Design and Development of an Online Multiplayer Videogame

Adrián Mon Maroto

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

July 3, 2023

Supervised by: Carlos González Ballester, PhD.

Acknowledgments

First of all, I would like to thank my family and friends for their support and help in developing and testing my Final Degree Work.

I also would like to thank my Final Degree Work supervisor, Carlos González Ballester, PhD, for helping me to decide the topic of this work and for his support and recommendations whenever I got stuck.

And 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

This document presents a video game called Fixing It, which is a multiplayer and arcade game where the player takes control of a mechanic and together with his/her co-workers, the other players in the game, he/she must fix as many objects as possible before his working day is over.

This document consists in the development of a multiplayer video game developed with Unity3D and focused for Windows. This has been done exploring the new Unity´s Netcode library next to the Unity´s Lobby and Relay libraries to create a lobby system and using this in turn to create an online mini-game.

In addition to this, the Observer Pattern is used throughout the development of the project in order to allow a more orderly and independent programming and is used to separate the UI logic from the internal logic of the game.

Keywords

Unity
Multiplayer
Lobby
Observer Pattern

Contents

Chapter 1

Introduction

Contents

This chapter shows what the purpose of the work was in the beginning, why and how this project was going to be developed.

1.1 Work Motivation

In the Universtat Jaume I the students of Video Game Design and Development never learn how to create a multiplayer game and it is also something that I have never faced before, so I see it as my own challenge.

I see it as an opportunity to learn the common logic in all the multiplayer minigames and to learn how to use the new Unity´s multiplayer librareis; Netcode for Gameobjects [9] , Unity Lobby Service [7] and Unity Relay [8].

Moreover, I was motivated to do so because there were information channels that were already investigating the new Unity multiplayer libraries, such as Code Monkey [3] or Tarodev [4] , so I could use them as references when developing the project.

Another of my motivations was to improve my code architecture and to investigate the use of the Observer Pattern [1] for this purpose.

[1]Go to A.1 to see the definition of the Observer Pattern

My main goal is to make the cleanest and most abstract code possible to be able to reuse it in other projects and at the same time to force myself to improve in terms of code design. The art is secondary.

I also wanted to do this work so that anyone who wants to try to do some multiplayer programming with Unity has some extra reference since it uses relatively new libraries.

1.2 Objectives

The objectives to be achieved are directly related to work motivation:

- **Create a Lobby and Relay System:** Create an abstract implementation of the Lobby and Relay Systems so that they can be recycled in any game.
- **Develop of a mini-game by applying the Observer Pattern:** Use the Observer Pattern to create cleaner and more scalable code.
- **Adapt the mini-game to multiplayer:** Use the Netcode for GameObjects Library to sync every global object and make the server in charge of important functions.

1.3 Environment and Initial State

The idea for the game started in October 2022, after a meeting with my Final Degree Work supervisor in which there was a debate about what to do. In the end he recommended me to make a multiplayer game, which I decided to do because it was something that wasn't done during my degree and because knowing the architecture behind multiplayer programming is highly valued by companies when it comes to hiring people, as it is something that only a minority of people do. In turn told me that I could apply some design pattern, and I chose the Observer Pattern because it is one that I had always wanted to start mastering due to the separation of classes that it allows when programming and that by applying it I would force myself to improve my code design.

After the meeting and doing an introductory GDD of what the game was going to be like I was able to start the project as such the first week of March 2023. This is because I was on Erasmus during the first semester of the academic year 2022/2023 and until I finished all the Erasmus exams and papers and did all the necessary bureaucracy I didn't have time to start it.

When I started in March I organised myself in such a way that when I had free time after my internship and classes I would spend from 16.00 to 22.00 researching and working on the project. It has been a routine that I have managed to keep to, except for occasional days or holidays.

Due to time constraints, the new Unity libraries having strange bugs and the fact that it was a one-man job, I focused on programming the game and most of the artwork is placeholder.

Chapter 2

Planning and resources evaluation

Contents

This chapter explains the technical part of the work, the planning and all the resources used.

2.1 Planning

This section is going to explain how the tasks were divided in time. Not all tasks were dependent on each other, i.e. some tasks might not be completed so that others could be done. The planning is the following:

- **Search for information about Unity's multiplayer libraries (40 hours):** Search the official Unity documentation, public forums and websites of people working with Unity as a professional tool.
- **Creation of the initial menu of the game (6 hours):** Create a basic menu where the player can choose whether to play; enter the options menu to adjust the display and sounds; or exit the game.
- **Creation of a Lobby and Relay System (100 hours):** Create another menu where the player can decide whether to join one of the listed public lobbies; join a private lobby through the internal code of that lobby; create a lobby, both public and private; or whether to return to the start menu. Internally it will use Unity's

Relay system to allow you to connect to a host by its IP and port; or to be the host and open a port.

- **Creation of the mini-game (50 hours):** Create the game mechanics based on those of the GDD and merge them to create a demo level.
- **Adaptation of the mini-game to multiplayer (74 hours):** Use the new Unity's Netcode for GameObjects library to dynamically synchronise and/or clone instantiated objects across all users in the game session. Such as player prefabs or objects that later in the game will be used by all people and need their positions and internal states to be updated for everyone.
- **Writing the final report (20hours):** Write the report, find images to link to it and correct minor errors in the file.
- **Preparation of FDW presentation (10 hours):** Prepare the slides used in the final presentation and an accompanying script.

The division of tasks can be seen in Figure 2.1

2.2 Resource Evaluation

The resources used for this project were:

- **Electricity:** Calculating the 300 hours worked and the use of a MSI GL63 8SD as a professional computer, with an average consumption of 180W and the average cost of electricity during the months of March, April, May and June, we estimate an electricity cost of around 4.32€.
- **Salary:** Also based on the salary of a junior software programmer in Spain, which is around 10-12€ per hour, it would have cost around 3300€.
- **Unity license:** Free of charge due to educational licensing.
- **Visual Studio 2022:** Free of charge since the Community version has been used.
- **Krita:** Free of charge due to is an Open Source tool.
- **Blender:** Free of charge due to is an Open Source tool.
- **External assets:** All of the external assets used in this project were free assets.

All these results in the sum of **3304.32€** as the costs of all the work.

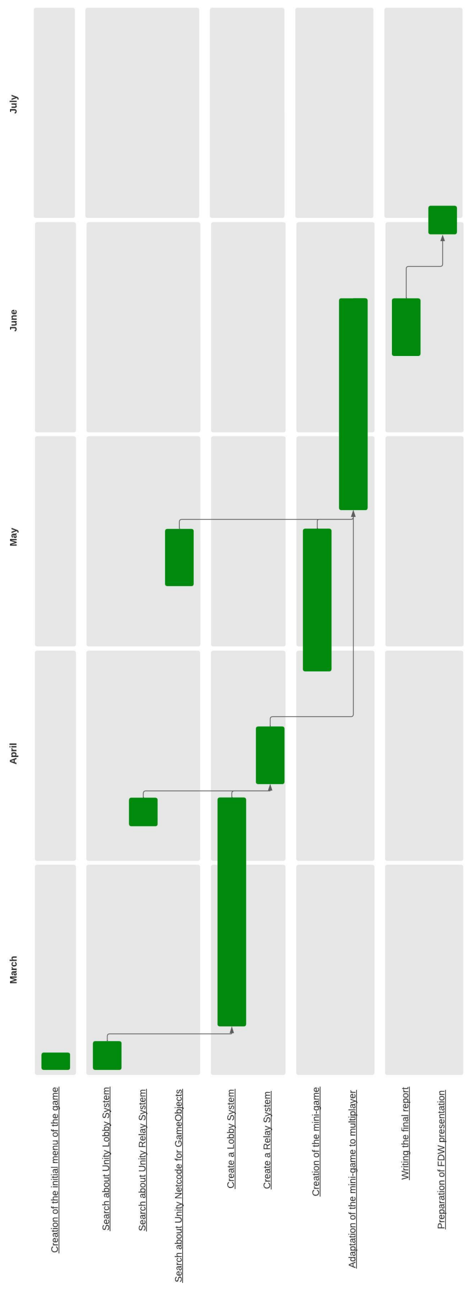


Figure 2.1: Gantt chart (made with lucid)

CHAPTER 3

GAME DESIGN DOCUMENT

Contents

3.1 Conceptualization

Title: Fixing It

Platforms: Windows

Game Summary: This game will be placed in a level that simulates a mechanical workshop where the players work. They have to fix objects that NPCs will bring to the workshop because they are broken. To do so, they will have to collect the necessary pieces scattered in different parts of the room to craft the tools, which will break after a set number of uses, and use them to fix the objects.

Similar Games: A game with a similar gameplay could be *Overcooked*.

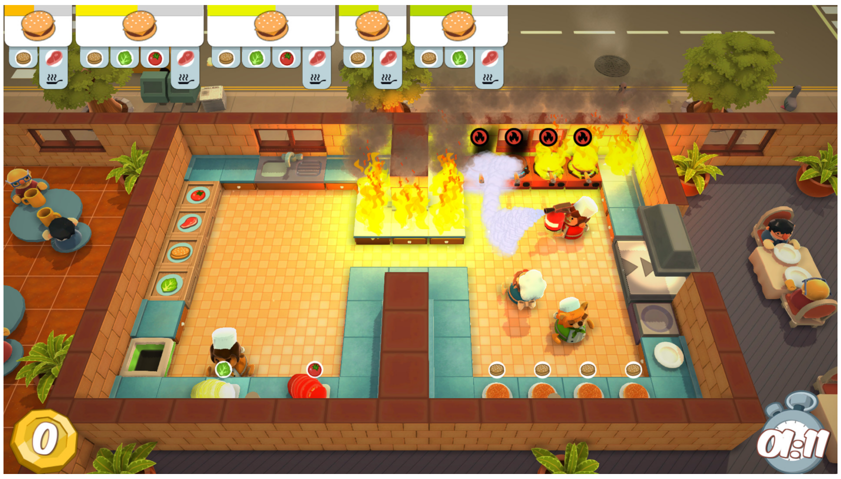


Figure 3.1: Overcooked gameplay

3.2 Demography

Age: Due to its future cute graphics and its absence of swear words and explicit elements this game is a game suitable for any player over 3 years of age. Although its complexity in coordinating with your peers and understanding what is going on, it is a game aimed at a slightly more logical thinking audience, ranging from pre-teens to young adults.

Genre: The game can be classified into three distinct genres: *Arcade* due to being designed in such a way that the player can jump in, have a couple of quick games and leave; *Puzzle* as the player must use a bit of logic to think of the most efficient way to fix the objects in the limited time; and *Multiplayer* as it is a game with the ability to connect with up to 3 people online to play together for the highest score.

Types of players: This game is for 2 types of players: casual gamers who simply want to unwind for half an hour a day and have a couple of quick games with their friends; and competitive gamers who will compete to get the highest possible scores in the game.

3.3 Rules

3.3.1 Design decisions

Objective: The main objective of the game is to fix as many objects as possible in the given time.

Difficulty: The difficulty of the game depends on the player. A novice player will not know the recipes for the tools so he will have to spend time going to the recipe book, i.e. he will have less time to fix items and it will be more difficult to get a high score. Meanwhile, an experienced player will be able to skip the step of looking at the recipe book, making it easier to get a high score.

Variety: This first version of the game will have only 3 objects to fix with a total of 3 tools and 5 different pieces. This has been designed to have a minimum to test the mechanics of the game and see if they are fun to keep expanding it or not.

Complexity: The only complexities for the players will be to be aware of which ojbect to fix is for which NPC; to be aware of the remaining uses of each tool before it breaks; and to manage the space available in the room for the pieces, tools and objects to fix.

Limits: The only limit is spatial, the player will not be able to leave his working room.

3.3.2 Modifiable rules

Before starting the mini-game, players can choose the colour of their character from a list of pre-set colours, as long as that colour is not chosen by another player.

3.3.3 Operational rules

The player will be able to do a total of 3 different actions:

- The player controls a character that can move along the XZ axes.
- The character can grab or release one of the different types of objects at a time when he is at a minimum distance away from the counter containing the object.
- The character can also interact at the same distance with the counters for a special use. This allows them to create tools, review crafting recipes or fix broken objects.

3.3.4 Foundational rules

These are the technical rules that spicify the behaviours of the operational rules:

- The players will have a constant speed *moveSpeed* which will allow them to move around the room. This movement will in turn rotate the character on the Y-axis at a constant *rotationSpeed* to align it with the direction it is moving in.
- When interacting with objects of any type to grab or release objects or to perform alternate interaction, a raycast will be launched from the base of the controlled character in the direction the character is facing and will have a range of *interact-Distance*. If a collision with any type of counter occurs, the counter will internally decide what action to take.

3.3.5 Written rules

There will be an optional menu in the main menu called *Controls*, where the player can see the different controls to understand how to play.

3.3.6 Advisory rules

The counters will show a local outline, i.e. they will only be displayed to the player who is playing rather than to the entire gaming room, which will be displayed when the player is inside *interactDistance* to indicate that they will interact with that object.

3.4 Mechanics

3.4.1 Actions

- **Movement:** The player will have a static camera that will show the whole level. He/She can move across the XZ axes in the level and the character will turn to face that direction.
- **Grab/Release objects:** When the character is facing a counter and inside a minimum range, it can grab the object that is on that counter only if it is not grabbing another object. At the same time it can release the object onto that counter only if the counter does not contains an object.
- **Craft objects:** The player has to release from 1 to 3 pieces in the Tool Counter. After doing it he/she can do the alternate interaction to let the Tool Counter do the crafting logic. If the objects in the tool counter belong to a recipe the Tool Counter will output the tool onto itself so the character can grab it.
- **Fix objects:** An object to fix needs to be onto some counter. After that if a controlled character tries to alternate interact with that counter it will check if he/she has one of the correct tools that the broken object needs to be fixed. If

that happens, the tool will be used and will check if it has more uses or if it will broke and disappear. If the object to fix does not need more tools, it will be ready to return to the NPC that asked about the repair.

- **Check recipe book:** The recipe book is on another special counter, so the player needs to alternate interact with it. The first column will show an image of the object to be created and the second, third and fourth columns of each page will show an image of the objects to be fixed. If the level has more tools than the possible ones to be shown in the 2 pages of the book, the interface will show the user buttons to go to the next or previous page and see the rest of the recipes. See Figure 3.3.

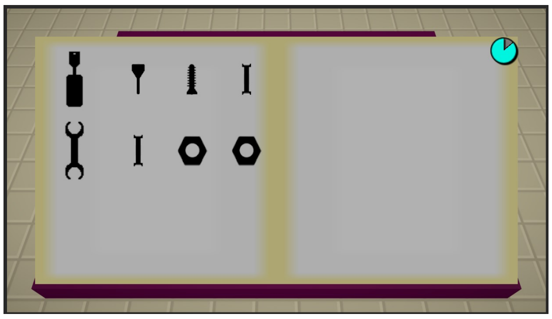


Figure 3.2: Recipe Book

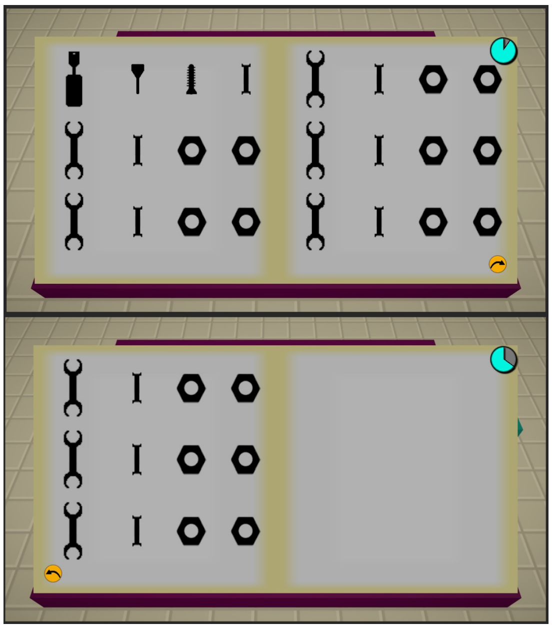


Figure 3.3: Recipe Book with a third page

3.4.2 Entities

- **Pieces:** One of the different objects that the player can grab or release. It is used to craft tools in the ToolCreatorCounter.
- **Tools:** The second type of object that the player can grab or release. It is used fix the objects to fix and can break after a certain number of uses.
- **Objects to Fix:** The last type of object that the player can grab or release. It is left by an NPC at the CustomerCounter for the player to pick up and fix using the necessary tools, these are shown above it with the pictorial representation of the tools.

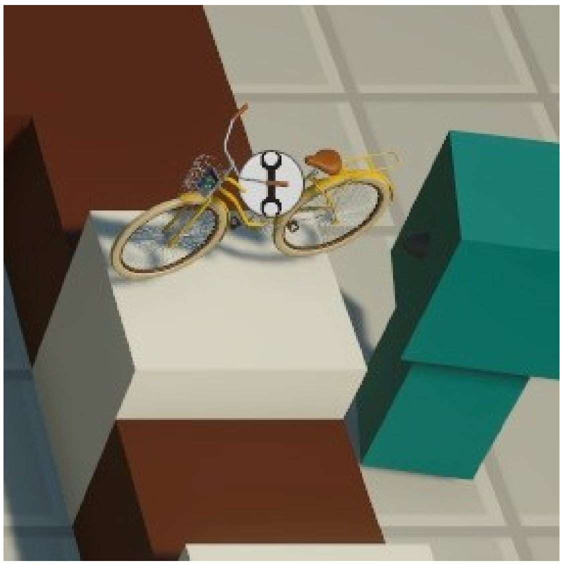


Figure 3.4: Object to Fix in CustomerCounter

- **NPC:** They are the customers who give the Objects to Fix to the CustomerCounters so the player gan grab and fix them.
- **TableCounter:** An empty counter where the player can grab or release objects and try to fix the objects that are on it. See Figure 3.5
- **ToolCreatorCounter:** A counter where the player can release up to 3 pieces and an alternate interact will consume those objects to try to create a tool. If the combination of those objects returns a tool, it will be created and ready to be grabbed by a player. See Figure 3.6
- **CustomerCounter:** A counter where a NPC will be waiting for a player to return its fixed item. In case there is no NPC assigned to this counter yet, it will be useless. See Figure 3.7

- **PiecesCounter:** Every time a player tries to interact with it while not grabbing an object, the counter will generate the piece related to itself and give it to the player. See Figure 3.8
- **ManualCounter:** A counter where the player can check the recipes of the tools after doing an alternate interaction. See Figure 3.9

