.SetActive(false);
39      }
40
41      private void buttonState()
42      {
43          int money = moneyManager.GetPlayerMoney();
44          velocidadButton.interactable = money  >= velocidadPrecio &&
45          !bonusManager.velocidadComprada;
46          invisibilidadButton.interactable = money >= invisibilidadPrecio &&
47          !bonusManager.invisibilidadComprada;
48      }
49
50      public void ComprarVelocidad()
51      {
52          if (moneyManager.GetPlayerMoney() >= velocidadPrecio)
53          {
54              moneyManager.SubstractMoney(velocidadPrecio);
55              bonusManager.VelocidadComprada();
56              checkMarkVelocidad.SetActive(true);
57              buttonState();
58          }
59      }
60
61      public void ComprarInvisibilidad()
62      {
63          if (moneyManager.GetPlayerMoney() >= invisibilidadPrecio)
64          {
65              moneyManager.SubstractMoney(invisibilidadPrecio);
66              bonusManager.InvisibilidadComprada();
67              checkMarkInvisibilidad.SetActive(true);
68              buttonState();
69          }
70      }
71  }
```

InvisibilityBonus script

```
1   using UnityEngine;
2   using TMPro;
3
4   public class InvisibilityBonus : MonoBehaviour
5   {
6       public TextMeshProUGUI cooldownTimer;
7       public TextMeshProUGUI bonusTimer;
```

```
8
9       private float cooldownTime = 30f;
10      private float bonusTime = 5f;
11
12      private bool cooldownActive = false;
13      private bool bonusActive = false;
14
15      void Start()
16      {
17          // Desactivar los textos al inicio
18          cooldownTimer.gameObject.SetActive(false);
19          bonusTimer.gameObject.SetActive(false);
20      }
21
22      void Update()
23      {
24          // Verificar si se presiona la tecla F para iniciar el temporizador
25          if (!cooldownActive && Input.GetKeyDown(KeyCode.F))
26          {
27              IniciarCooldown();
28              IniciarBonusTimer();
29          }
30
31          if (cooldownActive)
32          {
33              ActualizarTemporizador(ref cooldownTime, cooldownTimer);
34
35              if (cooldownTime <= 0)
36              {
37                  cooldownTime = 0;
38                  cooldownActive = false;
39                  cooldownTimer.gameObject.SetActive(false);
40              }
41          }
42
43          if (bonusActive)
44          {
45              ActualizarTemporizador(ref bonusTime, bonusTimer);
46
47              if (bonusTime <= 0)
48              {
49                  bonusTime = 0;
50                  bonusActive = false;
51                  bonusTimer.gameObject.SetActive(false);
52              }
53          }
54      }
55
56      // Función para iniciar el temporizador de cooldown
57      void IniciarCooldown()
58      {
59          cooldownTime = 30f;
60          cooldownActive = true;
61          cooldownTimer.gameObject.SetActive(true);
```

```
62      }
63
64      // Función para iniciar el temporizador de bonificación
65      void IniciarBonusTimer()
66      {
67          bonusTime = 5f;
68          bonusActive = true;
69          bonusTimer.gameObject.SetActive(true);
70      }
71
72      // Función para actualizar un temporizador
73      void ActualizarTemporizador(ref float tiempoRestante, TextMeshProUGUI textoTemporizador)
74      {
75          tiempoRestante -= Time.deltaTime;
76          textoTemporizador.text = Mathf.RoundToInt(tiempoRestante).ToString();
77      }
78
79      public bool GetInvisibility()
80      {
81          return bonusActive;
82      }
83  }
```

VelocityBonus script