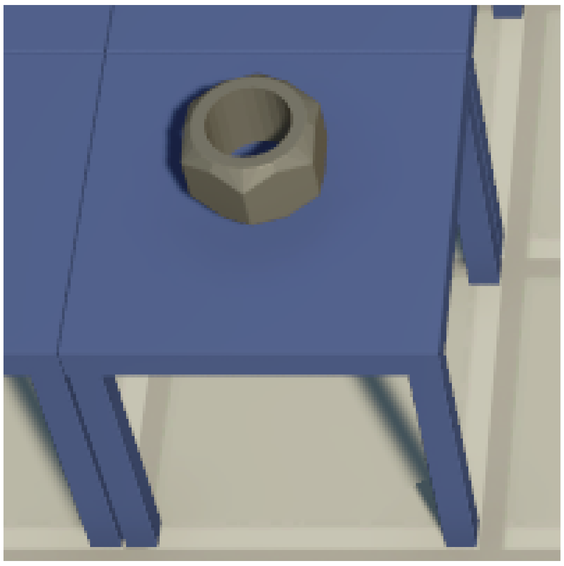


Figure 3.5: TableCounter with a piece on top

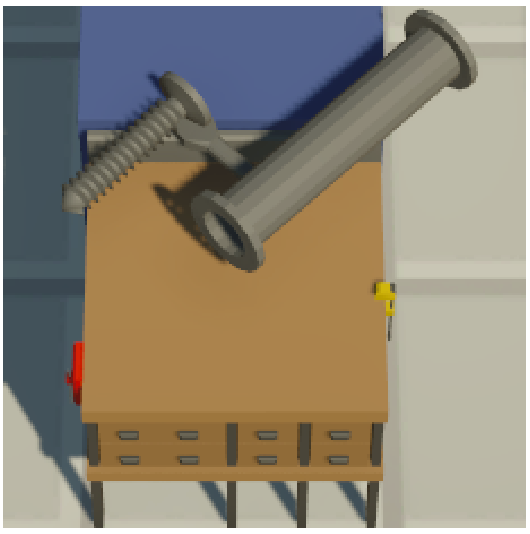


Figure 3.6: ToolCreatorCounter ready to create a tool

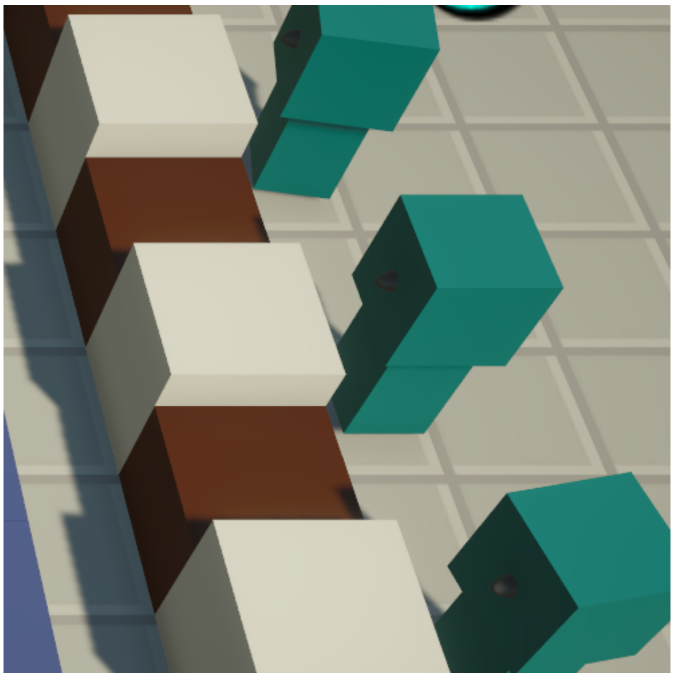


Figure 3.7: CustomerCounters with customers waiting for the fixed object

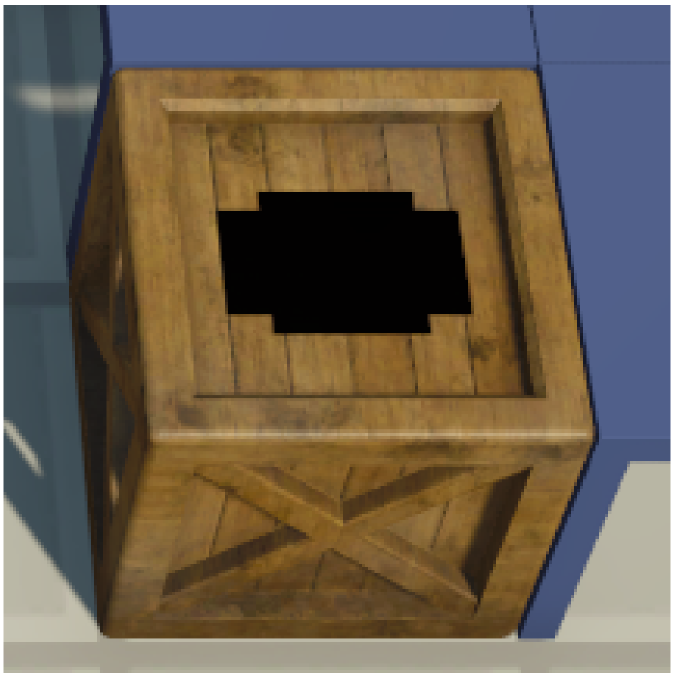


Figure 3.8: PiecesCounter

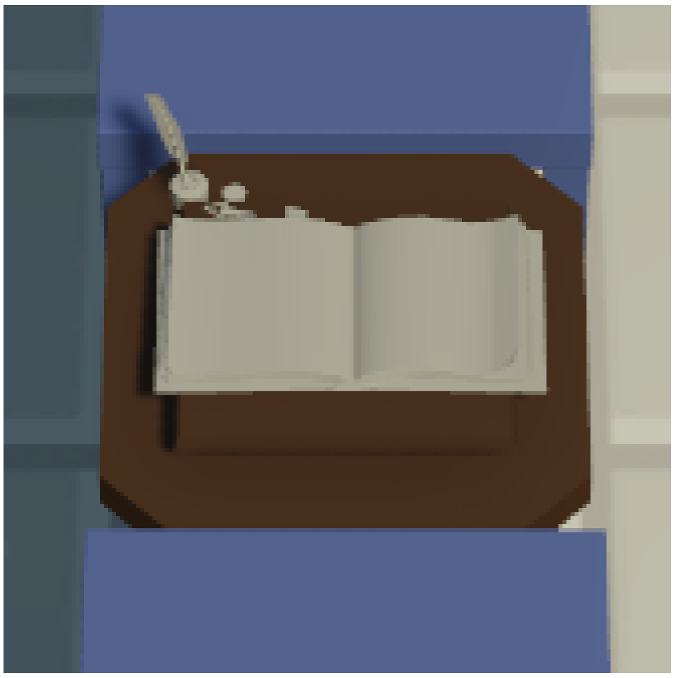


Figure 3.9: ManualCounter

3.5 Input Interface

The player will be able to use keyboard as well as any standard controller. There are 2 different inputs, one for gameplay (see Table 3.1) and one for navigation in any kind of menu or interface.

	KEYBOARD	**CONTROLLER**
Move	WASD	Left Stick
Interact	Space	Button South
Alternate Interact	E	Button West

Table 3.1: Gameplay controls

	KEYBOARD	**CONTROLLER**
Submit	Space/Enter	Button South
Cancel	Escape	Button East
Navigation	WASD/↑←↓→	Left Stick/D-Pad

Table 3.2: Menu Navigation controls

3.6 User interface

3.6.1 Flow structure

The navigation between the different menus of the game can be seen in the following figure 3.10:

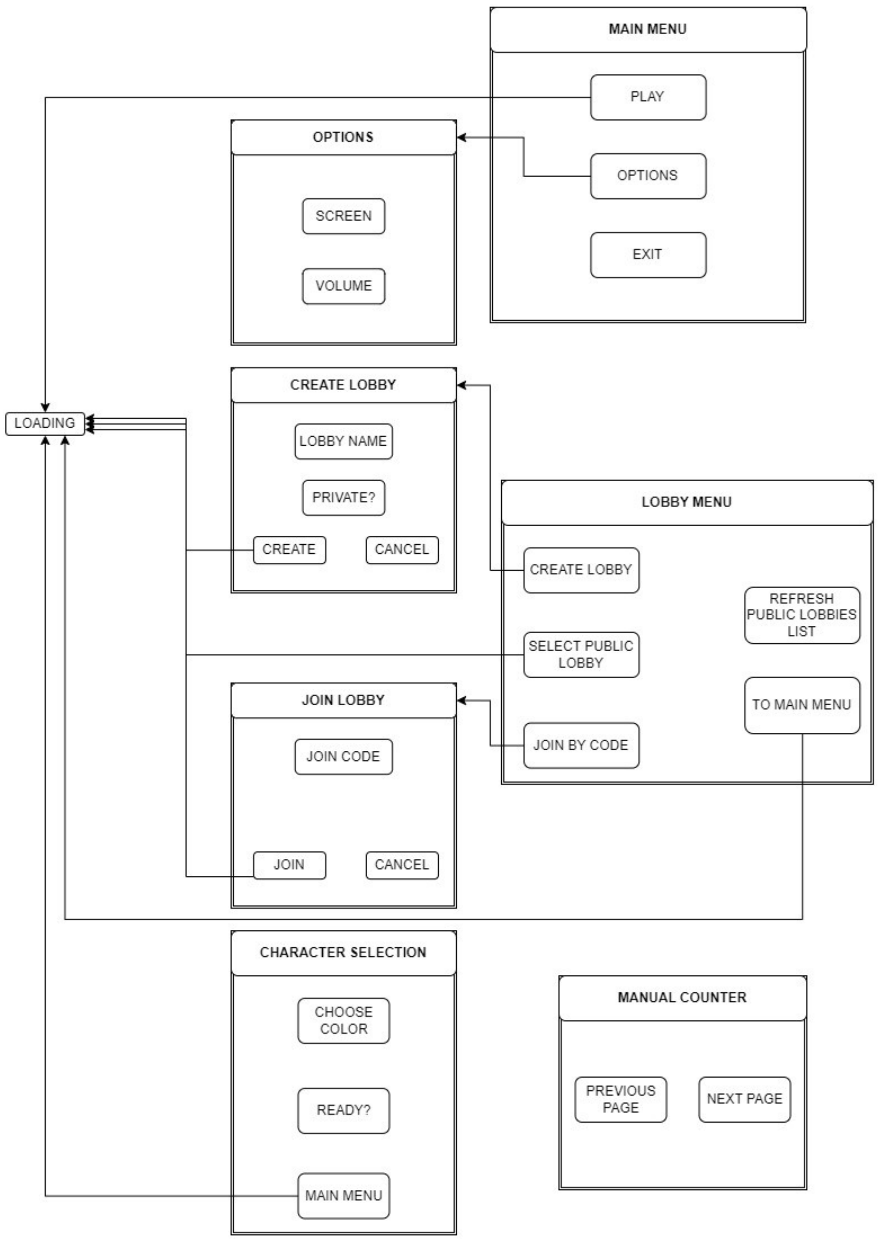


Figure 3.10: Menus Flow Structure

3.6.2 Some menu designs

These images are examples of the different menus throughout the game, such as: the Main Menu, Figure 3.11; the Options Menu, Figure 3.12; the Options Menu with the submenu for Screen Settings, Figure 3.13, or Volume Settings, Figure 3.14; the Lobby Menu, Figure 3.15; or the Character Selection Menu, Figure 3.16.

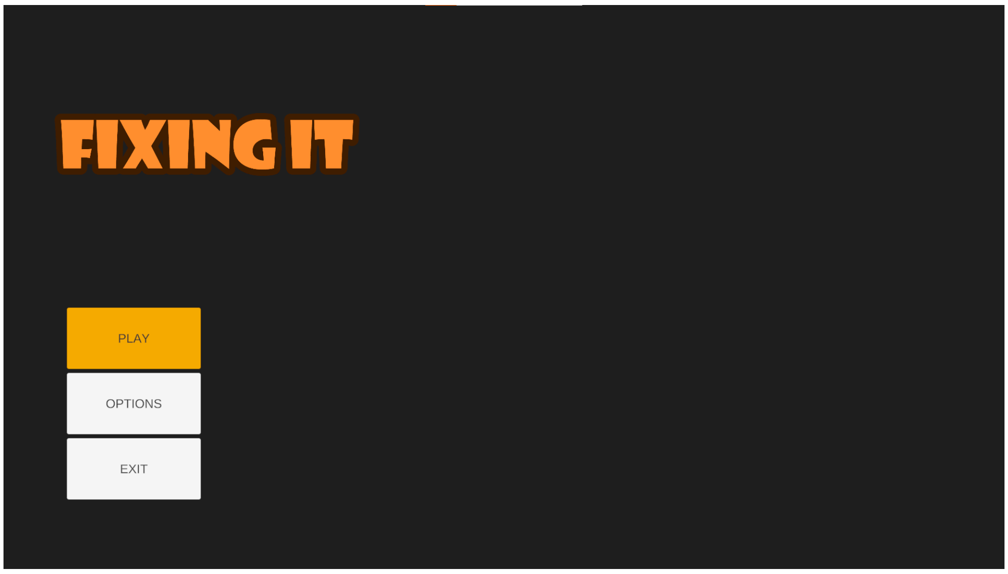


Figure 3.11: Main Menu

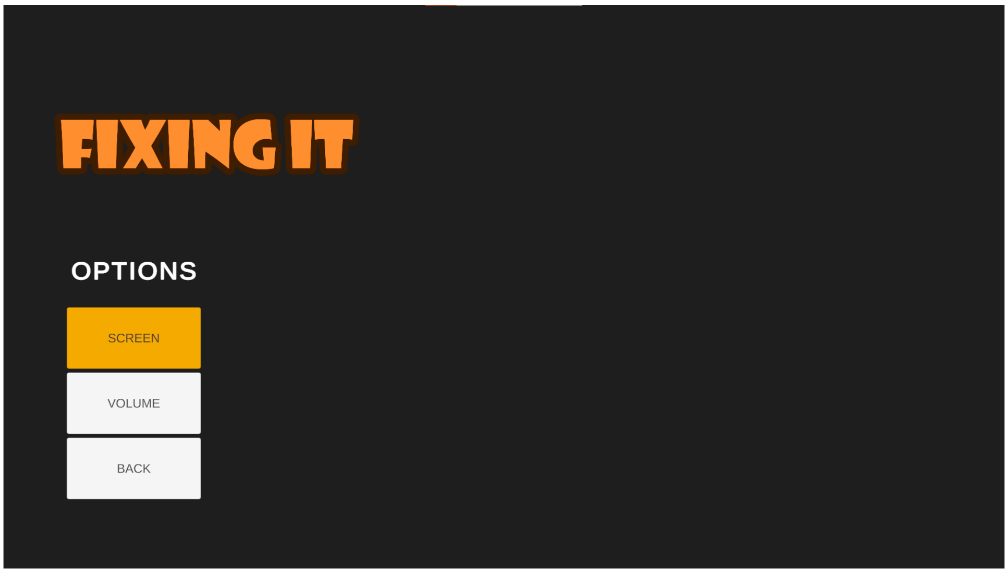


Figure 3.12: Options Menu

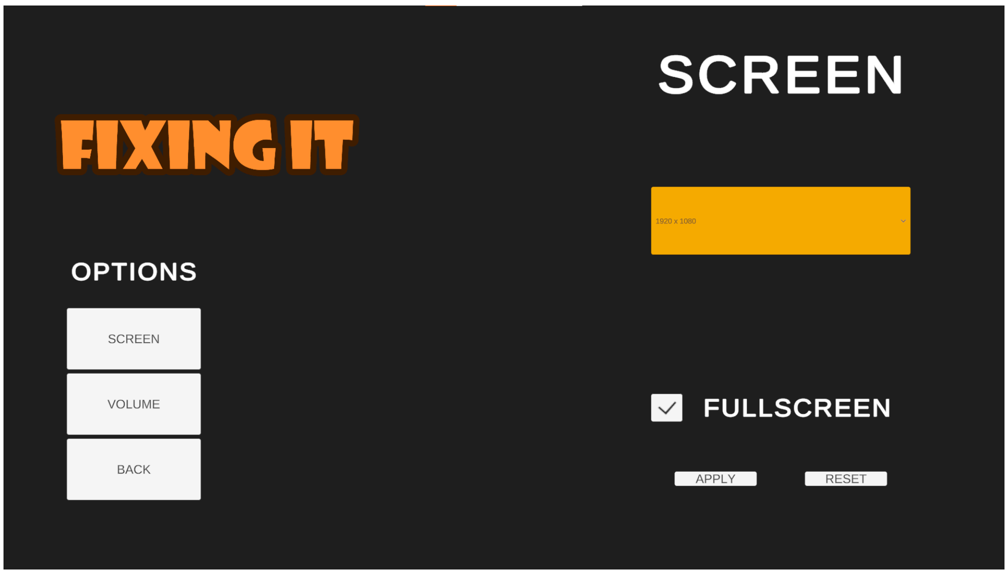


Figure 3.13: Options Menu in Screen Settings

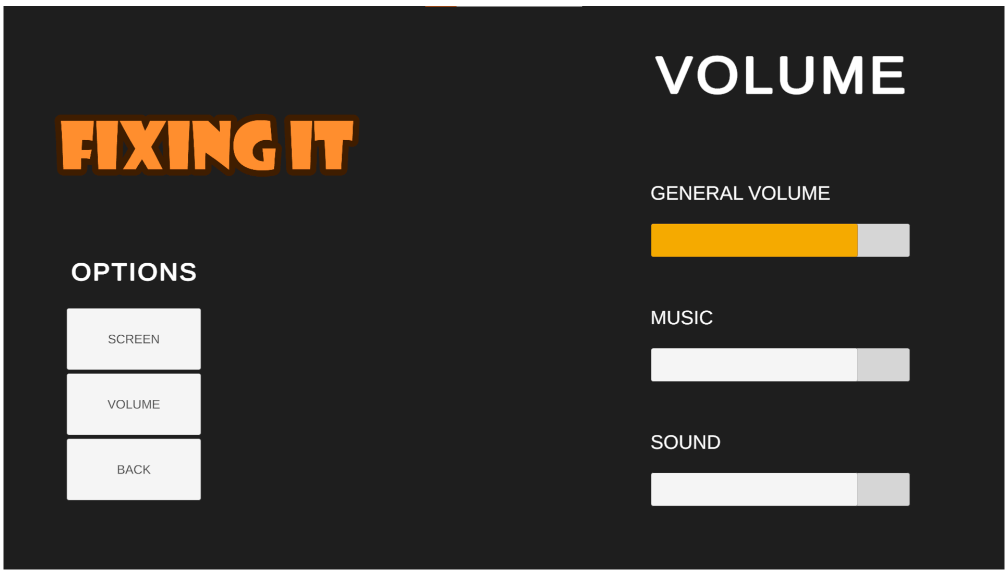


Figure 3.14: Options Menu in Volume Settings

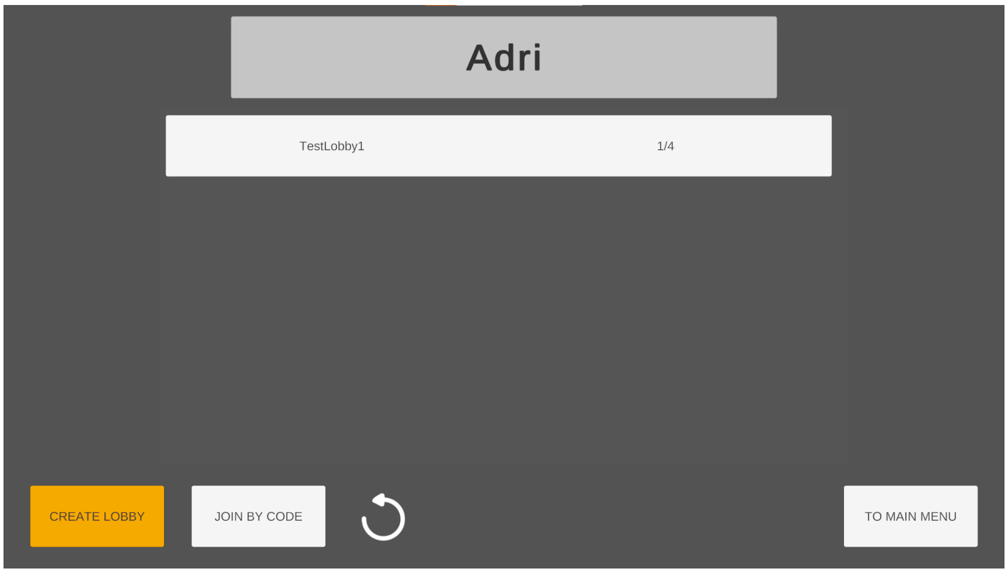


Figure 3.15: Lobby Menu with a public lobby available

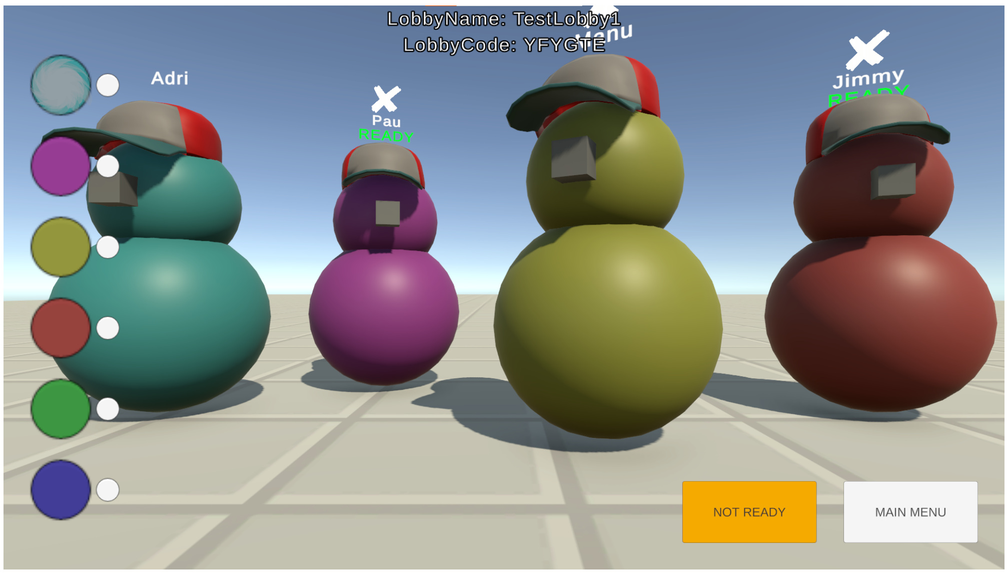


Figure 3.16: Character Selection Menu

3.7 Level design

The prototype level of the game is as shown in the following image.