```
1   using UnityEngine;
2   using UnityEngine.SceneManagement;
3
4   public class VelocityBonus : MonoBehaviour
5   {
6       public BonusManager bonusManager;
7
8       public float newMoveSpeed = 3.45f;
9       public float initialMoveSpeed = 3f;
10
11      void Start()
12      {
13          // Registrar el callback cuando se carga una nueva escena
14          SceneManager.sceneLoaded += OnSceneLoaded;
15
16          // Comprueba si ya se compró la velocidad al iniciar
17          if (bonusManager.GetVelocidadComprada())
18          {
19              UpdateVelocity();
20          }
21      }
22
23      void OnDestroy()
24      {
25          // Desregistrar el callback cuando el objeto se destruye
26          SceneManager.sceneLoaded -= OnSceneLoaded;
27      }
```

```
28
29      // Método que se llama cuando una nueva escena se carga
30      void OnSceneLoaded(Scene scene, LoadSceneMode mode)
31      {
32          if (bonusManager.GetVelocidadComprada())
33          {
34              UpdateVelocity();
35          }
36      }
37
38      public void UpdateVelocity()
39      {
40          GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
41          PlayerMovement playerMovement = playerGameObject.GetComponent<PlayerMovement>();
42          playerMovement.moveSpeed = newMoveSpeed;
43      }
44
45      public void restartVelocity()
46      {
47          GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
48          PlayerMovement playerMovement = playerGameObject.GetComponent<PlayerMovement>();
49          playerMovement.moveSpeed = initialMoveSpeed;
50      }
51  }
```

floatImage script

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class floatingImage : MonoBehaviour
7   {
8       public GameObject floatImg; // Referencia al objeto FloatingImage
9
10      private void Start()
11      {
12          floatImg.SetActive(false); // Desactiva la imagen flotante al inicio
13      }
14
15      private void OnTriggerEnter()
16      {
17          floatImg.SetActive(true); // Activa la imagen flotante
18      }
19
20      private void OnTriggerExit()
21      {
22          floatImg.SetActive(false); // Desactiva la imagen flotante
23      }
24
25      private void OnTriggerStay()
```

```
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            floatImg.SetActive(false); // Desactiva la imagen flotante
        }
    }
}
```

CitySpawnManager script

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;

public class CitySpawnManager : MonoBehaviour
{
    public Vector3 cityPlayerStartPosition; // Posición de inicio en el último edificio
                                            // al que se ha accedido
    public Vector3 policePlayerStartPosition; // Posición de inicio en la policia

    public static CitySpawnManager instance;
    public TextMeshProUGUI moneyMessage = null;
    public MoneyManager moneyManager;

    private bool playerCaught = false;
    private string previousScene;

    void Awake()
    {
        // Verifica si ya existe una instancia del CitySpawnManager
        if (instance == null)
        {
            // Si no existe, establece esta instancia como la instancia única
            instance = this;
            // Marca este GameObject como no destruible al cargar una nueva escena
            DontDestroyOnLoad(gameObject);
            SceneManager.sceneLoaded += OnSceneLoaded;
        }
        else
        {
            // Si ya existe una instancia, destruye este GameObject para evitar duplicados
            Destroy(gameObject);
        }
    }

    void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Carga la posición de inicio dependiendo de la escena actual
        string currentSceneName = SceneManager.GetActiveScene().name;
        if (currentSceneName == "City" && previousScene != "Lair")
        {
```

```
43              // Si el jugador fue capturado, establece la posición de inicio en la policia
44              if (playerCaught)
45              {
46                  SetPlayerStartPosition(policePlayerStartPosition);
47                  playerCaught = false;
48              }
49              else
50              {
51                  SetPlayerStartPosition(cityPlayerStartPosition);
52                  FindMoneyMessage();
53                  MoneyMessage();
54              }
55          }
56          else
57          {
58              previousScene = currentSceneName;
59          }
60      }
61
62      public void SetPlayerStartPosition(Vector3 position)
63      {
64          // Mover al jugador a la posición especificada
65          GameObject player = GameObject.FindGameObjectWithTag("Player");
66          player.transform.position = position;
67      }
68
69      public void PlayerCaught()
70      {
71          playerCaught = true;
72      }
73
74      public void LoadCityScene()
75      {
76          SceneManager.LoadScene("City");
77      }
78
79      private void OnDestroy()
80      {
81          SceneManager.sceneLoaded -= OnSceneLoaded;
82      }
83
84      private void FindMoneyMessage()
85      {
86          GameObject moneyMessageObject = GameObject.FindGameObjectWithTag("MoneyMessage");
87          moneyMessage = moneyMessageObject.GetComponent<TextMeshProUGUI>();
88      }
89
90      private void MoneyMessage()
91      {
92          AssignMessage();
93          AssignMoney();
94          moneyMessage.gameObject.SetActive(true);
95          Invoke("FadeAndDeactivateMoneyMessage", 5f); // Invocar el método después de 5 segundos
96      }
```