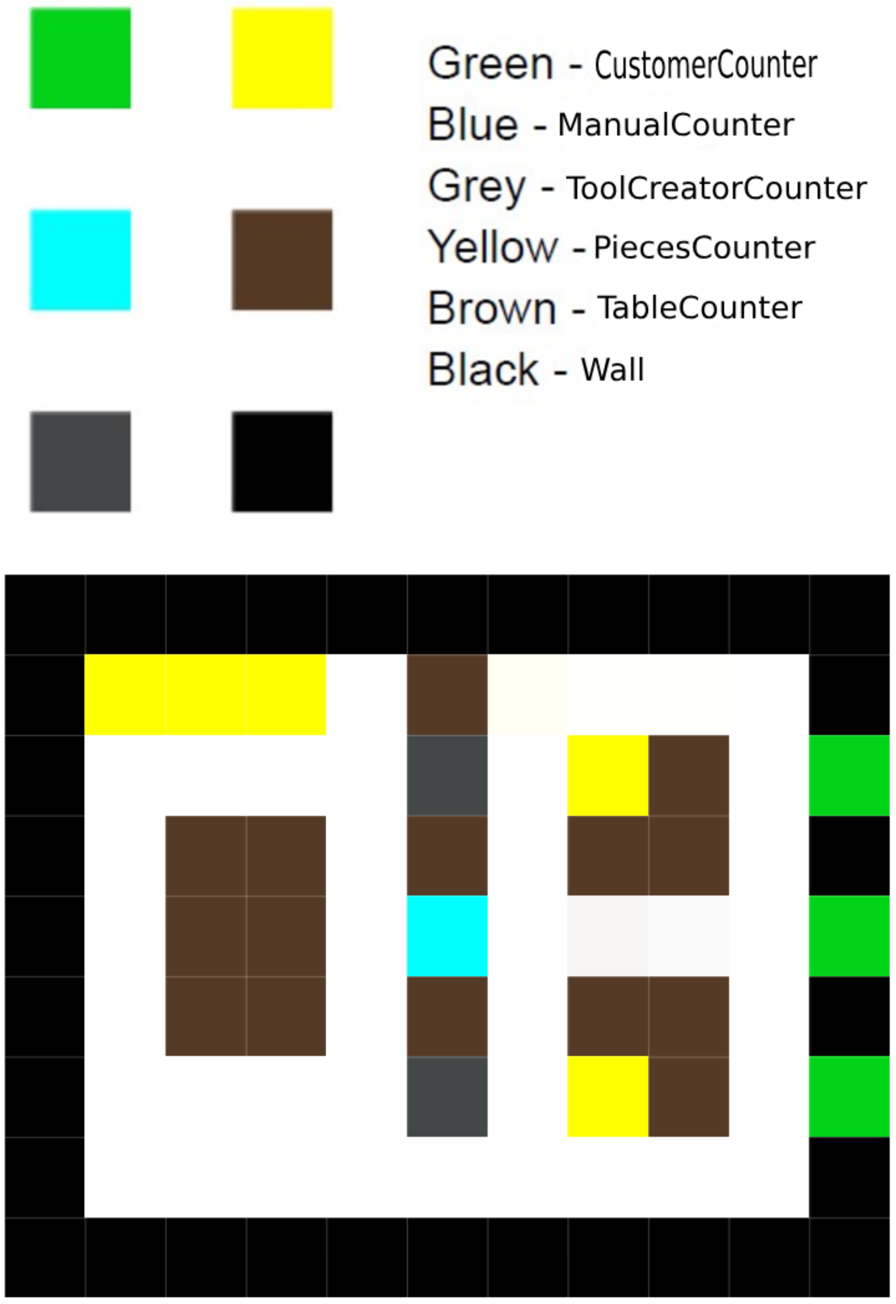


Figure 3.17: Level Design

3.8 Story

This game will have no story due to the fact that it is a prototype. But in case it will be wanted to be expanded, the story could be related to a group of mechanics who make a repair shop and as they become famous in the village they get more and more things to fix.

As they gain popularity, their workload increases with more complex repairs. The mechanics' skills and dedication become renowned, attracting customers from far and wide.

3.9 Art Style

Placeholders will be used in the first instance and if time permits, they will be replaced by Simplistic Low-poly approach, taking inspiration on games such as Overcooked, also implementing Unity Store Assets and maybe assets created by.

The final artwork will be added in future versions of the game, for the final prototype placeholders have been used.

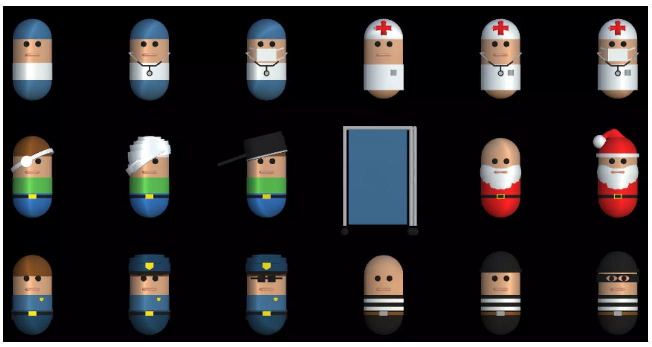


Figure 3.18: Reference Art 0

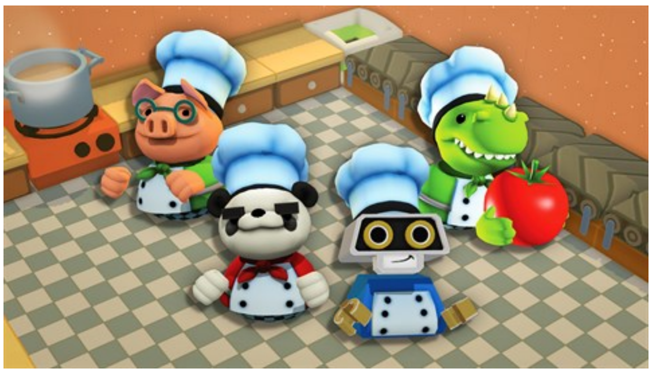


Figure 3.19: Reference Art 1

3.10 Soundtrack

3.10.1 Sound Effects

There will be small sound effects for when a tool is used or an item is fixed; when a tool is created; the customer receives their fixed item; any of the PiecesCounter in the level is used; a tool is broken; a Customer is moving; or for when the player character is moving.

3.10.2 Music

The music will be a bit fast to give the player even more of a time trial feeling when playing the mini-game.

3.11 Multiplayer

Up to 4 players per game will be able to connect online, each player having a different character colour.

Players will not be able to interact with each other with any direct action. What they will be able to do is to annoy each other with their collisions, thus generating situations where the gameplay changes as the characters become a moving wall for the other players.

Chapter 4 Work Development and Results

Contents

This chapter is going to explain how everything was developed and if anything has changed from the initial planning will be detailed and justified.

4.1 Work Development

This section explains the most relevant aspects of the developed work following the same order as shown in Figure 2.1 in the Planning.

4.1.1 Overview

The overall project follows a very similar structure. The *Observer Pattern* has been followed in order to create a clean and independent code architecture. For this, we have decided to use the *Scriptable Objects* [1] offered by Unity as a bridge between the observed object and the different observers, thus generating a layer of abstraction that would allow changing any side of the bridge without the other side being aware of anything, this system of using Scriptable Objects as bridges has been inspired by Unity's Open Project [10].

In addition, Unity's *Assembly definitions* [2] have been used to ensure that within the code dependencies, there are no dependencies in loops and at the same time to increase

[1]Go to A.2 to see the definition of a Scriptable Object
[2]Go to A.3 to see the definition of the Assembly Definition

the compilation times of the code, The basic understanding of how to use them was provided by channels such as Infallible Code [2].

The game has been designed in a way that there will always be at least 2 active scenes:

- **LocalPersistentManagers:** A scene where all the local managers, such as the SceneLoaderManager, the AudioManager or the InputController, will be located. It will also have a canvas which will be displayed while downloading and loading other scenes.

- **GameScene:** It will be the scene which will be replaced according to the scene we want the player to see. Depending on the moment of the game it can be the Main Menu scene, the Lobby Selection scene, the CharacterSelection scene or the Minigame scene.

4.1.2 Scene Loader

The first thing that was not originally planned was to make a SceneLoaderManager [3] and to apply for the first time the *Observer Pattern* with the *Scriptable Objects* as bridges, which was necessary when programming everything and seeing that at least 2 active scenes were needed simultaneously.

The very first thing we needed to do was to create our bridge system using *Scriptable Objects*. To do this we created a script with an abstract class which would contain a *UnityAction<T>* and a public function that would receive a parameter of type T and would invoke the *UnityAction* in case it has subscribed functions. [4]

We also create another new script for the UnityActions that do not receive any parameter instead of creating a child class because T cannot be void. [5]

Once this was done we designed how the GameScene loading and unloading system works.

There is a GameObject called ChangeSceneBridge inside LocalPersistentManagers that is in charge of listening to those VoidEventChannelSO that invoke its UnityAction and have been created to notify that you want to change scene.

When its private functions are called by those events, this script is in charge of invoking the UnityAction of the corresponding LoadSceneChannelSO.

LoadSceneChannelSO is a script that inherits from BaseEventChannelSO<T> which has set GameSceneSO as T, a subtype of ScriptableObject created for the project that has as its only parameter an AssetReference and will be used as a scene reference.

These last UnityAction are listened by the SceneLoaderManager, another GameObject inside LocalPersistentManagers, and because it receives as parameter a GameSceneSO the SceneLoaderManager will load its AssetReference as new GameScene of the game and in turn it will unload the previous GameScene and it will decide according to the LoadSceneChannelSO that has sent the event whether to load it as local scene or as multiplayer scene.

[3] Go to B.4 to see how the SceneLoaderManager is implemented
[4] Go to B.1 to see how the asbtract class is implemented
[5] Go to B.2 to see how the VoidEventChannelSO is implemented

In case the SceneLoaderManager decides to load multiplayer, the loading logic will be delegated through a StringBoolFuncSO, another bridge system created from Scriptable Objects in which instead of invoking UnityAction a Func is invoked where one side of the bridge assigns the desired function to the Func and the other side invokes that function [6] , so that a script called NetworkSceneLoader [7] is in charge of carrying all the logic of the loading and unloading of the multiplayer scenes.

When starting to load and when finishing loading the scenes it will invoke 2 different VoidEventChannelSOs, one for each moment, which will be listened by the Canvas located in LocalPersistentManagers to show a loading screen while everything is happening.

Once everything has been prepared, it is decided that the game will start with a scene called Initialisation, in charge of loading LocalPersistentManagers asynchronously and when it detects that they are already loaded directly invoke the UnityAction of the LoadSceneChannelSO, which is to load locally with parameter the GameSceneSO with the reference of the MainMenu, and instantly unload Initialisation asynchronously.

In this case it has been decided to call directly the LoadSceneChannelSO instead of a VoidEventChannelSO that is listened by the ChangeSceneBridge because this last one was created as a bridge between the different GameObjects that want to change scene and the SceneLoaderManager because when splitting the code with the assemblies definitions of Unity we realised that to use the LoadSceneChannelSO it was necessary to have the reference of the GameSceneSO, which GameObjects in general don't seem to need to have.

On the other hand, the Initialisation scene code does it directly because due to its operation it was decided to group it with the rest of the codes in charge of handling scenes, so they are in the same assembly definition and can use the LoadSceneChannelSO without problems.

The SceneLoaderManager is also in charge of closing the game.

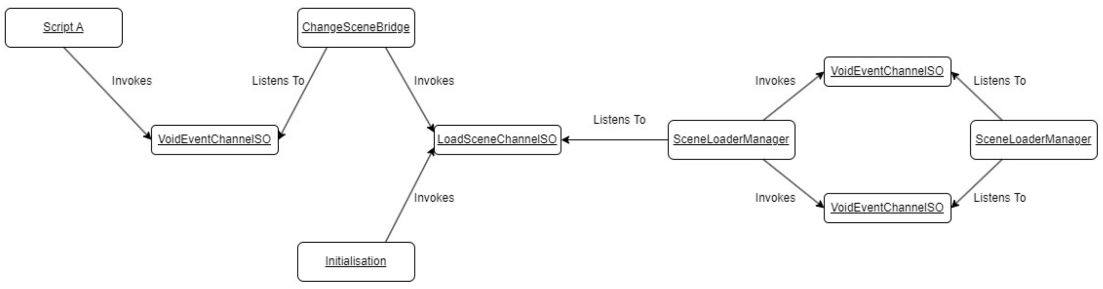


Figure 4.1: Relationship diagram of the scene loading

[6]Go to B.3 to see how the base abstract class for FuncSO is implemented
[7]Go to B.5 to see how the NetworkSceneLoader is implemented

4.1.3 Main Menu

It is a simple menu where the player can decide to exit the game; enter in the options menu and change the screen or volume settings; or play the game.

The first thing to do was to make a simple user interface design to reflect all actions. For this we created a canvas with 4 different panels:

- **Main Menu Panel:** It has Play, Options and Exit buttons. See Figure 3.11.
- **Options Panel:** It has Screen, Volume and Back buttons. See Figure 3.12.
- **Screen Settings Panel:** It has a dropdown with all the Screen Resolutions; a Fullscreen toggle; and Accept and Reset Buttons. See Figure 3.13.
- **Audio Settings Panel:** It has General Volume, Music and Sound Sliders. See Figure 3.14.

Once all the UI design was done and animated using a LeanTween library [1], we proceeded to make all the logic of it following the Observer Pattern and separating the logic of the UI from the internal logic.

Visual Logic

The first thing I did was to create a custom script for each panel which would have referenced all the objects on the panel: buttons, sliders... Then I added to these referenced objects that after being used by the user would invoke one of our custom Scriptable Objects.

- **Main Menu Panel:** Its buttons would each invoke a VoidEventChannelSO.
- **Options Menu Panel:** Its buttons would also invoke a VoidEventChannelSO.
- **Screen Settings Panel:** Only the Accept and Cancel buttons are associated to a custom Scriptable Object but these needed to pass 2 arguments, the index of the chosen resolution and a boolean to indicate the full screen, so it was not possible to make a subclass of BaseEventChannelSO<T>, so it was decided to make a homonymous class but with 2 parameters called BaseEventChannelSO<T1,T2> [8] and then make a subclass of it which would have as T1 an int and as T2 a bool, called ScreenSettingsChannelSO.
- **Audio Settings Panel:** Its 3 sliders invoke FloatEventChannelSO, subclass of BaseEventChannelSO<T> that have as T a float, and pass the values of the sliders as argument of the event.

Once all the panel scripts have been created a Canvas script was created to handle all the internal animations and which Panels will be displayed or disabled.

[8] Go to B.1 to see how BaseEventChannelSO<T1,T2> is implemented

Internal Logic

- **Main Menu Panel:**
  - Play Button: The VoidEventChannelSO invoked by this button is listened by the ChangeSceneBridge, which is already internally in charge of invoking the LoadSceneChannelSO listened by the SceneLoaderManager to load the LobbySelection scene as the GameScene locally.
  - Options Button: The VoidEventChannelSO invoked by this button is listened by the Canvas and it is in charge of making the animation to show the Options Menu Panel and to hide and deactivate the Main Menu Panel.
  - Exit Button: The VoidEventChannelSO invoked by this button is listened by the SceneLoaderManager and it closes the game.
- **Options Menu Panel:** The VoidEventChannelSOs invoked by each button are listened by the Canvas to display the Screen Settings Panel, Volume Settings Panel and to re-display the Main Menu Panel respectively.
- **Screen Settings Panel:** Both Accept and Cancel buttons invoke the same ScreenSettingsChannelSO but with different parameters, the first button gets the index of the current item in the dropdown and the gets if the toggle is On or not while the second button send the maximum resolution index and a true. The event is listened by another gameobject called ResolutionManager, which has a script that manages the resolution and whether it is in fullscreen, and resizes the game view.
- **Audio Settings Panel:**The FloatEventChannelSOs invoked by each Slider are listened by the AudioManager, a gameobject inside LocalPersistentManagers designed to control the sound channels and the music that is currently playing.

A demostration video of the Main Menu: MainMenu Demo Video

4.1.4 Lobby and Relay System

To do this, a scene called LobbySelection has been created.

Here the player will be in a menu where they can change their player name, choose whether to create a lobby, join one of the public lobbies, join a private lobby via code or return to the Main Menu.

As with the Main Menu, the first thing that was done was to design a UI, which can be organised in 4 different parts, see Figure 3.15:

- **PlayerNameInputField:** It is just an InputField where the player can modify his/her online name.

- **LobbiesScrollArea Panel:** It has a button within a vertical group that acts as a template join button to be duplicated when new public lobbies are detected, it also has a text to indicate that there are no public lobbies at the moment.
- **Lobby Options Panel:** It has the Create Lobby, Join by Code, Refresh Lobbies and To Main Menu buttons.
- **PopUp Panel:** Panel that internally has 4 panels: the CreateLobby Panel, JoinByCode Panel, LobbyErrorPanel and LobbyState Panel.

And as in the Main Menu, it was decided to separate the internal logic from the UI logic by using the Observer Pattern.

Visual Logic

The same structure as in the Main Menu has been followed again: create a script for each panel with references to its objects and also that these objects, when used, invoke the corresponding customs Scriptable Objects.

A script has also been made for the current canvas to act as a manager for all panels, it listens all the necesarry events from our customs Scriptable Objects to update or set the values or the visibility of the panels.

- **PlayerNameInputField:** Being an object with nothing internally in charge of it, its logic has been delegated to the Canvas. Each time its value is changed it invokes a StringEventChannelSO, another subclass of BaseEventChannelSO<T> with a string as T, passing as parameter its new value.
- **LobbiesScrollArea Panel:** Listens for a LobbiesChannelSO, another subclass of BaseEventChannelSO<T> that uses List<Lobby> as T, and when that LobbiesChannelSO is invoked causes this panel to receive all available public lists; turns on or off the message that no public lobbies are available depending on the size of the list received; and all dynamically generated buttons to join lobbies are destroyed and new ones are generated based on the lobbies in the list received. In addition, dynamically generated lobby join buttons display the lobby name, the number of people in the lobby and the maximum number of people at the time the button is created. It also internally stores the lobby reference so that when it is pressed it invokes a StringEventChannelSO passing the lobby ID as a parameter.
- **Lobby Options Panel:** Each button invokes a VoidEventChannelSO.
- **PopUp Panel:** Panel that acts as a manager for each of the possible PopUps that may appear in this scene.
  - CreateLobby Panel: Its Create button invokes a CreateLobbyChannelSO, a subclass of BaseEventChannelSO<T1,T2> and uses a string as T1 and a bool as T2, in which it passes as parameters the name chosen for the lobby and

the boolean to indicate whether it is public or not. Its cancel button invokes a VoidEventChannelSO. See Figure 4.2.

– JoinByCode Panel: Its Join button invokes a StringEventChannelSO in which it passes as parameter the lobby code entered by the user in the Panel Input Field. Its cancel button invokes a VoidEventChannelSO. See Figure 4.3.

– LobbyError Panel: It has only one button which invokes a VoidEventChannelSO. See Figure 4.4.

– LobbyState Panel: This panel has no event to invoke or listen to, it is handled directly by the PopUp Panel. See Figure 4.5.

Internal Logic

- **PlayerNameInputField:** The StringEventChannelSO invoked after its modification is listened by the FixingGameMultiplayer [9] which takes care of saving the name in a variable and in Unity's PlayerPrefs so you don't have to set it between game.

- **LobbiesScrollArea Panel:** The StringEventChannelSO invoked by the buttons to join a lobby are listened by the LobbyManager [10] which tries to join the actual user to the given id lobby. The LobbyManager invokes a StringEventChannelSO depending on where it is in the joining process and as a parameter a small text indicating the current state of the process. In case of an error in any part, a different StringEventChannelSO is invoked with parameter the reason for the error. Any of these 2 invocations are listened by the Canvas and it shows the PopUp of turn to indicate to the user the process. If the user successfully joins the lobby and the relay, a VoidEventChannelSO is invoked which is listened to by the FixingGameMultiplayer and is responsible for making connections as a client to that server and to invoke a VoidEventChannelSO which will be listened by the ChangeSceneBridge and this will invoke a LoadSceneChannelSO listened by the SceneLoaderManager to load the CharacterSelection scene in a multiplayer way.

- **Lobby Options Panel:**

  – Create Lobby and Join by Code Buttons: The VoidEventChannelSOs invoked by both buttons are listened to by the canvas itself, which is responsible for displaying the corresponding PopUp.

  – Refresh Lobbies Button: The VoidEventChannelSO invoked is listened by the LobbyManager, which invokes a StringEventChannelSO and as parameter a text indicating that it is looking for lobbies. After that it searches for

[9]Gameobject with a script in charge of handling purely multiplayer data such as if a player is ready to play, the name, the colour... You can see the implementation in B.6.

[10]Gameobject with a script in charge of handling the lobby creation, connection, destruction, kicking out players, managing erros... The script also handles the whole Relay System. You can see the implementation in B.7.

all available public lobbies and at the end it invokes a LobbiesChannelSO with the parameter the list of the lobbies found, which is listened by the LobbiesScrollArea and this is in charge of updating the UI. If there is any problem when searching for public lobbies, a different StringEventChannelSO is invoked with the parameter the reason of the problem and it would be listened by the Canvas to show the corresponding PopUp.

  - To Main Menu Button: The VoidEventChannelSO invoked is listened by the ChangeSceneBridge which is already internally in charge of invoking the LoadSceneChannelSO listened by the SceneLoaderManager to load the MainMenu scene as GameScene locally.

- **PopUp Panel:**
  - CreateLobby Panel: The CreateLobbyChannelSO invoked is listened by the LobbyManager and start creating the lobby and invokes a StringEventChannelSO depending on where it is in the creating process and as a parameter a small text indicating the current state of the process. In case of an error in any part, a different StringEventChannelSO is invoked with parameter the reason for the error. Any of these 2 invocations are listened by the Canvas and it shows the PopUp of turn to indicate to the user the process. If the user successfully creates the lobby and the relay, a VoidEventChannelSO is invoked which is listened to by the FixingGameMultiplayer and is responsible for making connections as a host, i.e. client and server. The VoidEventChannelSO is listened by the Canvas to hide the PopUp.
  - JoinByCode Panel: The StringEventChannelSO is listened by the LobbyManager which tries to join the actual user to the given code lobby. The custom Scriptable Objects that are invoked in the process are the same as when the user tries to join by id, i.e. trying to join a public lobby.
  - LobbyError Panel: The VoidEventChannelSO invoked is listened by the Canvas which hides the PopUp.
  - LobbyState Panel: It has no custom Scriptable Object invoked, so there is no internal logic to explain.

A demostration video of the Lobby and Relay System: Lobby Screen Video

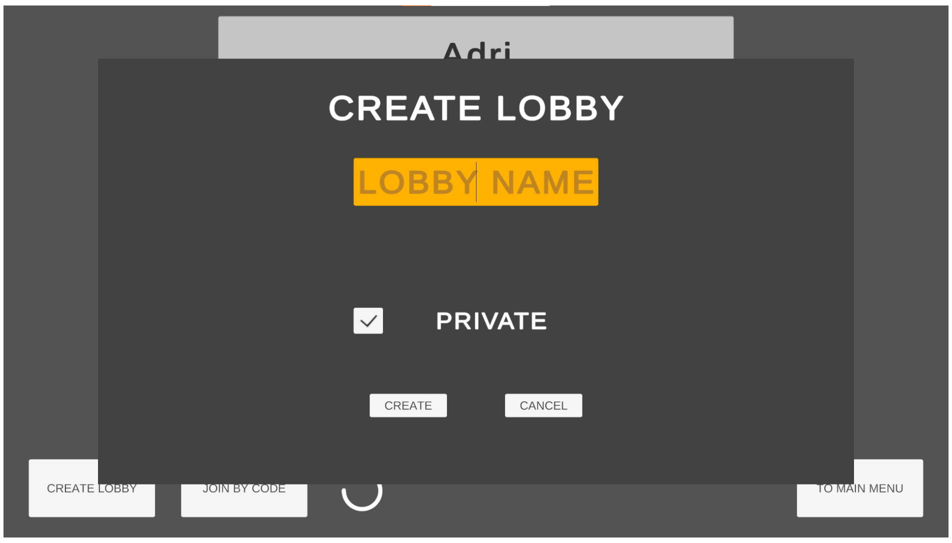


Figure 4.2: Create Lobby Panel

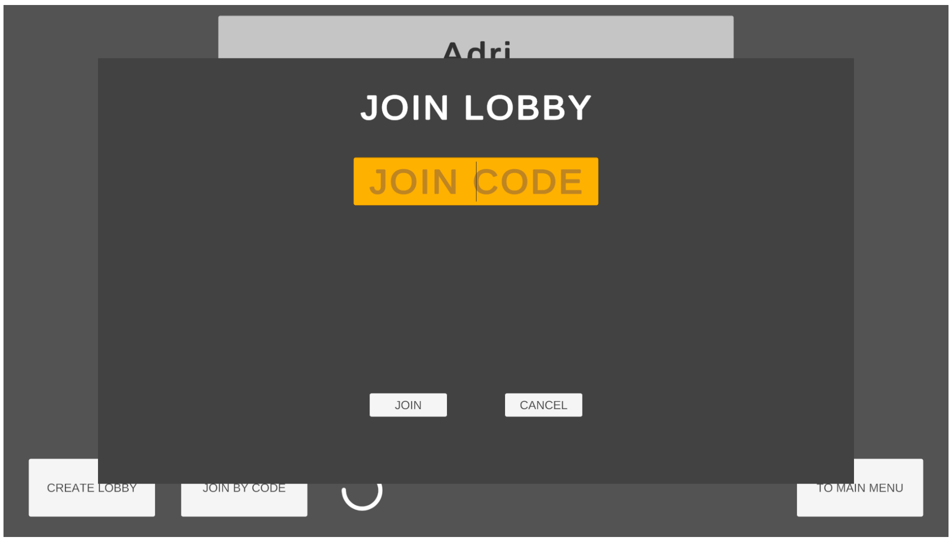
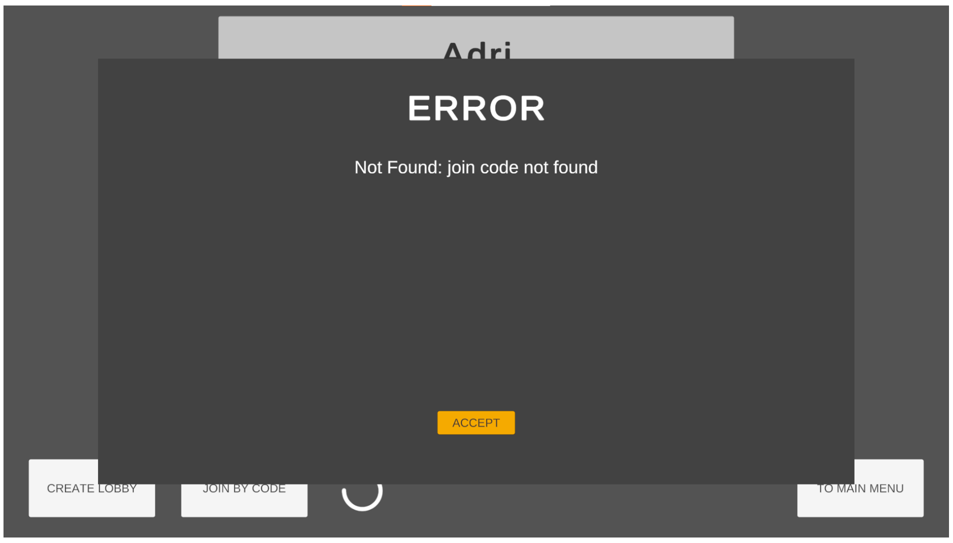
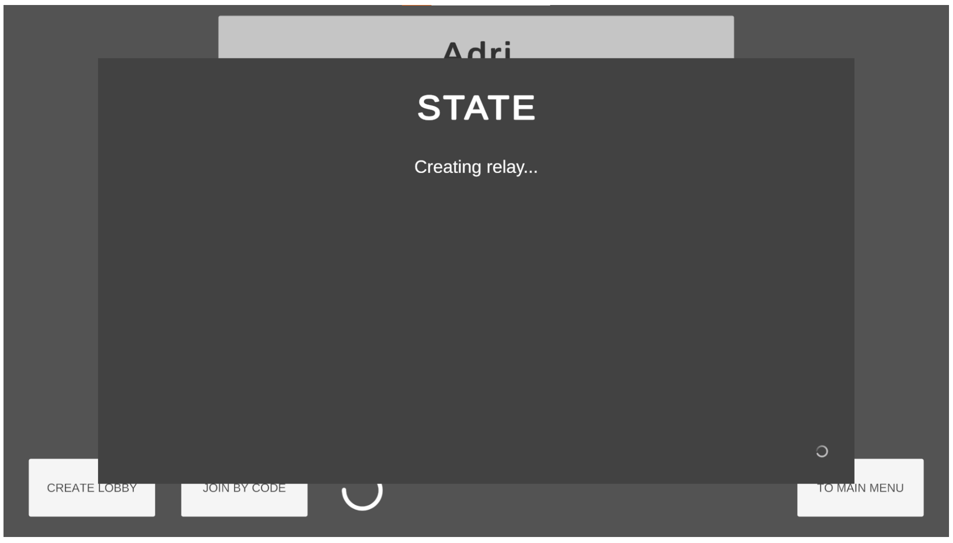


Figure 4.3: Join by Code Panel



Not Found: join code not found

Figure 4.4: Lobby Error Panel



Creating relay...

Figure 4.5: Lobby State Panel

4.1.5 Character Selection

To do this, a scene called CharacterSelection has been created.

This is a scene that will only be loaded in multiplayer mode and the user will be the host or a client.

It acts as a waiting room until all players in the lobby are ready to play. There can be up to 4 players at the same time.

While waiting for all players to be ready to play, the name of the lobby; the lobby joining code; the names and colours of the players; and whether they are ready or not will be displayed. There will also be a list of 6 colours for the players to decide which colour they want to have on their character, with the only rule that they can't have a colour that another character already has.

The first thing that was done was to design a UI and the visual representation of players, see Figure 3.16. It can be organised in:

- **The players representation:** This gameobject contains the PlayerVisual prefab; the text for the player name; the text for the ready text; and a small canvas with a button to let the host kick the player.
- **Buttons Panel:** It has Ready and Main Menu buttons.
- **Colours Panel:** It has 6 different buttons, each for a different colour.
- **Leave to Menu Panel:** It is a small popup asking the user if they really want to leave the lobby. It has Leave and Cancel buttons.
- **Kicked Panel:** It is another popup that notifies the player that he/she has been kicked. It only has Accept button.
- **Name and Code Panel:** It contains 2 texts that are filled in with the lobby name and the lobby joining code respectively.

All the panels will be inside a canvas that will act in the same way as the previous canvasses, as a manager of them.

Visual Logic

- **The players representation:** Every gameobject listens 2 VoidEventChannelSO: the first to update the visibility and colour of the gameobject; and the second to display or not the Ready text. It also invokes an ULongEventChannelSO, another subclass of BaseEventChannelSO<T> that uses an ulong as T, when the kick button is pressed and passes as parameter the id of the client that is kicked out. [11]

[11] Go to B.8 to see the implementation of the Player Representation

- **Buttons Panel:** Both buttons are handled directly by the canvas. The Ready button invokes a VoidEventChannelSO. The MainMenu button when pressed displays the Leave to Menu Panel.
- **Colours Panel:** This panel does not have a script, each button manages itself. When each button is pressed it sends an IntEventChannelSO, a subclass of BaseChannelSO<T> which uses an int as T, passing as a parameter the id of the colour.
- **Leave to Menu Panel:** Each button invokes a different VoidEventChannelSO when pressed.
- **Kicked Panel:** When its Accept button is pressed, a VoidEventChannelSO is invoked.
- **Name and Code Panel:** They are handled by the canvas, which at scene startup invokes 2 StringFuncSO, subclasses of BaseFuncSO<TResult>, and sets the values returned by those Funcs as the respective texts.

Internal Logic

- **The players representation:** The ULongEventChannelSO invoked is listened by the FixingGameMultiplayer and manages the disconnection of the player with that clientId.
- **Buttons Panel:** The VoidEventChannelSO invoked is listened by the CharacterSelectionManager [12] , another script designed to take into account which players are ready and which are not in order to send a VoidEventChannelSO event which will be listened to load the Minigame scene.
- **Colours Panel:** The IntEventChannelSO invoked is listened by the FixingGameMultiplayer and changes the color of the player who used the button.
- **Leave to Menu Panel:** The VoidEventChannelSO invoked by the Leave buttons is listened by the FixingGameMultiplayer which manages the disconnection of the player and after that it invokes another VoidEventChannelSO listened by ChangeSceneBridge and it invokes a LoadSceneChannelSO listened by the SceneLoaderManager to load the MainMenu.
- **Kicked Panel:** The VoidEventChannelSO invoked is listened by the ChangeSceneBridge and it invokes a LoadSceneChannelSO listened by the SceneLoaderManager to load the MainMenu.
- **Name and Code Panel:** They have no internal logic apart from the visual.

A demostration video of the Character Selection: Character Selection Screen

[12] Go to B.9 to see the implementation of the CharacterSelectionManager

4.1.6 Mini-game

A scene was created that will contain the mini-game.

First of all, all the mechanics and their relationships at the single player level were thought out.

The mini-game has used the Observer Pattern along with creating Components. This has been done because there are certain things that don't make sense to put that layer of abstraction because of its direct relationship, like the character and the sounds or particles it generates, it knows when to make sound or when to activate its particle system.

FixingGameManager

It is the manager of the level logic [13].

It is in charge of managing the activation and deactivation of the player input or changing the Input channel from Gameplay to Menu mode and vice versa; has a Scriptable Object with the recipes that the level will use and then the objects that need it will get it through a ToolRecipeManagerFuncSO, subclass of BaseFuncSO<TResult> that uses a ToolRecipeManagerSO as T, which in turn is a Scriptable Object where an array with the tool crafting recipes is stored; where to spawn the NPCs, which Object to Fix with and which counter to go to; and the game state.

Room Objects

They are part of the main mechanics of the game [14] .

They are divided according to a number into 3 different types: Pieces, Tools and Objects To Fix.

They have 2 AudioComponent, Component programmed in order to run a sound clip on a specific channel, one for when the object is used and one for when the object breaks. Although in reality they are only used by the Tools.

They have a reference to an IRoomObjectParent [15] , an interface designed for all those gameobjects that will be able to handle RoomObjects, which will be their parent in the hierarchy of objects in the scene and this IRoomObjectParent in turn has a reference to the RoomObject.

it has a static function that is in charge of spawning new RoomObjects in an IRoomObjectParent.

For the Object to Fix a subclass [16] has been created because it needs to recycle all the logic of the RoomObject but adding an array indicating the tools needed to fix the object and another array to take into account the tools already used.

[13]Go to B.10 to see the implementation

[14]Go to B.11 to see the implementation of RoomObject

[15]Go to B.12 to see the implementation of IRoomObjectParent

[16]Go to B.13 to see the implementation of the Object To Fix subclass, it is called ToFixRoomObject

To show the user the tools he/she needs to use, a component has been created for the Object to Fix that is in charge of showing the image representation of the tools to be used on top of the object. See Figure 3.4.

Player

Gameobject in charge of moving; rotating; acting as IRoomObjectParent; and handling the interactions that the player can perform with the different types of counter, calling the BaseCounter and this is in charge of inheriting the behaviour of the interaction [17].

It has 2 components in charge of the animation of the player and the walking sound.

And the gameobject has a particle system that is activated by the character's movement.

Counters

They are a group of different objects which are also necessary for the game. They all have a common parent class, BaseCounter [18], which each Counter overrides to implement its functionality.

There are a total of 5 different Counters and all of them act as IRoomObjectParent. It is possible to see all of them in the subsection 3.4.2.

NPCs

The code that controls them is CustomerController [19] and its internal logic is a hardcoded state machine because it is the only place in the project where one would be required. It also acts as IRoomObjectParent.

As can be seen in the figure 4.6 it can be seen that the first thing the NPC does is to go to a Counter, once there he waits until the broken object he had taken to fix is returned to him and when he has finished he leaves to leave the Counter for the next NPC.

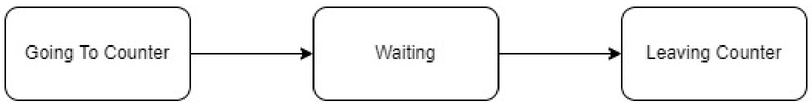


Figure 4.6: NPCs State Machine

4.1.7 Adaption of the mini-game to multiplayer

To modify the mini-game logic from singleplayer to multiplayer mode what has been done has been:

[17]Go to B.14 to see how the Player Controller is implemented
[18]Go to B.15 to see how the BaseCounter is implemented
[19]Go to B.16 to see how the CustomerController is implemented

- **NetworkVariables and NetworkLists:** Variables and lists which are synchronised between all users. They are used for important things like the number of uses of a tool or the remaining tools that an ObjectToFix needs.
- **ServerRpc and ClientRpc functions:** These are functions that run on the server and on the clients respectively. When a client tries to perform an action, it usually calls a ServerRpc function so that the server executes all the internal logic and when it is finished, it calls a ClientRpc function so that all the clients receive the corresponding visual changes.
- **Network events:** Events that NetworkVariables and NetworkLists have which are executed every time their values have been changed. There are certain parts of the code where this has been chosen instead of the ServerRpc and ClientRpc in order to avoid synchronisation failures of these NetworkVariables when calling the ClientRpc.

4.1.8 Extra

There are 3 things that have been done as extras to all of this:

- **Centralise the Input System with a Scriptable Object:** To do this Unity's new Input System have been used and it was created a code called InputReaderSO [20] which is a Scriptable Object that anyone can access and has the designated functions and events to control all the game's Input.
- **Centralising the colours of UI elements:** A Scriptable Object was created which contains the set of colours that any UI element has. After this, a code called SelectableUIData [21] was created which would be in charge of assigning the colours of the Scriptable Object to the elements that contain this code, thus allowing a centralisation of the colours and an ease of design.
- **Put the more generic codes in a separate folder to start creating a library:** As codes have been created that are tools for programming or design, I have decided to separate them in a separate folder called ProgramadorCastellano. This is because they are codes that can be improved and used in the future as a custom library. Inside would be codes such as BaseEventChannelSO or SelectableUIData.

4.2 Results

Although Unity's new multiplayer library is still buggy, a first version of a videogame has been created following the Observer Pattern as much as possible and it is playable by up to 4 simultaneous players in a single lobby, who can enjoy short 2-minute games.

[20] Go to B.17 to see how the InputReaderSO is implemented

[21] Go to B.18 to see how the SelectableUIData is implemented

From a computer science point of view, the separate codes in the ProgramadorCastellano folder can serve as a library for any programmer who wants to create an Observer Pattern. Also due to the separation of interface and logic codes anyone who wants to recreate the lobbies system can use the codes already created for this project without having to modify it for the most part.

Chapter 5

Conclusions and Future Work

Contents

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

5.1 Conclusions

It was a challenge to make a multiplayer game with a design pattern applied to it. This is because we have never dealt with anything related to multiplayer games in our career, nor have we ever put into practice the use of design patterns when organising code.

Also the use of a new library has resulted in more difficulties in the development of the project. Synchronisation errors of certain objects that could only be fixed by converting them to Singletons, even though this was not originally intended, that certain objects shared their NetworkObject ID so that the value had to be re-assigned by hand, or that when loading scenes asynchronously, clients could not disconnect correctly due to an internal bug in the library, causing the user to have to disconnect by brute force with all the problems that this entails.

So to develop a game of any kind with a new library I recommend waiting a bit until it is more developed and does not have major bugs like those mentioned above.

5.2 Future work

As future work the idea would be to wait a little while for the Unity multiplayer library bugs to be fixed.

Once there are no major bugs we would continue with the progress of the game to redo the asynchronous download of the clients without crashing, add visual and sound details to the game, take advantage of the Observer Pattern to make a system of achievements without too much difficulty and study about level design to make different levels with different tools and objects to fix so that there is more variety and therefore fun for the end player.

Regarding the more abstract codes stored in the ProgramadorCastellano folder the idea is to refine and extend those codes to generate a public library to have all the primitive BaseEventSO and BaseFuncSO ready to use just by downloading the folder and also to have a central modification system of the UI that allows apart from changing the colours, to change the images or the animations of the UI in a centralised way.

Bibliography

[1] LeanTween - LeanTween.

[2] Infallible Code. Before you continue to YouTube.

[3] Code Monkey. Before you continue to YouTube.

[4] Tarodev. Before you continue to YouTube.

[5] Unity Technologies. Unity - Manual: Assembly definitions.

[6] Unity Technologies. Unity - Manual: ScriptableObject.

[7] Unity Technologies. Unity Lobby Service.

[8] Unity Technologies. Unity Relay.

[9] Unity Technologies. About Netcode for GameObjects | Unity Multiplayer Networking, 6 2023.

[10] Unity. Bringing characters to life with animation | Open Projects Devlog, 5 2021.

[11] Wikipedia contributors. Observer pattern — Wikipedia, the free encyclopedia, 2023.

Appendix A

Glossary

A.1 Observer Pattern

“In software design and engineering, the observer pattern is a software design pattern in which an object, named the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

It is often used for implementing distributed event-handling systems in event-driven software. In such systems, the subject is usually named a "stream of events" or "stream source of events" while the observers are called "sinks of events." The stream nomenclature alludes to a physical setup in which the observers are physically separated and have no control over the emitted events from the subject/stream source. This pattern thus suits any process by which data arrives from some input that is not available to the CPU at startup, but instead arrives seemingly at random (HTTP requests, GPIO data, user input from peripherals, distributed databases and blockchains, etc.).

Most modern programming languages comprise built-in event constructs implementing the observer-pattern components. While not mandatory, most observer implementations use background threads listening for subject events and other support mechanisms provided by the kernel.” [11]

A.2 Scriptable Object

“A ScriptableObject is a data container that you can use to save large amounts of data, independent of class instances.” [6]

A.3 Assembly Definition

“An assembly is a C# code library that contains the compiled classes and structs that are defined by your scripts and which also define references to other assemblies.” [5]

Appendix B

Source code

Here you will find the most relevant codes necessary to better understand the explanation of the document.