```
97
98      private void FadeAndDeactivateMoneyMessage()
99      {
100         // Asumiendo que tienes un componente CanvasGroup en moneyMessage
101         // para controlar la transparencia
102         CanvasGroup canvasGroup = moneyMessage.GetComponent<CanvasGroup>();
103
104         // Iniciar el proceso de desvanecimiento utilizando una corrutina
105         StartCoroutine(FadeOut(canvasGroup, 1f)); // Duración de 1 segundo
106     }
107
108     private IEnumerator FadeOut(CanvasGroup canvasGroup, float duration)
109     {
110         float currentTime = 0f;
111         float startAlpha = canvasGroup.alpha;
112         float targetAlpha = 0f; // Desvanecer completamente
113
114         while (currentTime < duration)
115         {
116             currentTime += Time.deltaTime;
117             canvasGroup.alpha = Mathf.Lerp(startAlpha, targetAlpha, currentTime / duration);
118             yield return null;
119         }
120
121         canvasGroup.alpha = targetAlpha;
122
123         // Desactivar el mensaje después de desvanecerlo
124         moneyMessage.gameObject.SetActive(false);
125     }
126
127     private void AssignMessage()
128     {
129         if (previousScene == "Office") moneyMessage.text =
130         "Has␣vendido␣el␣USB␣a␣unos␣Señores␣Naturalmente␣Inteligentes.
131 ␣␣␣␣␣␣␣␣Has␣ganado␣1000Zc␣que␣han␣sido␣depositados␣en␣tu␣caja␣fuerte";
132     }
133
134     private void AssignMoney()
135     {
136         if (previousScene == "Office") moneyManager.AddMoney(1000);
137         Debug.Log(moneyManager.GetPlayerMoney());
138     }
139
140 }
```

StateMachine script

```
1 using System.Collections;
2 using UnityEngine;
3 using UnityEngine.AI;
4
5 public class StateMachine : MonoBehaviour
```

```
6  {
7      public enum State
8      {
9          Normal,
10         Investigate,
11         Wait,
12         Chase
13     }
14
15     public Transform[] waypoints;
16     public float secondsWaiting = 5f;
17     public float chaseSpeed = 5f;
18
19     private State currentState;
20
21     private NavMeshAgent navMeshAgent;
22     private int currentWaypointIndex;
23     private Coroutine currentStateCoroutine;
24
25     //Invisibility bonus
26     public BonusManager bonusManager;
27     public InvisibilityBonus invisibilityBonusCanvas;
28
29     void Start()
30     {
31         currentState = State.Normal;
32         currentWaypointIndex = 0;
33         navMeshAgent = GetComponent<NavMeshAgent>();
34         navMeshAgent.SetDestination(waypoints[currentWaypointIndex].position);
35
36         StartCoroutine(FSM());
37         Debug.Log(currentState);
38
39         //Invisibility bonus
40         bonusManager = FindObjectOfType<BonusManager>();
41         invisibilityBonusCanvas = bonusManager.GetComponent<InvisibilityBonus>();
42     }
43
44     IEnumerator FSM()
45     {
46         while (true)
47         {
48             yield return StartCoroutine(currentState.ToString());
49         }
50     }
51
52     IEnumerator Normal()
53     {
54         while (currentState == State.Normal)
55         {
56             Patrol();
57             yield return null;
58         }
59     }
```

```
60
61      IEnumerator Investigate()
62      {
63          while (currentState == State.Investigate)
64          {
65              InvestigateLogic();
66              yield return null;
67          }
68      }
69
70      IEnumerator Wait()
71      {
72          float timer = secondsWaiting;
73          while (currentState == State.Wait)
74          {
75              timer -= Time.deltaTime;
76              if (timer <= 0)
77              {
78                  ChangeState(State.Normal);
79              }
80              else
81              {
82                  // Cambiar a la animación "looking"
83                  GetComponent<Animator>().Play("looking");
84              }
85              yield return null;
86          }
87
88          // Restaurar la animación original al salir del estado de espera
89          GetComponent<Animator>().Play("walking");
90      }
91
92      IEnumerator Chase()
93      {
94          navMeshAgent.speed = chaseSpeed;
95          while (currentState == State.Chase)
96          {
97              ChasePlayer();
98              yield return null;
99          }
100     }
101
102     void Patrol()
103     {
104         if (!navMeshAgent.pathPending && navMeshAgent.remainingDistance
105         <= navMeshAgent.stoppingDistance)
106         {
107             currentWaypointIndex = (currentWaypointIndex + 1) % waypoints.Length;
108             navMeshAgent.SetDestination(waypoints[currentWaypointIndex].position);
109         }
110     }
111
112     void InvestigateLogic()
113     {
```

```
114         if (!navMeshAgent.pathPending && navMeshAgent.remainingDistance
115         <= navMeshAgent.stoppingDistance)
116         {
117             ChangeState(State.Wait);
118         }
119     }
120 
121     private void OnEnable()
122     {
123         CameraDetection.OnPlayerDetected += HandlePlayerDetected;
124     }
125 
126     private void OnDisable()
127     {
128         CameraDetection.OnPlayerDetected -= HandlePlayerDetected;
129     }
130 
131     private void HandlePlayerDetected(Vector3 playerPosition)
132     {
133         ChangeState(State.Investigate);
134         navMeshAgent.SetDestination(playerPosition);
135     }
136 
137     void ChasePlayer()
138     {
139         GameObject player = GameObject.FindGameObjectWithTag("Player");
140         if (player != null)
141         {
142             navMeshAgent.SetDestination(player.transform.position);
143         }
144     }
145 
146     void ChangeState(State newState)
147     {
148         if (currentState == newState) return;
149 
150         currentState = newState;
151 
152         if (currentStateCoroutine != null)
153         {
154             StopCoroutine(currentStateCoroutine);
155         }
156 
157         currentStateCoroutine = StartCoroutine(currentState.ToString());
158     }
159 
160     void OnTriggerEnter(Collider other)
161     {
162         if (currentState != State.Chase && other.gameObject.tag == "Player" &&
163         !invisibilityBonusCanvas.GetInvisibility())
164         {
165             ChangeState(State.Chase);
166         }
167     }
```

```
}
```

SecurityCameras script

```
using UnityEngine;

public class SecurityCameras : MonoBehaviour
{
    // Ángulo mínimo y máximo en el eje Y
    public float minYAngle = 0f;
    public float maxYAngle = 360f;

    // Velocidad de rotación en grados por segundo
    public float rotationSpeed = 30f;

    void FixedUpdate()
    {
        // Calcular el ángulo de rotación en el eje Y basado en el tiempo
        float yAngle = Mathf.Lerp(minYAngle, maxYAngle,
        Mathf.PingPong(Time.time * rotationSpeed, 1f));

        // Aplicar la rotación al objeto solo en el eje Y
        transform.rotation = Quaternion.Euler(0f, yAngle, 0f);
    }
}
```

CameraDetection script