To see all the code of the game: https://github.com/AdriMon27/FIXING-IT

B.1 BaseEventChannelSO

```
using ProgramadorCastellano.Base;
using UnityEngine;
using UnityEngine.Events;

namespace ProgramadorCastellano.Events
{
    /// <summary>
    /// Base Scriptable Object class for creating our personals EventChannelSO.
    /// Don t forget to add CreateAssetMenu.
    /// If you want to change the names in the parameters you can make a new RaiseEvent()
    /// and call base.RaiseEvent()
    /// </summary>
    /// <typeparam name="T">Struct,List,Dict... that the event channel will use</typeparam>
    public abstract class BaseEventChannelSO<T> : DescriptionBaseSO, IMyEventSO<T>
    {
        public UnityAction<T> OnEventRaised { get; set; }

        public void RaiseEvent(T eventArg)
        {
            if (OnEventRaised != null) {
                OnEventRaised.Invoke(eventArg);
            }
            else {
                Debug.LogWarning($"{errorMessage}␣with␣parameter␣{eventArg}");
            }
        }
    }

    public abstract class BaseEventChannelSO<T0,T1> : DescriptionBaseSO, IMyEventSO<T0, T1>
    {
        public UnityAction<T0,T1> OnEventRaised { get; set; }

        public void RaiseEvent(T0 eventArg0, T1 eventArg1)
        {
            if (OnEventRaised != null) {
                OnEventRaised.Invoke(eventArg0, eventArg1);
            }
            else {
                Debug.LogWarning($"{errorMessage}␣with␣parameters␣{eventArg0}␣and␣{eventArg1}");
            }
        }
    }
}
```

B.2 VoidEventChannelSO

```
using ProgramadorCastellano.Base;
using UnityEngine;
using UnityEngine.Events;

namespace ProgramadorCastellano.Events
{
    [CreateAssetMenu(menuName = "Events/Primitive/Void␣Event␣Channel")]
    public class VoidEventChannelSO : DescriptionBaseSO, IMyEventSO
    {
        public UnityAction OnEventRaised { get; set; }

        public void RaiseEvent()
        {
            if (OnEventRaised != null) {
                OnEventRaised.Invoke();
            }
            else {
                Debug.LogWarning(errorMessage);
            }
        }
    }
}
```

B.3 BaseFuncSO

```csharp
using ProgramadorCastellano.Base;
using System;
using UnityEngine;

namespace ProgramadorCastellano.Funcs
{
    /// <summary>
    /// Base Scriptable Object class for creating our personals FuncSO
    /// Don t forget to add CreateAssetMenu
    /// If you want to change the names in the parameters you can make a new RaiseFunc() ant call base.RaiseEvent()
    /// </summary>
    /// <typeparam name="TResult">Struct,List,Dict... that the Func will return</typeparam>
    public abstract class BaseFuncSO<TResult> : DescriptionBaseSO, IMyFuncSO<TResult>
    {
        private Func<TResult> OnFuncRaised;

        /// <summary>
        /// Set the Func to null
        /// </summary>
        public void ClearOnFuncRaised()
        {
            OnFuncRaised = null;
        }

        /// <summary>
        /// Invokes the Func
        /// </summary>
        /// <returns>The <typeparamref name="TResult"/> output object if OnFuncRaised is not null or default if the Fu
        public TResult RaiseFunc()
        {
            if (OnFuncRaised != null) {
                return OnFuncRaised.Invoke();
            }
            else {
                Debug.LogWarning($"{errorMessage}");
                return default;
            }
        }

        /// <summary>
        /// Tries to set a new Func to the internal Func
        /// </summary>
        /// <param name="newFunc"></param>
        /// <returns>If it was possible to set the Func, if the Func was null</returns>
        public bool TrySetOnFuncRaised(Func<TResult> newFunc)
        {
            if (OnFuncRaised != null) {
                return false;
            }

            OnFuncRaised = newFunc;
            return true;
        }
    }

    public abstract class BaseFuncSO<T0, TResult> : DescriptionBaseSO, IMyFuncSO<T0, TResult>
    {
        private Func<T0, TResult> OnFuncRaised;

        /// <summary>
        /// Set the Func to null
        /// </summary>
        public void ClearOnFuncRaised()
        {
            OnFuncRaised = null;
        }

        /// <summary>
        /// Tries to set a new Func to the internal Func
        /// </summary>
        /// <param name="arg0">First argument of the func</param>
        /// <returns>The <typeparamref name="TResult"/> output object if OnFuncRaised is not null or default i
        public TResult RaiseFunc(T0 arg0)
        {
            if (OnFuncRaised != null) {
                return OnFuncRaised.Invoke(arg0);
            }
            else {
                Debug.LogWarning($"{errorMessage}␣with␣parameter␣{arg0}");
                return default;
            }
        }

        /// <summary>
        /// Tries to set a new Func to the internal Func
        /// </summary>
        /// <param name="newFunc"></param>
        /// <returns>If it was possible to set the Func, if the Func was null</returns>
        public bool TrySetOnFuncRaised(Func<T0, TResult> newFunc)
        {
            if (OnFuncRaised != null) {
                return false;
            }

            OnFuncRaised = newFunc;
            return true;
        }
    }

    public abstract class BaseFuncSO<T0, T1, TResult> : DescriptionBaseSO, IMyFuncSO<T0, T1, TResult>
    {
        private Func<T0, T1, TResult> OnFuncRaised;

        /// <summary>
        /// Set the Func to null
        /// </summary>
        public void ClearOnFuncRaised()
        {
            OnFuncRaised = null;
        }

        /// <summary>
        /// Tries to set a new Func to the internal Func
        /// </summary>
        /// <param name="arg0">First argument of the func</param>
        /// <param name="arg1">Second argument of the func</param>
        /// <returns>The <typeparamref name="TResult"/> output object if OnFuncRaised is not null or default if the Fu
        public TResult RaiseFunc(T0 arg0, T1 arg1)
        {
            if (OnFuncRaised != null) {
                return OnFuncRaised.Invoke(arg0, arg1);
            }
            else {
                Debug.LogWarning($"{errorMessage}␣with␣parameters␣{arg0}␣and␣{arg1}");
                return default;
            }
        }

        /// <summary>
        /// Tries to set a new Func to the internal Func
        /// </summary>
        /// <param name="newFunc"></param>
        /// <returns>If it was possible to set the Func, if the Func was null</returns>
        public bool TrySetOnFuncRaised(Func<T0, T1, TResult> newFunc)
        {
            if (OnFuncRaised != null) {
                return false;
            }

            OnFuncRaised = newFunc;
            return true;
        }
    }
}
```

B.4 SceneLoaderManager

```
using FixingIt.Events;
using FixingIt.SceneManagement.ScriptableObjects;
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using System;
using UnityEngine;
using UnityEngine.ResourceManagement.AsyncOperations;
using UnityEngine.ResourceManagement.ResourceProviders;
using UnityEngine.SceneManagement;

namespace FixingIt.SceneManagement.Logic
{
    /// <summary>
    /// Persistent Manager, doesnt need an Instance or to clear the funcs
    /// </summary>
    public class SceneLoaderManager : MonoBehaviour
    {
        // filled after loaded one scene
        private GameSceneSO _sceneToLoad;
        private GameSceneSO _currentSceneLoaded;
        private bool _isSceneToLoadNetwork = false;
        private bool _isCurrentSceneNetwork = false;

        private float _necessaryWaitSeconds = 0.1f;

        [Header("Listening␣To")]
        [SerializeField]
        private LoadSceneChannelSO _loadSceneChannel;
        [SerializeField]
        private LoadSceneChannelSO _loadNetworkSceneChannel;
        [SerializeField]
        private VoidEventChannelSO _exitGameEvent;
        [SerializeField]
        private VoidEventChannelSO _unloadedNetworkEventCompleted;

        [Header("Broadcasting␣To")]
        [SerializeField] VoidEventChannelSO _startSceneLoadingEvent;
        [SerializeField] VoidEventChannelSO _sceneLoadedEvent;

        [Header("Invoking␣Func")]
        [SerializeField]
        private StringBoolFuncSO _loadNewNetworkSceneByNameFunc;
        [SerializeField]
        private StringBoolFuncSO _unloadNetworkSceneByNameFunc;

        [Header("Setting␣Func")]
        [SerializeField]
        private StringFuncSO _getCurrentSceneNameFunc;

        private void Awake()
        {
            //DontDestroyOnLoad(gameObject);

            _getCurrentSceneNameFunc.TrySetOnFuncRaised(() => _currentSceneLoaded.name);
        }

        private void OnEnable()
        {
            _loadSceneChannel.OnEventRaised += LoadScene;
            _loadNetworkSceneChannel.OnEventRaised += LoadNetworkScene;
            _exitGameEvent.OnEventRaised += ExitGame;

            _unloadedNetworkEventCompleted.OnEventRaised += SceneLoaderManager_UnloadedNetworkEventCompleted;
        }

        private void OnDisable()
        {
            _loadSceneChannel.OnEventRaised -= LoadScene;
            _loadNetworkSceneChannel.OnEventRaised -= LoadNetworkScene;
            _exitGameEvent.OnEventRaised -= ExitGame;

            _unloadedNetworkEventCompleted.OnEventRaised -= SceneLoaderManager_UnloadedNetworkEventCompleted;
        }

        #region Local Load
        private void LoadScene(GameSceneSO sceneToLoad)
        {
            // send event
            _startSceneLoadingEvent.RaiseEvent();

            _sceneToLoad = sceneToLoad;
            _isSceneToLoadNetwork = false;

            UnloadPreviousScene();
            LoadNewScene();
        }

        private void LoadNewScene()
        {
            // load scene
            var loadingOperationHandle = _sceneToLoad.SceneReference.LoadSceneAsync(LoadSceneMode.Additive);
            loadingOperationHandle.Completed += OnNewSceneLoaded;

            _isCurrentSceneNetwork = false;
        }

        private void OnNewSceneLoaded(AsyncOperationHandle<SceneInstance> obj)
        {
            _currentSceneLoaded = _sceneToLoad;

            Scene s = obj.Result.Scene;
            SceneManager.SetActiveScene(s);

            //LightProbes.TetrahedralizeAsync(); //not necessary

            // send event
            _sceneLoadedEvent.RaiseEvent();
        }
        #endregion

        #region Network Load
        private void LoadNetworkScene(GameSceneSO sceneToLoad)
        {
            // send event
            _startSceneLoadingEvent.RaiseEvent();

            _sceneToLoad = sceneToLoad;
            _isSceneToLoadNetwork = true;

            UnloadPreviousScene();

            // si hemos descargado una local
            if (!_isCurrentSceneNetwork)
            {
                Invoke(nameof(LoadNewNetworkScene), _necessaryWaitSeconds);  // necessary wait to prevent erro
            }
            // else => lo maneja UnloadPreviousSceneNetwork
        }

        /// <summary>
        /// Function that should be called after all the clients have unload async the previous networkScene
        /// </summary>
        private void LoadNewNetworkScene()
        {
            // load scene
            string sceneName = _sceneToLoad.name;
            bool hasNetSceneLoaded = false;
            try
            {
                hasNetSceneLoaded = _loadNewNetworkSceneByNameFunc.RaiseFunc(sceneName);
            }
            catch (Exception e)
            {
                ManageNetworkSceneLoaderExceptions(e);
            }

            if (hasNetSceneLoaded)
                _currentSceneLoaded = _sceneToLoad;

            _isCurrentSceneNetwork = hasNetSceneLoaded;
        }
        #endregion

        #region Unload
        /// <summary>
        /// Se podra hacer Coroutine para aadir  efecto de pantalla de carga
        /// </summary>
        private void UnloadPreviousScene()
        {
            // null when we are entering in the Main Menu
            if (_currentSceneLoaded == null)
            {
                return;
            }

            // check if it is managed by SceneManager or by NetworkSceneManager
            if (!_isCurrentSceneNetwork)
            {
                UnloadPreviousSceneLocal();
            }
            else
            {
                UnloadPreviousSceneNetwork();
            }
        }

        private void UnloadPreviousSceneLocal()
        {
            _currentSceneLoaded.SceneReference.UnLoadScene();
        }
        private void UnloadPreviousSceneNetwork()
        {
            string sceneName = _currentSceneLoaded.name;

            try
            {
                _unloadNetworkSceneByNameFunc.RaiseFunc(sceneName);
            }
            catch (Exception e)
            {
                ManageNetworkSceneLoaderExceptions(e);
            }
        }
        #endregion

        private void ExitGame()
        {
            Debug.Log("Quitting␣Application!");
            Application.Quit();
        }

        private void ManageNetworkSceneLoaderExceptions(Exception e)
        {
            Debug.LogError($"Error␣catched:␣{e.Message}");
        }

        private void SceneLoaderManager_UnloadedNetworkEventCompleted()
        {
            if (_isSceneToLoadNetwork)
            {
                Invoke(nameof(LoadNewNetworkScene), _necessaryWaitSeconds);  // necessary wait to prevent erro
            }
            // else -> managed by its function
        }
    }
}
```

B.5 NetworkSceneLoader

```
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using System;
using Unity.Netcode;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace FixingIt.SceneManagement.Logic
{
    public class NetworkSceneLoader : NetworkBehaviour
    {
        public static NetworkSceneLoader Instance { get; private set; }

        [Header("Broadcasting␣To")]
        [SerializeField]
        private VoidEventChannelSO _sceneLoadedEvent;
        [SerializeField]
        private VoidEventChannelSO _unloadedNetworkSceneEventCompleted;

        [Header("Setting␣Func")]
        [SerializeField]
        private StringBoolFuncSO _loadNewNetworkSceneByNameFunc;
        [SerializeField]
        private StringBoolFuncSO _unloadNetworkSceneByNameFunc;

        private void Awake()
        {
            if (Instance != null && Instance != this)
            {
                Destroy(this);
            }
            else
            {
                Instance = this;
                DontDestroyOnLoad(this);
            }

            // clear funcs
            _loadNewNetworkSceneByNameFunc.ClearOnFuncRaised();
            _unloadNetworkSceneByNameFunc.ClearOnFuncRaised();

            // set funcs
            _loadNewNetworkSceneByNameFunc.TrySetOnFuncRaised(LoadNewNetworkScene);
            _unloadNetworkSceneByNameFunc.TrySetOnFuncRaised(UnloadNetworkScene);
        }

        public override void OnNetworkSpawn()
        {
            if (IsServer)
            {
                NetworkManager.Singleton.SceneManager.OnSceneEvent += NetworkSceneLoader_OnNetworkSceneEvent;
            }
            Debug.Log($"OnNetworkSpawn␣and␣IsServer:␣{IsServer}");

            base.OnNetworkSpawn();
        }

        #region Load
        /// <summary>
        /// Function that should be called after all the clients have unload async the previous networkScene.
        /// Loads async a new NetworkScene
        /// </summary>
        /// <param name="sceneName">Name of the scene to load</param>
        /// <returns>If the Scene was loaded && You are server</returns>
        /// <exception cref="Exception">It was not possible to load the scene due to NetworkErrors</exception>
        private bool LoadNewNetworkScene(string sceneName)
        {
            if (!IsServer)
                return false;

            var status = NetworkManager.Singleton.SceneManager.LoadScene(sceneName, LoadSceneMode.Additive);
            if (status != SceneEventProgressStatus.Started)
            {
                string errorMsg = $"Failed␣to␣load␣{sceneName}␣with␣a␣{nameof(SceneEventProgressStatus)}:␣{status}";

                Debug.LogWarning(errorMsg);
                throw new Exception(errorMsg);
            }

            return true;
        }
        #endregion

        #region Unload
        /// <summary>
        /// Unloads a NetworkScene
        /// </summary>
        /// <param name="sceneName">Name of the scene to unload</param>
        /// <returns>If the Scene was unloaded && You are server</returns>
        /// <exception cref="Exception">It was not possible to unload the scene due to nameScene error</except
        private bool UnloadNetworkScene(string sceneName)
        {
            // TODO: mirar esto bien
            if (!IsServer)
                return false;

            Scene sceneToUnload = SceneManager.GetSceneByName(sceneName);
            if (!sceneToUnload.IsValid())
            {
                string errorMsg = $"UnloadingError:␣Escena␣con␣nombre␣{sceneName}␣no␣existe␣ahora␣mismo";

                Debug.LogWarning(errorMsg);
                throw new Exception(errorMsg);
            }

            NetworkManager.Singleton.SceneManager.UnloadScene(sceneToUnload);
            return true;
        }
        #endregion

        private void NetworkSceneLoader_OnNetworkSceneEvent(SceneEvent sceneEvent)
        {
            var clientOrServer = sceneEvent.ClientId == NetworkManager.ServerClientId ? "server" : "client";
            switch (sceneEvent.SceneEventType)
            {
                // locally cases
                case SceneEventType.LoadComplete:
                    {
                        //if (sceneEvent.ClientId == OwnerClientId) {
                        //    SceneManager.SetActiveScene(sceneEvent.Scene);
                        //}

                        Debug.Log($"Loaded␣the␣{sceneEvent.SceneName}␣scene␣on␣{clientOrServer}-({sceneEvent.C
                        break;
                    }
                case SceneEventType.UnloadComplete:
                    {
                        Debug.Log($"Unloaded␣the␣{sceneEvent.SceneName}␣scene␣on␣{clientOrServer}-({sceneEvent
                        break;
                    }
                // when server && all clients
                case SceneEventType.LoadEventCompleted:
                    {
                        Debug.Log($"Load␣event␣completed␣for␣the␣following␣client␣identifiers:({sceneEvent.Cli
                        if (sceneEvent.ClientsThatTimedOut.Count > 0)
                        {
                            Debug.LogWarning($"Load␣event␣timed␣out␣for␣the␣following␣client␣identifiers:({sceneEvent.
                        }

                        // send event
                        _sceneLoadedEvent.RaiseEvent();
                        break;
                    }
                case SceneEventType.UnloadEventCompleted:
                    {
                        Debug.Log($"Unload␣event␣completed␣for␣the␣following␣client␣identifiers:({sceneEvent.ClientsTh
                        if (sceneEvent.ClientsThatTimedOut.Count > 0)
                        {
                            Debug.LogWarning($"Unload␣event␣timed␣out␣for␣the␣following␣client␣identifiers:({sceneEven
                        }

                        // in theory this code is unreachable for clients but just in case
                        if (IsServer)
                        {
                            _unloadedNetworkSceneEventCompleted.RaiseEvent();
                        }

                        break;
                    }
            }
        }
    }
}
```

B.6 FixingGameMultiplayer NOT FINAL VERSION

```
using FixingIt.Events;
using FixingIt.Funcs;
using FixingIt.Minigame;
using FixingIt.PlayerGame;
using FixingIt.RoomObjects.Logic;
using FixingIt.RoomObjects.SO;
using FixingIt.SceneManagement.ScriptableObjects;
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using System.Linq;
using Unity.Netcode;
using Unity.Services.Authentication;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace FixingIt.Multiplayer
{
    public class FixingGameMultiplayer : NetworkBehaviour
    {
        private const string PLAYER_PREFS_PLAYER_NAME_MULTIPLAYER = "PlayerNameMultiplayer";

        public static FixingGameMultiplayer Instance { get; private set; }

        [SerializeField] private GameSceneSO _characterSelectionSceneSO;
        [SerializeField] private Color[] _playerColorArray;
        [SerializeField] private RoomObjectsSOListSO _allRoomObjectsSOListSO;

        private NetworkList<PlayerData> _playerDataNetworkList;
        private string _playerName;

        [Header("Broadcasting␣To")]
        [SerializeField]
        private VoidEventChannelSO _hostStartedEvent;
        [SerializeField]
        private StringEventChannelSO _rejectedToServerEvent;
        [SerializeField]
        private VoidEventChannelSO _playerDataNetworkListChangedEvent;
        [SerializeField]
        private VoidEventChannelSO _networkToMainMenuEvent;
        [SerializeField]
        private StringEventChannelSO _playerIdDisconnectedEvent;

        [Header("Listening␣To")]
        [SerializeField]
        private VoidEventChannelSO _lobbyCreatedEvent;
        [SerializeField]
        private VoidEventChannelSO _lobbyJoinedEvent;
        [SerializeField]
        private IntEventChannelSO _changePlayerColorId;
        [SerializeField]
        private VoidEventChannelSO _leaveGameToMainMenuEvent;
        [SerializeField]
        private ULongEventChannelSO _kickPlayerEvent;
        [SerializeField]
        private StringEventChannelSO _setPlayerNameEvent;
        [SerializeField]
        private TryToSpawnRoomObjectChannelSO _tryToSpawnRoomObjectEvent;

        [Header("Invoking␣Func")]
        [SerializeField]
        private StringFuncSO _getCurrentSceneNameFunc;

        [Header("Setting␣Func")]
        [SerializeField]
        private IntBoolFuncSO _isPlayerIndexConnected;
        [SerializeField]
        private IntPlayerdataFuncSO _getPlayerDataFromPlayerIndex;
        [SerializeField]
        private IntColorFuncSO _getPlayerColorFunc;
        [SerializeField]
        private PlayerdataFuncSO _getClientPlayerData;
        [SerializeField]
        private StringFuncSO _getPlayerNameFunc;
        //[SerializeField]
        //private SpawnRoomObjectFuncSO _spawnRoomObjectFunc;

        private void Awake()
        {
            if (Instance != null && Instance != this)
            {
                Destroy(this);
            }
            else
            {
                Instance = this;
                DontDestroyOnLoad(gameObject);
            }
            // si sobra tiempo, refactorizar a un FuncSO y que el sistema de guardado sea externo
            _playerName = PlayerPrefs.GetString(PLAYER_PREFS_PLAYER_NAME_MULTIPLAYER, $"PlayerName{Random.Range(100,␣1

            _playerDataNetworkList = new NetworkList<PlayerData>(readPerm: NetworkVariableReadPermission.Everyone);
            _playerDataNetworkList.OnListChanged += OnPlayerDataNetworkListChanged;

            // clear funcs just in case
            _isPlayerIndexConnected.ClearOnFuncRaised();
            _getPlayerDataFromPlayerIndex.ClearOnFuncRaised();
            _getPlayerColorFunc.ClearOnFuncRaised();
            _getClientPlayerData.ClearOnFuncRaised();
            _getPlayerNameFunc.ClearOnFuncRaised();
            //_spawnRoomObjectFunc.ClearOnFuncRaised();

            // set funcs
            _isPlayerIndexConnected.TrySetOnFuncRaised(IsPlayerIndexConnected);
            _getPlayerDataFromPlayerIndex.TrySetOnFuncRaised(GetPlayerDataFromPlayerIndex);
            _getPlayerColorFunc.TrySetOnFuncRaised(GetPlayerColor);
            _getClientPlayerData.TrySetOnFuncRaised(GetPlayerData);
            _getPlayerNameFunc.TrySetOnFuncRaised(GetPlayerName);
            //_spawnRoomObjectFunc.TrySetOnFuncRaised(SpawnRoomObject);
        }

        private void OnEnable()
        {
            _lobbyCreatedEvent.OnEventRaised += StartHost;
            _lobbyJoinedEvent.OnEventRaised += StartClient;

            _changePlayerColorId.OnEventRaised += ChangePlayerColor;

            _leaveGameToMainMenuEvent.OnEventRaised += LeaveToMainMenu;

            _kickPlayerEvent.OnEventRaised += KickPlayer;

            _setPlayerNameEvent.OnEventRaised += SetPlayerName;

            _tryToSpawnRoomObjectEvent.OnEventRaised += SpawnRoomObject;
        }

        private void OnDisable()
        {
            _lobbyCreatedEvent.OnEventRaised -= StartHost;
            _lobbyJoinedEvent.OnEventRaised -= StartClient;

            _changePlayerColorId.OnEventRaised -= ChangePlayerColor;

            _leaveGameToMainMenuEvent.OnEventRaised -= LeaveToMainMenu;

            _kickPlayerEvent.OnEventRaised -= KickPlayer;

            _setPlayerNameEvent.OnEventRaised -= SetPlayerName;

            _tryToSpawnRoomObjectEvent.OnEventRaised -= SpawnRoomObject;
        }

        private void StartHost()
        {
            NetworkManager.Singleton.ConnectionApprovalCallback += NetworkManager_ConnectionApprovalCallback;
            NetworkManager.Singleton.OnClientConnectedCallback += NetworkManager_OnClientConnectedCallback;
            NetworkManager.Singleton.OnClientDisconnectCallback += NetworkManager_Server_OnClientDisconnectCal
            NetworkManager.Singleton.StartHost();

            // send event
            _hostStartedEvent.RaiseEvent();
        }

        private void StartClient()
        {
            NetworkManager.Singleton.OnClientConnectedCallback += NetworkManager_Client_OnClientConnectedCallb
            NetworkManager.Singleton.OnClientDisconnectCallback += NetworkManager_Client_OnClientDisconnectCal
            NetworkManager.Singleton.StartClient();
        }

        private void OnPlayerDataNetworkListChanged(NetworkListEvent<PlayerData> changeEvent)
        {
            _playerDataNetworkListChangedEvent.RaiseEvent();
        }

        private void LeaveToMainMenu()
        {
            NetworkManager.Singleton.Shutdown();

            // send event
            _networkToMainMenuEvent.RaiseEvent();
        }

        private void KickPlayer(ulong clientId)
        {
            NetworkManager.Singleton.DisconnectClient(clientId);
            NetworkManager_Server_OnClientDisconnectCallback(clientId);
        }

        #region Player Info
        private bool IsPlayerIndexConnected(int playerIndex)
        {
            //if (!IsServer)
            //return false;

            return playerIndex < _playerDataNetworkList.Count;
        }

        private PlayerData GetPlayerDataFromPlayerIndex(int playerIndex)
        {
            return _playerDataNetworkList[playerIndex];
        }

        private PlayerData GetPlayerData()
        {
            return GetPlayerDataFromClientId(NetworkManager.Singleton.LocalClientId);
        }

        private int GetPlayerDataIndexFromClientId(ulong clientId)
        {
            for (int i = 0; i < _playerDataNetworkList.Count; i++) {
                if (_playerDataNetworkList[i].ClientId == clientId) {
                    return i;
                }
            }

            return -1;
        }

        private PlayerData GetPlayerDataFromClientId(ulong clientId)
        {
            foreach (PlayerData playerData in _playerDataNetworkList) {
                if (playerData.ClientId == clientId) {
                    return playerData;
                }
            }

            return default;
        }

        private Color GetPlayerColor(int colorId)
        {
            return _playerColorArray[colorId];
        }

        private void ChangePlayerColor(int colorId)
        {
            ChangePlayerColorServerRpc(colorId);
        }

        [ServerRpc(RequireOwnership = false)]
        private void ChangePlayerColorServerRpc(int colorId, ServerRpcParams serverRpcParams = default)
        {
            if (!IsColorAvailable(colorId)) {
                return;
            }

            // get playerdata struct. We cannot modify directly the clientId in a NetworkList
            int playerDataIndex = GetPlayerDataIndexFromClientId(serverRpcParams.Receive.SenderClientId);
            PlayerData playerData = _playerDataNetworkList[playerDataIndex];

            // set playerdata struct
            playerData.ColorId = colorId;
            _playerDataNetworkList[playerDataIndex] = playerData;
        }

        private string GetPlayerName()
        {
            return _playerName;
        }

        private void SetPlayerName(string playerName)
        {
            // no dejar nombres vacios
            if (playerName == string.Empty)
                return;

            _playerName = playerName;
            // si sobra tiempo, refactorizar a un FuncSO y que el sistema de guardado sea externo
            PlayerPrefs.SetString(PLAYER_PREFS_PLAYER_NAME_MULTIPLAYER, playerName);
        }

        [ServerRpc(RequireOwnership = false)]
        private void SetPlayerNameServerRpc(string playerName, ServerRpcParams serverRpcParams = default)
        {
            // get playerdata struct. We cannot modify directly the clientId in a NetworkList
            int playerDataIndex = GetPlayerDataIndexFromClientId(serverRpcParams.Receive.SenderClientId);
            PlayerData playerData = _playerDataNetworkList[playerDataIndex];

            // set playerdata struct
            playerData.PlayerName = playerName;
            _playerDataNetworkList[playerDataIndex] = playerData;
        }

        [ServerRpc(RequireOwnership = false)]
        private void SetPlayerIdServerRpc(string playerId, ServerRpcParams serverRpcParams = default)
        {
            // get playerdata struct. We cannot modify directly the clientId in a NetworkList
            int playerDataIndex = GetPlayerDataIndexFromClientId(serverRpcParams.Receive.SenderClientId);
            PlayerData playerData = _playerDataNetworkList[playerDataIndex];

            // set playerdata struct
            playerData.PlayerId = playerId;
            _playerDataNetworkList[playerDataIndex] = playerData;
        }
        #endregion

        #region Color
        private bool IsColorAvailable(int colorId)
        {
            foreach (PlayerData playerData in _playerDataNetworkList) {
                if (playerData.ColorId == colorId) {
                    // already in use
                    return false;
                }
            }

            return true;
        }

        private int GetFirstUnusedColorId()
        {
            for (int i = 0; i < _playerColorArray.Length; i++) {
                if (IsColorAvailable(i)) {
                    return i;
                }
            }

            return -1;
        }
        #endregion

        #region Minigame
        // TODO: cambiar por sistema de pooling como extra
        private void SpawnRoomObject(RoomObjectSO roomObjectSO, NetworkObjectReference roomObjectParentNORef)
        {
            SpawnRoomObjectServerRpc(GetRoomObjectSOIndex(roomObjectSO), roomObjectParentNORef);
        }

        [ServerRpc(RequireOwnership = false)]
        private void SpawnRoomObjectServerRpc(int roomObjectSOIndex, NetworkObjectReference roomObjectParentNORef)
        {
            GameObject roomObjectGO = Instantiate(GetRoomObjectSOFromIndex(roomObjectSOIndex).RoomObjectPrefab);

            NetworkObject roomObjectNO = roomObjectGO.GetComponent<NetworkObject>();
            roomObjectNO.Spawn();

            RoomObject roomObject = roomObjectGO.GetComponent<RoomObject>();

            roomObjectParentNORef.TryGet(out NetworkObject roomObjectParentNO);
            IRoomObjectParent roomObjectParent = roomObjectParentNO.GetComponent<IRoomObjectParent>();
            roomObject.SetRoomObjectParent(roomObjectParent);
        }

        private int GetRoomObjectSOIndex(RoomObjectSO roomObjectSO)
        {
            return _allRoomObjectsSOListSO.RoomObjectsSO.IndexOf(roomObjectSO);
        }

        private RoomObjectSO GetRoomObjectSOFromIndex(int index)
        {
            return _allRoomObjectsSOListSO.RoomObjectsSO[index];
        }
        #endregion

        #region NetworkCallbacks
        private void NetworkManager_ConnectionApprovalCallback(NetworkManager.ConnectionApprovalRequest connec
            NetworkManager.ConnectionApprovalResponse connectionApprovalResponse)
        {
            string rejectedReason = string.Empty;

            // server no est en characerselection
            int numberOfActiveScenes = SceneManager.sceneCount;
            string[] activeSceneNames = new string[numberOfActiveScenes];
            for (int i = 0; i < SceneManager.sceneCount; i++)
            {
                activeSceneNames[i] = SceneManager.GetSceneAt(i).name;
            }

            if (!activeSceneNames.Contains(_characterSelectionSceneSO.name))
            {
                connectionApprovalResponse.Approved = false;
                connectionApprovalResponse.Reason = "Game␣has␣already␣started!";

                return;
            }

            connectionApprovalResponse.Approved = true;
        }

        /// <summary>
        /// Only subscribed by the host
        /// Host manages what to do when a client connects
        /// </summary>
        /// <param name="clientId">clientId that has connected</param>
        private void NetworkManager_OnClientConnectedCallback(ulong clientId)
        {
            _playerDataNetworkList.Add(new PlayerData()
            {
                ClientId = clientId,
                ColorId = GetFirstUnusedColorId(),
                //PlayerName = _playerName,
            });
            SetPlayerNameServerRpc(GetPlayerName());
        }

        private void NetworkManager_Server_OnClientDisconnectCallback(ulong clientId)
        {
            for (int i = 0; i < _playerDataNetworkList.Count; i++) {
                PlayerData playerData = _playerDataNetworkList[i];
                if (playerData.ClientId == clientId) {
                    // Disconnected
                    _playerDataNetworkList.RemoveAt(i);

                    string playerId = playerData.PlayerId.ToString();
                    _playerIdDisconnectedEvent.RaiseEvent(playerId);
                }
            }
        }

        private void NetworkManager_Client_OnClientConnectedCallback(ulong clientId)
        {
            SetPlayerNameServerRpc(GetPlayerName());
            SetPlayerIdServerRpc(AuthenticationService.Instance.PlayerId);
        }

        private void NetworkManager_Client_OnClientDisconnectCallback(ulong obj)
        {
            _rejectedToServerEvent.RaiseEvent(NetworkManager.Singleton.DisconnectReason);
        }
        #endregion
    }
}
```

B.7 LobbyManager

```
using FixingIt.Events;
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using System.Collections.Generic;
using System.Threading.Tasks;
using Unity.Netcode;
using Unity.Netcode.Transports.UTP;
using Unity.Networking.Transport.Relay;
using Unity.Services.Authentication;
using Unity.Services.Core;
using Unity.Services.Lobbies;
using Unity.Services.Lobbies.Models;
using Unity.Services.Relay;
using Unity.Services.Relay.Models;
using UnityEngine;

namespace FixingIt.GameLobby
{
    // It is a Singleton just to work through scenes and delete it whenever I want
    public class LobbyManager : MonoBehaviour
    {
        private const int MAX_PLAYER_AMOUNT = 4;
        private const string DTLS_CONNECTION_TYPE = "dtls"; //type that Unity docummentation recommends
        private const string RELAY_JOIN_CODE = "RelayJoinCode";

        private const string CREATE_LOBBY_MSG = "Creating␣lobby...";
        private const string CREATE_RELAY_MSG = "Creating␣relay...";
        private const string JOIN_LOBBY_MSG = "Joining␣lobby...";
        private const string JOIN_RELAY_MSG = "Joining␣relay...";
        private const string SEARCHING_LOBBIES_MSG = "Searching␣for␣public␣lobbies...";

        public static LobbyManager Instance { get; private set;}

        private const string PLAYER_NAME = "PlayerName";

        private Lobby _hostLobby;
        private Lobby _joinedLobby;
        private float _heartbeatTimer;
        private float _lobbyUpdateTimer;
        private string _playerName;

        [Header("Broadcasting␣To")]
        [SerializeField]
        private LobbiesChannelSO _lobbiesListedEvent;
        [SerializeField]
        private VoidEventChannelSO _lobbyCreatedEvent;
        [SerializeField]
        private VoidEventChannelSO _lobbyJoinedEvent;
        [SerializeField]
        private StringEventChannelSO _lobbyErrorCatchedEvent;
        [SerializeField]
        private StringEventChannelSO _lobbyStateUpdated;

        [Header("Listening␣To")]
        [SerializeField]
        private VoidEventChannelSO _refreshLobbiesListEvent;
        [SerializeField]
        private CreateLobbyChannelSO _createLobbyChannel;
        [SerializeField]
        private StringEventChannelSO _joinByIdEvent;
        [SerializeField]
        private StringEventChannelSO _joinByCodeEvent;
        [SerializeField]
        private VoidEventChannelSO _allPlayersReadyEvent;
        [SerializeField]
        private VoidEventChannelSO _toMainMenuScreenEvent;
        [SerializeField]
        private VoidEventChannelSO _leaveGameToMainMenuEvent;
        [SerializeField]
        private StringEventChannelSO _kickPlayerPlayerIdEvent;
        [SerializeField]
        private StringEventChannelSO _playerIdDisconnectedEvent;

        [Header("Setting␣Func")]
        [SerializeField]
        private StringFuncSO _getLobbyNameFunc;
        [SerializeField]
        private StringFuncSO _getLobbyCodeFunc;

        private void Awake()
        {
            if (Instance != null && Instance != this) {
                Destroy(this);
            }
            else {
                Instance = this;
                DontDestroyOnLoad(gameObject);
            }

            // clear funcs just in case
            _getLobbyNameFunc.ClearOnFuncRaised();
            _getLobbyCodeFunc.ClearOnFuncRaised();

            // set funcs
            _getLobbyNameFunc.TrySetOnFuncRaised(() => _joinedLobby.Name);
            _getLobbyCodeFunc.TrySetOnFuncRaised(() => _joinedLobby.LobbyCode);
        }

        private void OnEnable()
        {
            _refreshLobbiesListEvent.OnEventRaised += ListLobbies;
            _createLobbyChannel.OnEventRaised += CreateLobby;
            _joinByIdEvent.OnEventRaised += JoinLobbyById;
            _joinByCodeEvent.OnEventRaised += JoinLobbyByCode;

            _allPlayersReadyEvent.OnEventRaised += DeleteLobby;

            _toMainMenuScreenEvent.OnEventRaised += LeaveLobby;
            _leaveGameToMainMenuEvent.OnEventRaised += LeaveLobby;

            //_kickPlayerPlayerIdEvent.OnEventRaised += KickPlayer;
            _playerIdDisconnectedEvent.OnEventRaised += KickPlayer;
        }

        private void OnDisable()
        {
            _refreshLobbiesListEvent.OnEventRaised -= ListLobbies;
            _createLobbyChannel.OnEventRaised -= CreateLobby;
            _joinByIdEvent.OnEventRaised -= JoinLobbyById;
            _joinByCodeEvent.OnEventRaised -= JoinLobbyByCode;

            _allPlayersReadyEvent.OnEventRaised -= DeleteLobby;

            _toMainMenuScreenEvent.OnEventRaised -= LeaveLobby;
            _leaveGameToMainMenuEvent.OnEventRaised -= LeaveLobby;

            //_kickPlayerPlayerIdEvent.OnEventRaised -= KickPlayer;
            _playerIdDisconnectedEvent.OnEventRaised -= KickPlayer;
        }

        private async void Start()
        {
            // to prevent initialize and signin when we are signed
            if (UnityServices.State != ServicesInitializationState.Initialized)
            {
                InitializationOptions initializationOptions = new InitializationOptions();
                initializationOptions.SetProfile(Random.Range(0, 1000).ToString()); // to allow test with mult

                await UnityServices.InitializeAsync(initializationOptions);

                await AuthenticationService.Instance.SignInAnonymouslyAsync();
            }

            _playerName = $"Guest{Random.Range(1000,␣9999)}";   // It is not unique
            Debug.Log(_playerName);

            ListLobbies();
        }

        private void Update()
        {
            HandleLobbyHeartBeat();
            //HandleLobbyPollForUpdates();
        }

        #region HandleLobby
        private async void HandleLobbyHeartBeat()
        {
            if (!IsLobbyHost())
```

```
                return;

            _heartbeatTimer -= Time.deltaTime;
            if (_heartbeatTimer < 0f)
            {
                float heartbeatTimerMax = 15f;
                _heartbeatTimer = heartbeatTimerMax;

                await LobbyService.Instance.SendHeartbeatPingAsync(_joinedLobby.Id);
            }
        }

        private async void HandleLobbyPollForUpdates()
        {
            if (!IsLobbyHost())
                return;

            _lobbyUpdateTimer -= Time.deltaTime;
            if (_lobbyUpdateTimer < 0f)
            {
                float lobbyUpdateTimerMax = 1.1f;
                _lobbyUpdateTimer = lobbyUpdateTimerMax;

                Lobby lobby = await LobbyService.Instance.GetLobbyAsync(_joinedLobby.Id);
                _joinedLobby = lobby;

                // TODO: send event to Update UI
            }
        }
        #endregion

        private bool IsLobbyHost()
        {
            return _joinedLobby != null
                && _joinedLobby.HostId == AuthenticationService.Instance.PlayerId;
        }

        private async void ListLobbies()
        {
            try
            {
                _lobbyStateUpdated.RaiseEvent(SEARCHING_LOBBIES_MSG);

                // lobbies with al least 1 slot and sorted in created order
                QueryLobbiesOptions queryLobbiesOptions = new QueryLobbiesOptions
                {
                    Filters = new List<QueryFilter> {
                    new QueryFilter(QueryFilter.FieldOptions.AvailableSlots, "0", QueryFilter.OpOptions.GT) // con al
                    // I can put another filter to compare the data, for example for the GameMode, not necessary in my
                    //new QueryFilter(QueryFilter.FieldOptions.S1, "SpeedFix", QueryFilter.OpOptions.EQ)
                },
                    Order = new List<QueryOrder> {
                    new QueryOrder (false, QueryOrder.FieldOptions.Created)
                }
                };

                QueryResponse queryResponse = await Lobbies.Instance.QueryLobbiesAsync(queryLobbiesOptions);

                // send event
                _lobbiesListedEvent.RaiseEvent(queryResponse.Results);
            }
            catch (LobbyServiceException e)
            {
                ManageLobbyErrors(e);
            }
        }

        #region Relay
        private async Task<Allocation> AllocateRelay()
        {
            try {
                // max players - the host
                Allocation allocation = await RelayService.Instance.CreateAllocationAsync(MAX_PLAYER_AMOUNT -

                return allocation;
            }
            catch (RelayServiceException e) {
                ManageRelayErrors(e);

                return default;
            }
        }

        private async Task<string> GetRelayJoinCode(Allocation allocation)
        {
            try {

                string relayJoinCode = await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);

                return relayJoinCode;
            }
            catch(RelayServiceException e) {
                ManageRelayErrors(e);

                return default;
            }
        }

        private async Task<JoinAllocation> JoinRelay(string joinCode)
        {
            try {
                JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(joinCode);

                return joinAllocation;
            }
            catch (RelayServiceException e) {
                ManageRelayErrors(e);

                return default;
            }
        }

        private async Task JoinRelayTransport(Lobby lobbyJoined)
        {
            string relayJoinCode = lobbyJoined.Data[RELAY_JOIN_CODE].Value;
            JoinAllocation joinAllocation = await JoinRelay(relayJoinCode);

            NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(new RelayServerData(joinAllocat
        }
        #endregion

        private async void CreateLobby(string lobbyName, bool isPrivate)
        {
            try
            {
                _lobbyStateUpdated.RaiseEvent(CREATE_LOBBY_MSG);

                CreateLobbyOptions createLobbyOptions = new CreateLobbyOptions
                {
                    IsPrivate = isPrivate,
                    Player = GetPlayer(),
                };

                Lobby lobby = await LobbyService.Instance.CreateLobbyAsync(lobbyName, MAX_PLAYER_AMOUNT, createLobbyOp

                _hostLobby = lobby;
                _joinedLobby = _hostLobby;

                Debug.Log($"Created␣lobby!␣{lobby.Name},␣{lobby.MaxPlayers},␣{lobby.Id},␣{lobby.LobbyCode}");

                // create relay
                _lobbyStateUpdated.RaiseEvent(CREATE_RELAY_MSG);

                Allocation allocation = await AllocateRelay();

                string relayJoinCode = await GetRelayJoinCode(allocation);

                await LobbyService.Instance.UpdateLobbyAsync(_joinedLobby.Id, new UpdateLobbyOptions {
                   Data = new Dictionary<string, DataObject> {
                       {RELAY_JOIN_CODE, new DataObject(DataObject.VisibilityOptions.Member, relayJoinCode)}
                   }
                });

                NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(new RelayServerData

                // send event
                _lobbyCreatedEvent.RaiseEvent();
            }
            catch (LobbyServiceException e)
            {
                ManageLobbyErrors(e);
            }
        }

        private async void JoinLobbyById(string lobbyId)
        {
            try
            {
                _lobbyStateUpdated.RaiseEvent(JOIN_LOBBY_MSG);

                JoinLobbyByIdOptions joinLobbyByIdOptions = new JoinLobbyByIdOptions
                {
                    Player = GetPlayer()
                };

                Lobby lobby = await Lobbies.Instance.JoinLobbyByIdAsync(lobbyId, joinLobbyByIdOptions);
                _joinedLobby = lobby;

                Debug.Log($"Joined Lobby with id: {lobbyId}");

                // join relay
                _lobbyStateUpdated.RaiseEvent(JOIN_RELAY_MSG);

                await JoinRelayTransport(_joinedLobby);

                // send event
                _lobbyJoinedEvent.RaiseEvent();
            }
            catch (LobbyServiceException e)
            {
                ManageLobbyErrors(e);
            }
        }

        private async void JoinLobbyByCode(string lobbyCode)
        {
            try
            {
                _lobbyStateUpdated.RaiseEvent(JOIN_LOBBY_MSG);

                JoinLobbyByCodeOptions joinLobbyByCodeOptions = new JoinLobbyByCodeOptions
                {
                    Player = GetPlayer()
                };

                Lobby lobby = await Lobbies.Instance.JoinLobbyByCodeAsync(lobbyCode, joinLobbyByCodeOptions);
                _joinedLobby = lobby;

                // join relay
                _lobbyStateUpdated.RaiseEvent(JOIN_RELAY_MSG);

                await JoinRelayTransport(_joinedLobby);

                // send event
                _lobbyJoinedEvent.RaiseEvent();
            }
            catch (LobbyServiceException e)
            {
                ManageLobbyErrors(e);
            }
        }

        private async void DeleteLobby()
        {
            if (!IsLobbyHost())
                return;

            try {
                await LobbyService.Instance.DeleteLobbyAsync(_joinedLobby.Id);

                _joinedLobby = null;
            }
            catch (LobbyServiceException e) {
                ManageLobbyErrors(e);
            }
        }

        private async void LeaveLobby()
        {
            if (_joinedLobby == null)
                return;

            try {
                //if (IsLobbyHost()) {
                //    DeleteLobby();
                //}

                await LobbyService.Instance.RemovePlayerAsync(_joinedLobby.Id, AuthenticationService.Instance.PlayerI

                _joinedLobby = null;
            }
            catch (LobbyServiceException e) {
                ManageLobbyErrors(e);
            }
        }

        // When player kicked, managed by the FixingGameMultiplayer, NetworkManager
        private async void KickPlayer(string playerId)
        {
            if (!IsLobbyHost())
                return;

            try {
                await LobbyService.Instance.RemovePlayerAsync(_joinedLobby.Id, playerId);
            }
            catch (LobbyServiceException e) {
                ManageLobbyErrors(e);
            }
        }

        #region PlayerLobby
        private Player GetPlayer()
        {
            return new Player
            {
                Data = new Dictionary<string, PlayerDataObject> {
                {PLAYER_NAME, new PlayerDataObject(PlayerDataObject.VisibilityOptions.Member, _playerName) }
            }
            };
        }
        #endregion

        private void ManageLobbyErrors(LobbyServiceException e)
        {
            Debug.LogException(e);

            string userErrorMsg = e.Message;
            userErrorMsg = userErrorMsg[0].ToString().ToUpper() + userErrorMsg.Substring(1);

            _lobbyErrorCatchedEvent.RaiseEvent(userErrorMsg);
        }

        private void ManageRelayErrors(RelayServiceException e)
        {
            Debug.LogException(e);

            string userErrorMsg = e.Message;
            userErrorMsg = userErrorMsg[0].ToString().ToUpper() + userErrorMsg.Substring(1);

            _lobbyErrorCatchedEvent.RaiseEvent(userErrorMsg);
        }
    }
}
```