```
using System;
using UnityEngine;

public class CameraDetection : MonoBehaviour
{
    // Delegado para el evento de detección de jugador
    public static event Action<Vector3> OnPlayerDetected;

    //Invisibility bonus
    public BonusManager bonusManager;
    public InvisibilityBonus invisibilityBonusCanvas;

    void Start()
    {
        //Invisibility bonus
        bonusManager = FindObjectOfType<BonusManager>();
        invisibilityBonusCanvas = bonusManager.GetComponent<InvisibilityBonus>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player") && !invisibilityBonusCanvas.GetInvisibility())
```

```
        {
            Debug.Log("Detectado");
            // Obtener la posición del jugador
            Vector3 playerPosition = other.transform.position;
            // Notificar a todos los suscriptores del evento
            OnPlayerDetected?.Invoke(playerPosition);
        }
    }
}
```

GameEnding script

```
using UnityEngine;

public class GameEnding : MonoBehaviour
{
    public float fadeDuration = 1f;
    public float displayImageDuration = 3f;
    public GameObject player;
    public CanvasGroup caughtBackgroundImageCanvasGroup;
    public CitySpawnManager citySpawnManager;

    float m_Timer;
    bool playerCaught = false;

    //Invisibility bonus
    public BonusManager bonusManager;
    public InvisibilityBonus invisibilityBonusCanvas;

    void Start()
    {
        citySpawnManager = FindObjectOfType<CitySpawnManager>();

        //Invisibility bonus
        bonusManager = FindObjectOfType<BonusManager>();
        invisibilityBonusCanvas = bonusManager.GetComponent<InvisibilityBonus>();
    }

    private void Update()
    {
        if (playerCaught) EndLevel();
    }

    void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Player" && !invisibilityBonusCanvas.GetInvisibility())
        {
            playerCaught = true;
        }
    }

    void EndLevel()
```

```
    {
        m_Timer += Time.deltaTime;
        caughtBackgroundImageCanvasGroup.alpha = m_Timer / fadeDuration;

        if (m_Timer > fadeDuration + displayImageDuration)
        {
            citySpawnManager.PlayerCaught();
            citySpawnManager.LoadCityScene();
        }
    }
}
```

GuideScript script

```
using UnityEngine;

public class GuideScript : MonoBehaviour
{
    public GameObject guia;
    public GameObject icono;

    private void Start()
    {
        // Desactivar la guía y activar el icono al incio
        guia.SetActive(false);
        icono.SetActive(true);
    }
    void Update()
    {
        // Verificar si se presiona o se suelta la tecla Tabulador
        if (Input.GetKeyDown(KeyCode.Tab))
        {
            // Activar la guía y desactivar el icono al presionar la tecla Tabulador
            guia.SetActive(true);
            icono.SetActive(false);
        }
        else if (Input.GetKeyUp(KeyCode.Tab))
        {
            // Desactivar la guía y activar el icono al soltar la tecla Tabulador
            guia.SetActive(false);
            icono.SetActive(true);
        }
    }
}
```

RandomObjectVisibility script

```
using UnityEngine;

public class RandomObjectVisibility : MonoBehaviour
```

```
{
    public GameObject[] objetos;

    void Start()
    {
        // Oculta todos los objetos al inicio
        foreach (GameObject objetos in objetos)
        {
            objetos.SetActive(false);
        }

        // Escoge un objeto aleatorio para volverlo visible
        int indexObjetoVisible = Random.Range(0, objetos.Length);
        objetos[indexObjetoVisible].SetActive(true);
    }
}
```

SetVisibility script

```
using UnityEngine;

public class SetVisibility : MonoBehaviour
{
    public GameObject objeto;

    void Start()
    {
        objeto.SetActive(false);
    }

    private void OnTriggerStay()
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            objeto.SetActive(true);
        }
    }

    public void OnTriggerExit()
    {
        objeto.SetActive(false);
    }
}
```

ItemCollector script