B.8 CharacterSelectionPlayer

```
using FixingIt.Funcs;
using FixingIt.PlayerGame;
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using TMPro;
using Unity.Netcode;
using UnityEngine;
using UnityEngine.UI;

namespace FixingIt.CharacterSelection
{
    public class CharacterSelectionPlayer : MonoBehaviour
    {
        [SerializeField] private int _playerIndex;
        [SerializeField] private GameObject _readyGameObject;
        [SerializeField] private TextMeshPro _playerNameText;
        [SerializeField] private PlayerVisualComp _playerVisualComp;
        [SerializeField] private Button _kickButton;

        [Header("Broadcasting␣To")]
        [SerializeField]
        private ULongEventChannelSO _kickPlayerClientIdEvent;
        [SerializeField]
        private StringEventChannelSO _kickPlayerPlayerIdEvent;

        [Header("Listening␣To")]
        [SerializeField]
        private VoidEventChannelSO _playerDataNetworkListChangedEvent;
        [SerializeField]
        private VoidEventChannelSO _clientReadyChangedEvent;

        [Header("Invoking␣Func")]
        [SerializeField]
        private IntBoolFuncSO _isPlayerIndexConnectedFunc;
        [SerializeField]
        private ULongBoolFuncSO _isPlayerReadyFunc;
        [SerializeField]
        private IntPlayerdataFuncSO _getPlayerDataFromPlayerIndexFunc;
        [SerializeField]
        private IntColorFuncSO _getPlayerColorFunc;
        [SerializeField]
        private StringFuncSO _getPlayerNameFunc;

        private void Awake()
        {
            _kickButton.onClick.AddListener(KickPlayer);
        }

        private void Start()
        {
            _playerDataNetworkListChangedEvent.OnEventRaised += UpdateCSPlayer;
            _clientReadyChangedEvent.OnEventRaised += UpdateCSPlayer;

            // server is 0
            _kickButton.gameObject.SetActive(NetworkManager.Singleton.IsServer && _playerIndex != 0);

            UpdateCSPlayer();
        }

        private void OnDestroy()
        {
            _playerDataNetworkListChangedEvent.OnEventRaised -= UpdateCSPlayer;
            _clientReadyChangedEvent.OnEventRaised -= UpdateCSPlayer;
        }

        private void UpdateCSPlayer()
        {
            if (_isPlayerIndexConnectedFunc.RaiseFunc(_playerIndex))
            {
                Show();

                // show ready
                PlayerData playerData = _getPlayerDataFromPlayerIndexFunc.RaiseFunc(_playerIndex);
                bool isReady = _isPlayerReadyFunc.RaiseFunc(playerData.ClientId);
                _readyGameObject.SetActive(isReady);

                // show name
                _playerNameText.text = playerData.PlayerName.ToString();

                // show color
                Color playerColor = _getPlayerColorFunc.RaiseFunc(playerData.ColorId);
                _playerVisualComp.SetPlayerColor(playerColor);
            }
            else
            {
                Hide();
            }
        }

        private void KickPlayer()
        {
            PlayerData playerData = _getPlayerDataFromPlayerIndexFunc.RaiseFunc(_playerIndex);
            //_kickPlayerPlayerIdEvent.RaiseEvent(playerData.PlayerId.ToString());
            _kickPlayerClientIdEvent.RaiseEvent(playerData.ClientId);
        }

        private void Show()
        {
            gameObject.SetActive(true);
        }

        private void Hide()
        {
            gameObject.SetActive(false);
        }
    }
}
```

B.9 CharacterSelectionManager

```
using ProgramadorCastellano.Events;
using ProgramadorCastellano.Funcs;
using System.Collections.Generic;
using Unity.Netcode;
using UnityEngine;

namespace FixingIt.CharacterSelection
{
    public class CharacterSelectionManager : NetworkBehaviour
    {
        [SerializeField] private float _timeCountdownMax = 5f;
        private float _timeCountdownTimer;

        private Dictionary<ulong, bool> _playerReadyDictionary;

        private bool _allPlayersReady;

        [Header("Broadcasting␣To")]
        [SerializeField]
        private VoidEventChannelSO _allPlayersReadyEvent;
        [SerializeField]
        private VoidEventChannelSO _clientReadyChangedEvent;
        [SerializeField]
        private FloatEventChannelSO _countdownEvent;
        [SerializeField]
        private VoidEventChannelSO _allPlayersReadyCancelledEvent;

        [Header("Listening␣To")]
        [SerializeField]
        private VoidEventChannelSO _readyButtonEvent;

        [Header("Setting␣Func")]
        [SerializeField]
        private ULongBoolFuncSO _isPlayerReadyFunc;

        private void Awake()
        {
            _timeCountdownTimer = _timeCountdownMax;

            _playerReadyDictionary = new Dictionary<ulong, bool>();
            _allPlayersReady = false;

            // clear func just in case
            _isPlayerReadyFunc.ClearOnFuncRaised();

            // set func
            _isPlayerReadyFunc.TrySetOnFuncRaised(IsPlayerReady);
        }

        private void OnEnable()
        {
            _readyButtonEvent.OnEventRaised += TogglePlayerReady;
        }

        private void OnDisable()
        {
            _readyButtonEvent.OnEventRaised -= TogglePlayerReady;
        }

        private void Update()
        {
            HandleCountdown();
        }

        #region SetPlayerReady
        private void TogglePlayerReady()
        {
            TogglePlayerReadyServerRpc();
        }

        [ServerRpc(RequireOwnership = false)]
        private void TogglePlayerReadyServerRpc(ServerRpcParams serverRpcParams = default)
        {
            bool isPlayerReady = true;

            if (_playerReadyDictionary.ContainsKey(serverRpcParams.Receive.SenderClientId)) {
                isPlayerReady = !_playerReadyDictionary[serverRpcParams.Receive.SenderClientId];
            }

            SetPlayerReadyClientRpc(isPlayerReady, serverRpcParams.Receive.SenderClientId);
            _playerReadyDictionary[serverRpcParams.Receive.SenderClientId] = isPlayerReady;

            // check if all clients are ready
            bool allClientsReady = true;
            foreach (ulong clientId in NetworkManager.Singleton.ConnectedClientsIds) {
                if (!_playerReadyDictionary.ContainsKey(clientId)
                    || !_playerReadyDictionary[clientId])
                {
```

```
                // this player is not ready
                allClientsReady = false;
                break;
            }
        }

        if (allClientsReady) {
            _timeCountdownTimer = _timeCountdownMax;
            _allPlayersReady = true;
        }
        else {
            _allPlayersReady = false;

            AllPlayersReadyCancelledClientRpc();
            // send event
            //_allPlayersReadyCancelledEvent.RaiseEvent();
        }
    }

    [ClientRpc]
    private void AllPlayersReadyCancelledClientRpc()
    {
        _allPlayersReadyCancelledEvent.RaiseEvent();
    }

    [ClientRpc]
    private void SetPlayerReadyClientRpc(bool isPlayerReady, ulong clientId)
    {
        _playerReadyDictionary[clientId] = isPlayerReady;

        // send event
        _clientReadyChangedEvent.RaiseEvent();
    }
    #endregion

    private bool IsPlayerReady(ulong clientId)
    {
        return _playerReadyDictionary.ContainsKey(clientId) && _playerReadyDictionary[clientId];
    }

    private void HandleCountdown()
    {
        if (!_allPlayersReady) {
            return;
        }

        _timeCountdownTimer -= Time.deltaTime;
        RaiseCountdownEventClientRpc(_timeCountdownTimer);
        if (_timeCountdownTimer < 0f) {

            // send event
```

```
                _allPlayersReadyEvent.RaiseEvent();
            }
        }

        [ClientRpc]
        private void RaiseCountdownEventClientRpc(float remainingTime)
        {
            _countdownEvent.RaiseEvent(remainingTime);
        }
    }
}
```

B.10 FixingGameManager

```
using FixingIt.Counters;
using FixingIt.Customer;
using FixingIt.Events;
using FixingIt.Funcs;
using FixingIt.InputSystem;
using FixingIt.RoomObjects.Logic;
using FixingIt.RoomObjects.SO;
using ProgramadorCastellano.Events;
using System.Collections.Generic;
using System.Linq;
using Unity.Netcode;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace FixingIt.Minigame
{
    public class FixingGameManager : NetworkBehaviour
    {
        private enum GameState
        {
            WaitingToStart,
            Playing,
            End
        }

        [SerializeField] InputReaderSO _inputReaderSO;

        [SerializeField] ToolRecipeManagerSO _levelToolRecipeManagerSO;
        [SerializeField] Transform _baseTransformToSpawn;

        private float _waitingToStartTimer;
        private float _gameplayTimer;
        private float _customerSpawnerTimer;
        //private GameState _gameState;
        private NetworkVariable<GameState> _gameState = new NetworkVariable<GameState>(GameState.WaitingToStart);

        private NetworkVariable<int> _numberObjectsFixed = new NetworkVariable<int>(0);

        [Header("Player")]
        [SerializeField] GameObject _playerPrefab;
        [SerializeField] Transform[] _playerSpawnPositions;

        [Header("Timers")]
        [SerializeField] private float _waitingToStartTimerMax = 5f;
        [SerializeField] private float _gameplayTimerMax = 60f;
        [SerializeField] private float _customerSpawnerTimerMax = 10f;

        [Header("Customers")]
        [SerializeField] GameObject _customerPrefab;
        [SerializeField] Transform _customerStartPosition;

        [Header("Customer␣Counters")]
        [SerializeField]
        private CustomerCounter[] _customerCounters;
        [SerializeField]
        private RoomObjectSO[] _objectsToFixSO;
        public int TestIndex;

        [Header("Broadcasting␣To")]
        [SerializeField]
        private FloatEventChannelSO _waitingToStartTimerEvent;
        [SerializeField]
        private FloatEventChannelSO _gameplayTimerNormalizedEvent;
        [SerializeField]
        private IntEventChannelSO _numberObjectsFixedEvent;

        [Header("Listening␣To")]
        [SerializeField]
        private VoidEventChannelSO _inMenuEvent;
        [SerializeField]
        private VoidEventChannelSO _outMenuEvent;
        [SerializeField]
        private RoomObjectParentChannelSO _customerWithObjectFixedEvent;

        [Header("Setting␣Func")]
        [SerializeField]
        private ToolRecipeManagerFuncSO _getLevelToolRecipeManagerSOFunc;

        private void Awake()
        {
            _getLevelToolRecipeManagerSOFunc.ClearOnFuncRaised();
            _getLevelToolRecipeManagerSOFunc.TrySetOnFuncRaised(() => _levelToolRecipeManagerSO);

            _waitingToStartTimer = _waitingToStartTimerMax;
            _gameplayTimer = _gameplayTimerMax;
        }

        private void OnEnable()
        {
            _inMenuEvent.OnEventRaised += ToMenuMode;
            _outMenuEvent.OnEventRaised += ToGameplayMode;

            _customerWithObjectFixedEvent.OnEventRaised += ObjectFixedAndReturned;
        }

        private void OnDisable()
        {
            _inMenuEvent.OnEventRaised -= ToMenuMode;
            _outMenuEvent.OnEventRaised -= ToGameplayMode;

            _customerWithObjectFixedEvent.OnEventRaised += ObjectFixedAndReturned;
        }

        private void Start()
        {
            //_gameState = GameState.WaitingToStart;
            _inputReaderSO.DisableAllInput();
        }

        public override void OnNetworkSpawn()
        {
            _gameState.OnValueChanged += State_OnValueChanged;

            if (IsServer) {
                NetworkManager.Singleton.SceneManager.OnLoadEventCompleted += NM_SM_OnLoadEventCompleted;
            }
        }

        private void State_OnValueChanged(GameState previousValue, GameState newValue)
        {
            switch (newValue)
            {
                case GameState.WaitingToStart:
                    _inputReaderSO.DisableAllInput();
                    break;
                case GameState.Playing:
                    _inputReaderSO.EnableGameplayInput();
                    break;
                case GameState.End:
                    _inputReaderSO.EnableMenuInput();
                    _numberObjectsFixedEvent.RaiseEvent(_numberObjectsFixed.Value);
                    break;
                default:
                    break;
            }
        }

        private void NM_SM_OnLoadEventCompleted(string sceneName, LoadSceneMode loadSceneMode, List<ulong> clientsComp
        {
            for (int i = 0; i < NetworkManager.Singleton.ConnectedClientsIds.Count; i++) {
                ulong clientId = NetworkManager.Singleton.ConnectedClientsIds[i];

                GameObject playerGO = Instantiate(_playerPrefab, _baseTransformToSpawn);
                playerGO.transform.position = _playerSpawnPositions[i].position;
                playerGO.GetComponent<NetworkObject>().SpawnAsPlayerObject(clientId, true);
                Debug.Log(playerGO.GetComponent<NetworkObject>().OwnerClientId);
                Debug.Log(NetworkManager.Singleton.LocalClientId);
            }
        }

        private void Update()
        {
            if (!IsServer) {
                return;
            }

            switch (_gameState.Value) {
                case GameState.WaitingToStart:
                    // esperar countdown to start
                    _waitingToStartTimer -= Time.deltaTime;
                    if (_waitingToStartTimer < 0f) {
                        _gameState.Value = GameState.Playing;
                        //_inputReaderSO.EnableGameplayInput();
                    }

                    WaitingToStartRaiseEventClientRpc(_waitingToStartTimer);
                    //_waitingToStartTimerEvent.RaiseEvent(_waitingToStartTimer);
                    break;
                case GameState.Playing:
                    // timer juego
                    _gameplayTimer -= Time.deltaTime;
                    if (_gameplayTimer < 0f) {
                        _gameState.Value = GameState.End;
                        //_inputReaderSO.EnableMenuInput();

                        //_numberObjectsFixedEvent.RaiseEvent(_numberObjectsFixed);
                    }

                    // timer npcs
                    _customerSpawnerTimer -= Time.deltaTime;
                    if (_customerSpawnerTimer < 0f) {
                        _customerSpawnerTimer = _customerSpawnerTimerMax;

                        SpawnNewCustomer();
                    }

                    float gameplayTimerNormalized = GetTimerNormalized(_gameplayTimer, _gameplayTimerMax);
                    GameplayTimerNormalizedRaiseEventClientRpc(gameplayTimerNormalized);
                    //_gameplayTimerNormalizedEvent.RaiseEvent(GetTimerNormalized(_gameplayTimer, _gameplayTim
                    break;
                case GameState.End:
                    // mostrar puntuacion
                    Debug.Log(_numberObjectsFixed);
                    break;
                default:
                    Debug.LogWarning($"{_gameState}␣is␣not␣implemented");
                    break;
            }


        }

        [ClientRpc]
        private void WaitingToStartRaiseEventClientRpc(float waitingToStartTimer)
        {
            _waitingToStartTimerEvent.RaiseEvent(waitingToStartTimer);
        }

        [ClientRpc]
        private void GameplayTimerNormalizedRaiseEventClientRpc(float gameplayTimerNormalized)
        {
            _gameplayTimerNormalizedEvent.RaiseEvent(gameplayTimerNormalized);
        }

        private float GetTimerNormalized(float timer, float timerMax)
        {
            return timer / timerMax;
        }

        #region GameplayMode
        private void ToMenuMode()
        {
            _inputReaderSO.EnableMenuInput();
        }

        private void ToGameplayMode()
        {
            _inputReaderSO.EnableGameplayInput();
        }
        #endregion

        #region Game Loop

        #region CustomerCounters
        private CustomerCounter GetFirstCustomerCounterFree()
        {
            foreach (CustomerCounter counter in _customerCounters) {
                if (!counter.HasCustomerAssigned()) {
                    return counter;
                }
            }

            return null;
        }

        private int GetCustomerCounterIndex(CustomerCounter customerCounter)
        {
            return System.Array.IndexOf(_customerCounters, customerCounter);
        }

        private CustomerCounter GetCustomerCounterFromIndex(int index)
        {
            return _customerCounters[index];
        }
        #endregion

        #region ObjectToFixSO
        private RoomObjectSO GetRandomObjecToFixSO()
        {
            int randIndex = Random.Range(0, _objectsToFixSO.Length);

            return _objectsToFixSO[randIndex];
        }

        private int GetObjectToFixSOIndex(RoomObjectSO roomObjectSO)
        {
            return System.Array.IndexOf(_objectsToFixSO, roomObjectSO);
        }

        private RoomObjectSO GetObjectToFixSOFromIndex(int index)
        {
            return _objectsToFixSO[index];
        }
        #endregion

        private void SpawnNewCustomer()
        {
            CustomerCounter freeCounter = GetFirstCustomerCounterFree();
            if (freeCounter == null) {
                return;
            }
            int freeCounterIndex = GetCustomerCounterIndex(freeCounter);

            RoomObjectSO objectToFixSO = GetRandomObjecToFixSO();
            int objectToFixSOIndex = GetObjectToFixSOIndex(objectToFixSO);

            SpawnNewCustomerServerRpc(freeCounterIndex, objectToFixSOIndex);
        }

        [ServerRpc]
        private void SpawnNewCustomerServerRpc(int freeCounterIndex, int objectToFixSOIndex)
        {
            GameObject customerGO = Instantiate(_customerPrefab, _customerStartPosition.position, Quaternion.i
            customerGO.transform.position = _customerStartPosition.position;
            customerGO.transform.rotation = _customerStartPosition.rotation;


            CustomerController customerController = customerGO.GetComponent<CustomerController>();

            if (customerController == null) {
                Debug.LogError($"The␣prefab␣{_customerPrefab}␣is␣not␣a␣Customer␣Controller");
                return;
            }

            NetworkObject customerNO = customerGO.GetComponent<NetworkObject>();
            customerNO.Spawn();
            customerGO.transform.parent = RoomObject.StaticInSceneTransform;

            CustomerCounter freeCounter = GetCustomerCounterFromIndex(freeCounterIndex);
            RoomObjectSO objectToFixSO = GetObjectToFixSOFromIndex(objectToFixSOIndex);
            customerController.InitCustomer(_customerStartPosition, freeCounter, objectToFixSO);
            freeCounter.SetCustomerAssigned(customerController);


            RoomObject.SpawnRoomObject(objectToFixSO, customerController);
        }

        private void ObjectFixedAndReturned(IRoomObjectParent customerWithObject)
        {
            _numberObjectsFixed.Value++;
        }
        #endregion
    }
}
```