```
using UnityEngine;

public class ItemCollector : MonoBehaviour
{
```

```
    public GameObject itemImage; // Imagen del objeto en el canvas
    public GameObject door;
    public GameObject exit = null;

    public bool lastObject = false; // Bool para saber si es el objeto que completa el nivel

    private void Start()
    {
        itemImage.SetActive(false);
        exit.SetActive(false);
    }

    private void OnTriggerStay()
    {
        if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
        {
            gameObject.SetActive(false);
            itemImage.SetActive(true);
            rotateDoor();
            levelCompleted();
        }
    }

    private void rotateDoor()
    {
        if (door != null)
        {
            // Girar el objeto 90 grados alrededor del eje Y
            door.transform.Rotate(0, 90, 0);
        }
    }

    private void levelCompleted()
    {
        if (lastObject && exit != null)
        {
            // Activar salida
            exit.SetActive(true);
        }
    }
}
```

CodeScript script

```
using TMPro;
using UnityEngine;

public class CodeScript : MonoBehaviour
{
    // Referencia al componente TextMeshPro
    public TextMeshProUGUI code;
    public GameObject note;
```

```
9
10      void Start()
11      {
12          // Generar cuatro números aleatorios entre 0 y 9 y asignarlos como texto
13          string randomNumbers = GenerateRandomNumbers();
14          code.text = randomNumbers;
15          note.SetActive(false);
16      }
17
18      // Método para generar cuatro números aleatorios como texto
19      string GenerateRandomNumbers()
20      {
21          string numbers = "";
22          for (int i = 0; i < 4; i++)
23          {
24              // Generar un número aleatorio entre 0 y 9 y agregarlo al texto
25              int randomNumber = Random.Range(0, 10);
26              numbers += randomNumber.ToString();
27          }
28          return numbers;
29      }
30
31      private void OnTriggerStay()
32      {
33          if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
34          {
35              note.SetActive(true);
36          }
37      }
38
39      public void OnTriggerExit()
40      {
41          note.SetActive(false);
42      }
43  }
```

KeyPadScript script

```
1  using TMPro;
2  using UnityEngine;
3
4  public class KeyPadScript : MonoBehaviour
5  {
6      string code;
7      string padCode = null;
8      int padCodeIndex = 0;
9
10     public GameObject door;
11     public GameObject padTrigger;
12     public TextMeshProUGUI padCodeText;
13     public TextMeshProUGUI codeText;
14
```

```
15      private void Start()
16      {
17          code = codeText.text;
18          Debug.Log(code);
19      }
20
21      public void CodeFunction(string Numbers)
22      {
23          if(padCodeIndex < 4)
24          {
25              padCodeIndex++;
26              padCode = padCode + Numbers;
27              padCodeText.text = padCode;
28          }
29      }
30      public void Enter()
31      {
32          if (padCode == code)
33          {
34              RotateDoor();
35              padTrigger.SetActive(false);
36              gameObject.SetActive(false);
37          }
38      }
39      public void Delete()
40      {
41          padCodeIndex = 0;
42          padCode = null;
43          padCodeText.text = padCode;
44      }
45
46      public void RotateDoor()
47      {
48          if (door != null)
49          {
50              // Girar el objeto 90 grados alrededor del eje Y
51              door.transform.Rotate(0, 90, 0);
52          }
53      }
54  }
```

exitManagment script

```
1   using UnityEngine;
2
3   public class exitManagment : MonoBehaviour
4   {
5       public CitySpawnManager citySpawnManager;
6
7       private void Start()
8       {
9           // Buscar automáticamente una instancia de CitySpawnManager en la escena actual
```

```
10          citySpawnManager = FindObjectOfType<CitySpawnManager>();
11      }
12
13      private void OnTriggerStay()
14      {
15          if (Input.GetKey(KeyCode.E)) // Comprueba que se pulsa la tecla E
16          {
17              citySpawnManager.LoadCityScene(); // Carga la escena indicada
18          }
19      }
20  }
```