B.11 RoomObject NOT FINAL VERSION

```
using FixingIt.ActorComponents;
using FixingIt.Events;
using FixingIt.RoomObjects.SO;
using ProgramadorCastellano.Events;
using Unity.Netcode;
using UnityEngine;

namespace FixingIt.RoomObjects.Logic
{
    [RequireComponent(typeof(FollowTransformComponent))]
    public class RoomObject : NetworkBehaviour
    {
        private static TryToSpawnRoomObjectChannelSO _staticTryToSpawnRoomObjectEvent;
        public static Transform StaticInSceneTransform { get; private set; }

        [SerializeField] private RoomObjectSO _roomObjectSO;
        [SerializeField] private int _numberOfUses = 1;

        [Header("Components")]
        [SerializeField] private AudioComponent _roomObjectUsedAudioComp;
        [SerializeField] private AudioComponent _roomObjectBrokenAudioComp;
        private FollowTransformComponent _followTransformComp;

        [Header("Broadcasting␣To")]
        //[SerializeField]
        //private VoidEventChannelSO _roomObjectUsedEvent;
        [SerializeField]
        private VoidEventChannelSO _roomObjectBrokenAfterUseEvent;
        [SerializeField]
        private TryToSpawnRoomObjectChannelSO _tryToSpawnRoomObjectEvent;   // its value should be the same among all

        private IRoomObjectParent _roomObjectParent;

        public RoomObjectSO RoomObjectSO => _roomObjectSO;
        public GameObject RoomObjectVisualPrefab => transform.GetChild(0).gameObject;

        protected virtual void Awake()
        {
            _followTransformComp = GetComponent<FollowTransformComponent>();
        }

        private void Start()
        {
            if (_tryToSpawnRoomObjectEvent != null) {
                _staticTryToSpawnRoomObjectEvent = _tryToSpawnRoomObjectEvent;
                StaticInSceneTransform = transform;
            }
        }

        public void SetRoomObjectParent(IRoomObjectParent newRoomObjectParent)
        {
            SetRoomObjectParentServerRpc(newRoomObjectParent.GetNetworkObject());
        }

        [ServerRpc(RequireOwnership = false)]
        private void SetRoomObjectParentServerRpc(NetworkObjectReference newRoomObjectParentNORef)
        {
            transform.parent = StaticInSceneTransform;
            SetRoomObjectParentClientRpc(newRoomObjectParentNORef);
        }

        [ClientRpc]
        private void SetRoomObjectParentClientRpc(NetworkObjectReference newRoomObjectParentNORef)
        {
            newRoomObjectParentNORef.TryGet(out NetworkObject newRoomObjectParentNO);
            IRoomObjectParent newRoomObjectParent = newRoomObjectParentNO.GetComponent<IRoomObjectParent>();

            // clear parent info
            _roomObjectParent?.ClearRoomObject();

            // set new parent
            _roomObjectParent = newRoomObjectParent;

            if (newRoomObjectParent.HasRoomObject())
            {
                Debug.LogError("IRoomObjectParent␣already␣has␣a␣RoomObject");
            }

            newRoomObjectParent.SetRoomObject(this);

            // set transform
            _followTransformComp.SetTargetTransform(newRoomObjectParent.GetRoomObjectTransform());
        }

        public void Use()
        {
            _numberOfUses--;

            if (_numberOfUses <= 0)
            {
                _roomObjectBrokenAfterUseEvent.RaiseEvent();

                //if (_roomObjectBrokenAudioComp == null) {
                //    Debug.LogWarning("Only Tools should be broken");
                //}
                //else {
                //    _roomObjectBrokenAudioComp.PlaySound();
                //}
                Broke();
            }
            else
            {
                if (_roomObjectUsedAudioComp == null)
                {
                    Debug.LogWarning("Only␣Tools␣should␣be␣used");
                }
                else
                {
                    _roomObjectUsedAudioComp.PlaySound(false);
                }
            }
        }

        // TODO: cambiar por sistema de pooling como extra
        public void Broke()
        {
            _roomObjectParent.ClearRoomObject();

            Destroy(gameObject);
        }

        #region Static
        // TODO: cambiar por sistema de pooling como extra
        public static void SpawnRoomObject(RoomObjectSO roomObjectSO, IRoomObjectParent roomObjectParent)
        {
            _staticTryToSpawnRoomObjectEvent.RaiseEvent(roomObjectSO, roomObjectParent.GetNetworkObject());
        }
        #endregion
    }
}
```

B.12 IRoomObjectParent

```
using Unity.Netcode;
using UnityEngine;

namespace FixingIt.RoomObjects.Logic
{
    public interface IRoomObjectParent
    {
        public Transform transform { get; }

        public Transform GetRoomObjectTransform();
        public RoomObject GetRoomObject();
        public void SetRoomObject(RoomObject roomObject);
        public bool HasRoomObject();
        public void ClearRoomObject();

        public NetworkObject GetNetworkObject();
    }
}
```

B.13 ToFixRoomObject

```
using FixingIt.RoomObjects.SO;
using System.Linq;
using UnityEngine;

```

```
namespace FixingIt.RoomObjects.Logic
{
    public class ToFixRoomObject : RoomObject
    {
        [SerializeField] private ToFixRoomObjectVisualComp _toFixRoomObjectVisualComp;
        [SerializeField] private RoomObjectSO[] _toolsToBeFixedSO;
        private bool[] _toolsUsedSO;

        public bool IsFixed => _toolsUsedSO.All(valor => valor);

        protected override void Awake()
        {
            base.Awake();
            _toolsUsedSO = new bool[_toolsToBeFixedSO.Length];
        }

        private void Start()
        {
            _toFixRoomObjectVisualComp.UpdateTFROVisual(_toolsToBeFixedSO, _toolsUsedSO);
        }

        //private void FixObject()
        //{
        //    // change visual?
        //    _toFixRoomObjectVisualComp.UpdateTFROVisual(_toolsToBeFixedSO, _toolsUsedSO);
        //    Debug.Log("FixObject");
        //}

        public bool TryToFix(RoomObjectSO toolUsed, out bool toolBeenUsed)
        {
            toolBeenUsed = false;

            // should be check from outside but just in case
            if (IsFixed)
            {
                Debug.Log("cannot␣fix␣a␣object␣that␣is␣already␣fixed");
                return false;
            }

            for (int i = 0; i < _toolsToBeFixedSO.Length; i++)
            {
                if (_toolsUsedSO[i])
                    continue;

                if (toolUsed == _toolsToBeFixedSO[i])
                {
                    _toolsUsedSO[i] = true;
                    toolBeenUsed = true;
                    break;
                }
            }

            _toFixRoomObjectVisualComp.UpdateTFROVisual(_toolsToBeFixedSO, _toolsUsedSO);

            if (!IsFixed)
                return false;

            return true;
        }
    }
}
```

B.14 PlayerController

```
using FixingIt.ActorComponents;
using FixingIt.Counters;
using FixingIt.InputSystem;
using FixingIt.RoomObjects.Logic;
using ProgramadorCastellano.Funcs;
using Unity.Netcode;
using UnityEngine;

namespace FixingIt.PlayerGame
{
    [RequireComponent(typeof(Rigidbody))]
    public class PlayerController : NetworkBehaviour, IRoomObjectParent
    {
        [SerializeField] InputReaderSO _inputReaderSO;

        [SerializeField] private Transform _holdingPoint;

        [Header("Player␣Comps")]
        [SerializeField] private PlayerVisualComp _playerVisualComp;
        [SerializeField] private PlayerAnimationComp _animationComp;
        [SerializeField] private AudioComponent _audioComp;

        [Header("Player␣Stats")]
        [SerializeField] private float _moveSpeed = 5f;
        [SerializeField] private float _rotateSpeed = 10f;
        [SerializeField] private float _interactDistance = 2f;
        [SerializeField] private LayerMask _countersLayerMask;

        [Header("Invoking␣Func")]
        [SerializeField]
        private ULongColorFuncSO _getColorFromClientIdFunc;

        private Rigidbody _rb;
        private RoomObject _roomObject;
        private Vector2 _direction;

        private Outline _currentOutline;

        private void Awake()
        {
            _rb = GetComponent<Rigidbody>();
        }

        private void OnEnable()
        {
            // se maneja dentro de las funciones para evitar una condicion de carrera
            //if (OwnerClientId == NetworkManager.LocalClientId)
            //    Debug.Log("ofnesoinfie");
            //if (!IsOwner) {
            //    return;
            //}

            _inputReaderSO.MoveEvent += SetPlayerDirection;
            _inputReaderSO.InteractEvent += HandleInteraction;
            _inputReaderSO.AlternateInteractEvent += HandleAlternateInteraction;
        }

        private void OnDisable()
        {
            // si se deja ocurre una condicion de carrera que bloqueaba el personaje
            //if (!IsOwner) {
            //    return;
            //}

            _inputReaderSO.MoveEvent -= SetPlayerDirection;
            _inputReaderSO.InteractEvent -= HandleInteraction;
            _inputReaderSO.AlternateInteractEvent -= HandleAlternateInteraction;
        }

        private void Start()
        {
            _playerVisualComp.SetPlayerColor(_getColorFromClientIdFunc.RaiseFunc(OwnerClientId));
        }

        private void FixedUpdate()
        {
            if (!IsOwner) {
                return;
            }

            HandleMovement();
        }

        private void Update()
        {
            if (!IsOwner) {
                return;
            }

            HandleRotation();
            HandleSelectionOutline();
        }

        private void HandleSelectionOutline()
        {
            Vector3 rayOrigin = transform.position;
            Vector3 rayDir = transform.forward;

            if (Physics.Raycast(rayOrigin, rayDir, out RaycastHit rayHit, _interactDistance, _countersLayerMask)) {
                if (rayHit.transform.TryGetComponent(out Outline outline)) {
                    if (_currentOutline == outline) {
                        return;
                    }

                    if (_currentOutline != null) {
                        _currentOutline.enabled = false;
                    }

                    _currentOutline = outline;
                    _currentOutline.enabled = true;
                }
                else {
                    if (_currentOutline == null) {
                        return;
                    }

                    _currentOutline.enabled = false;
                    _currentOutline = null;
                }
            }
            else {
                if (_currentOutline == null) {
                    return;
                }

                _currentOutline.enabled = false;
                _currentOutline = null;
            }
        }

        private void HandleMovement()
        {
            Vector3 velocity = new Vector3(_direction.x, 0f, _direction.y);
            velocity *= _moveSpeed;

            _rb.velocity = velocity;

            bool isMoving = velocity != Vector3.zero;
            _animationComp.SetIsWalking(isMoving);

            if (isMoving) {
                _audioComp.PlaySound();
            }
            else {
                _audioComp.StopSound();
            }
        }

        private void HandleRotation()
        {
            Vector3 desiredRotation = new Vector3(_direction.x, 0f, _direction.y);

            transform.forward = Vector3.Slerp(transform.forward, desiredRotation, Time.deltaTime * _rotateSpee
        }

        public PlayerVisualComp GetPlayerVisualComp()
        {
            return _playerVisualComp;
        }

        #region InputActions
        private void SetPlayerDirection(Vector2 directionInput)
        {
            if (!IsOwner) {
                return;
            }

            _direction = directionInput;
        }

        private void HandleInteraction()
        {
            if (!IsOwner) {
                return;
            }

            Vector3 rayOrigin = transform.position;
            Vector3 rayDir = transform.forward;

            if (Physics.Raycast(rayOrigin, rayDir, out RaycastHit rayHit, _interactDistance, _countersLayerMas
                if (rayHit.transform.TryGetComponent(out BaseCounter baseCounter)) {
                    baseCounter.Interact(this);
                }

                Debug.DrawRay(rayHit.point, rayDir * _interactDistance, Color.green, 5f);
            }

            Debug.DrawRay(rayOrigin, rayDir * _interactDistance, Color.red, 5f);
        }

        private void HandleAlternateInteraction()
        {
            if (!IsOwner) {
                return;
            }

            Vector3 rayOrigin = transform.position;
            Vector3 rayDir = transform.forward;

            if (Physics.Raycast(rayOrigin, rayDir, out RaycastHit rayHit, _interactDistance, _countersLayerMask)) {
                if (rayHit.transform.TryGetComponent(out BaseCounter baseCounter)) {
                    baseCounter.AlternateInteract(this);
                }

                Debug.DrawRay(rayHit.point, rayDir * _interactDistance, Color.green, 5f);
            }

            Debug.DrawRay(rayOrigin, rayDir * _interactDistance, Color.blue, 5f);
        }
        #endregion

        #region IRoomObjectParent
        public Transform GetRoomObjectTransform()
        {
            return _holdingPoint;
        }

        public RoomObject GetRoomObject()
        {
            return _roomObject;
        }

        public void SetRoomObject(RoomObject roomObject)
        {
            _roomObject = roomObject;
        }

        public bool HasRoomObject()
        {
            return _roomObject != null;
        }

        public void ClearRoomObject()
        {
            _roomObject = null;
        }

        public NetworkObject GetNetworkObject()
        {
            return NetworkObject;
        }
        #endregion
    }
}
```

B.15 BaseCounter

```
using FixingIt.RoomObjects.Logic;
using Unity.Netcode;
using UnityEngine;

namespace FixingIt.Counters
{
    public abstract class BaseCounter : NetworkBehaviour, IRoomObjectParent
    {
        [SerializeField] private Transform _topPoint;

        private RoomObject _roomObject;

        public abstract void Interact(IRoomObjectParent roomObjectParent);
        public abstract void AlternateInteract(IRoomObjectParent roomObjectParent);

        public Transform GetRoomObjectTransform()
        {
            return _topPoint;
        }

        public RoomObject GetRoomObject()
        {
            return _roomObject;
        }

        public void SetRoomObject(RoomObject roomObject)
        {
            _roomObject = roomObject;
        }

        public bool HasRoomObject()
        {
            return _roomObject != null;
        }

        public void ClearRoomObject()
        {
            _roomObject = null;
        }

        public NetworkObject GetNetworkObject()
        {
            return NetworkObject;
        }
    }
}
```

B.16 CustomerController

```
using FixingIt.ActorComponents;
using FixingIt.RoomObjects.Logic;
using FixingIt.RoomObjects.SO;
using Unity.Netcode;
using UnityEngine;
using UnityEngine.AI;

namespace FixingIt.Customer
{
    public class CustomerController : NetworkBehaviour, IRoomObjectParent
    {
        // could have be done with a state machine but this case is too simple
        private enum ClientState
        {
            Waiting,
            GoingToCounter,
            LeavingCounter
        }

        [SerializeField] Transform _holdingPoint;

        private NavMeshAgent _agent;
        private ClientState _clientState;

        private RoomObject _roomObject;

        private Transform _startTransform;
        private IRoomObjectParent _parentToLeaveBrokenObject;

        [Header("Customer Comps")]
        [SerializeField]
        private AudioComponent _audioComp;

        private void Awake()
        {
            _agent = GetComponent<NavMeshAgent>();

            _clientState = ClientState.Waiting;
        }

        private void Start()
        {
            if (!IsServer)
                return;

            GoToCounter();
        }

        private void Update()
        {
            if (!IsServer)
                return;

            switch (_clientState)
            {
                case ClientState.Waiting:
                    break;
                case ClientState.GoingToCounter:
                    _audioComp.PlaySound();
                    if (_agent.remainingDistance < _agent.stoppingDistance)
                    {
                        _clientState = ClientState.Waiting;
                        _roomObject.SetRoomObjectParent(_parentToLeaveBrokenObject);

                        _audioComp.StopSound();
                    }
                    break;
                case ClientState.LeavingCounter:
                    _audioComp.PlaySound();
                    if (_agent.remainingDistance < _agent.stoppingDistance)
                    {
                        // si da tiempo cambiarlo por un sistema de pooling
                        Destroy(gameObject);
                    }
                    break;
                default:
                    Debug.LogWarning($"{_clientState}␣is␣not␣implemented");
                    break;
            }
        }

        private void GoToCounter()
        {
            Debug.Log(_parentToLeaveBrokenObject != null);
            _agent.SetDestination(_parentToLeaveBrokenObject.transform.position);
            _clientState = ClientState.GoingToCounter;
        }

        public void LeaveCounter()
        {
            _agent.SetDestination(_startTransform.position);
            _clientState = ClientState.LeavingCounter;
        }

        public void InitCustomer(Transform startTransform, IRoomObjectParent parentToLeaveBrokenObject, RoomObjectSO _
        {
            _startTransform = startTransform;
            _parentToLeaveBrokenObject = parentToLeaveBrokenObject;

            transform.position = _startTransform.position;
            transform.rotation = _startTransform.rotation;

            //RoomObject.SpawnRoomObject(_objectToFixSO, this);
        }

        #region RoomObjectParent
        public Transform GetRoomObjectTransform()
        {
            return _holdingPoint;
        }

        public RoomObject GetRoomObject()
        {
            return _roomObject;
        }

        public void SetRoomObject(RoomObject roomObject)
        {
            _roomObject = roomObject;
        }

        public bool HasRoomObject()
        {
            return _roomObject != null;
        }

        public void ClearRoomObject()
        {
            _roomObject = null;
        }

        public NetworkObject GetNetworkObject()
        {
            return NetworkObject;
        }
        #endregion
    }
}
```

B.17 InputReaderSO

```
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.InputSystem;

namespace FixingIt.InputSystem
{
//[CreateAssetMenu(fileName = "Input Reader", menuName = "Game/ Input Reader")]
    public class InputReaderSO : ScriptableObject, GameInput.IGameplayActions, GameInput.IMenuActions
    {
        private GameInput _gameInput;

        // Assign delegate{} to events to initialise them with an empty delegate
        // so we can skip the null check when we use them

        // Gameplay
        public event UnityAction<Vector2> MoveEvent = delegate { };
        public event UnityAction InteractEvent = delegate { };
        public event UnityAction AlternateInteractEvent = delegate { };

        // Menu
        public event UnityAction MenuConfirmEvent = delegate { };
        public event UnityAction MenuCancelEvent = delegate { };
        public event UnityAction<Vector2> MenuNavigationEvent = delegate { };

        /*
         * On Enable/Disable Functions
         */
        #region On Enable/Disable
        private void OnEnable()
        {
            if (_gameInput == null) {
                _gameInput = new GameInput();

                // set all callbacks
                _gameInput.Gameplay.SetCallbacks(this);
                _gameInput.Menu.SetCallbacks(this);
            }

            // EnableGameplayInput(); // TODO: se debe manejar de forma externa
        }

        private void OnDisable()
        {
            DisableAllInput();
        }
        #endregion

        /*
```

```
         * Turn On/Off Inputs
         */
        #region Turn On/Off Inputs
        public void EnableGameplayInput()
        {
            DisableAllInput();

            _gameInput.Gameplay.Enable();
        }

        public void EnableMenuInput()
        {
            DisableAllInput();

            _gameInput.Menu.Enable();
        }

        public void DisableAllInput()
        {
            _gameInput.Gameplay.Disable();
            _gameInput.Menu.Disable();
        }
        #endregion

        /*
         *  Gameplay Acions
         */
        #region Gameplay Actions
        public void OnMove(InputAction.CallbackContext context)
        {
            MoveEvent.Invoke(context.ReadValue<Vector2>());
        }

        public void OnInteract(InputAction.CallbackContext context)
        {
            if (context.phase == InputActionPhase.Performed) {
                InteractEvent.Invoke();
            }
        }

        public void OnAlternateInteract(InputAction.CallbackContext context)
        {
            if (context.phase == InputActionPhase.Performed) {
                AlternateInteractEvent.Invoke();
            }
        }
        #endregion

        /*
         *  Menu Actions
         */
        #region Menu Actions
        public void OnSubmit(InputAction.CallbackContext context)
        {
            if (context.phase == InputActionPhase.Performed) {
                MenuConfirmEvent.Invoke();
            }
        }

        public void OnCancel(InputAction.CallbackContext context)
        {
            if (context.phase == InputActionPhase.Performed) {
                MenuCancelEvent.Invoke();
            }
        }

        public void OnNavigation(InputAction.CallbackContext context)
        {
            if (context.phase == InputActionPhase.Performed) {
                MenuNavigationEvent.Invoke(context.ReadValue<Vector2>());
            }
        }
        #endregion
    }
}
```

B.18 SelectableUIData

```
using UnityEngine;
using UnityEngine.UI;

namespace ProgramadorCastellano.UI
{
    [RequireComponent(typeof(Selectable))]
    public class SelectableUIData : MonoBehaviour
    {
        [SerializeField] private SelectableUISkinDataSO _skinDataSO;

        private Selectable _selectable;

        private void Awake()
        {
            //_selectable = GetComponent<Selectable>();

            OnSkinUI();
        }

        private void OnSkinUI()
        {
            _selectable = GetComponent<Selectable>();
            //Debug.Log(_selectable.gameObject.name);

            _selectable.colors = _skinDataSO.Colors;
        }


        // COMMENT THE UPDATE AFTER DESGIN
        /*private void Update()
        {
            if (Application.isEditor)
            {
                OnSkinUI();
            }
        }*/
    }
}
```