**Fun Run Clone**

**(SPRC Project)**

Coordinator, Student,

Prof. Cristian Niculita Luca Sergiu

YEAR IV CTI

GRUPA 22C32 A

AN 2023-2024

Content

[Introduction 3](#_Toc168517172)

[What is a game engine? 3](#_Toc168517173)

[The most popular game engines: 3](#_Toc168517174)

[Why did I choose Godot 4? 4](#_Toc168517175)

[The game manual 5](#_Toc168517176)

[Launching the Server 5](#_Toc168517177)

[Launching the Application 6](#_Toc168517178)

[Level playing 8](#_Toc168517179)

[Game resources 10](#_Toc168517180)

[Technologies used 12](#_Toc168517181)

[View the source code 13](#_Toc168517182)

[C# components used 15](#_Toc168517183)

[FinishLine.cs 15](#_Toc168517184)

[GameManager.cs 16](#_Toc168517185)

[MultiplayerController.cs 16](#_Toc168517186)

[Player.cs 17](#_Toc168517187)

[PlayerInfo.cs 18](#_Toc168517188)

[PowerUp.cs 18](#_Toc168517189)

[SceneManager.cs 19](#_Toc168517190)

[System architecture and structure 20](#_Toc168517191)

[Bibliography, resources and bookstores 21](#_Toc168517192)

# Introduction

## What is a game engine?

A game engine is like a virtual toolbox for game developers. Instead of building everything from scratch, a game engine provides pre-built components and essential functionality, such as:

* **Graphic rendering:** Display images and animations on the screen.
* **Physics:** Simulating the movement and interactions of objects.
* **Sound:** Sound effects and music management.
* **Input:** Reading commands from the player (keyboard, mouse, controller).
* **Scripting:** Ability to add custom logic and behavior to the game.

Using a game engine speeds up development and allows creators to focus on the unique aspects of their game.

## The most popular game engines:

There are a variety of game engines available, each with their own strengths and specializations. Among the most popular are:

* **Unity:** Versatile and easy to learn, great for 2D and 3D games of all genres.
* **Unreal Engine:** Known for high quality graphics, ideal for AAA games and realistic simulations.
* **Godot:** Open-source and free, with an active and growing community, suitable for independent and experimental projects.
* **GameMaker Studio 2:** Focused on 2D games with an easy-to-use visual scripting language.

## Why did I choose Godot 4?

Godot 4 was chosen for this "fun run clone" project for several reasons:

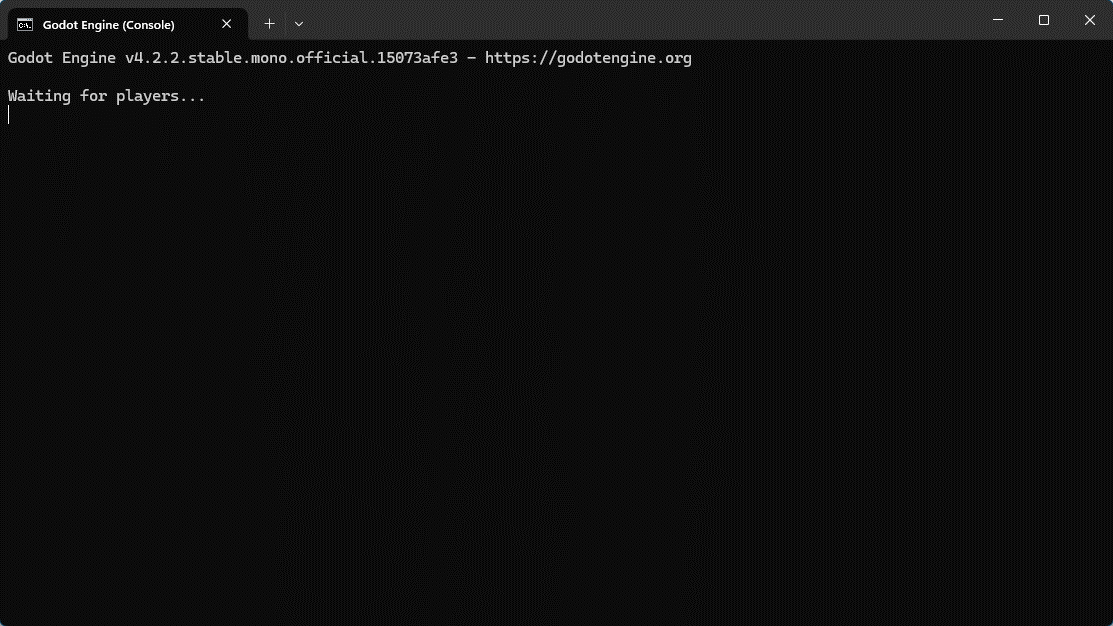
* **Open-source and free:** It removes financial barriers and allows total flexibility in development.
* **Easy to use:** The intuitive interface and scene system simplify the game creation process.
* **Performance:** The powerful and optimized 2D engine can easily handle the required graphics and physics.
* **Community:** An active community and numerous online resources provide support and inspiration.
* **Portability:** Games created in Godot can be exported to a wide range of platforms (PC, mobile, web).

# The game manual

In Fun Run Clone, all you have to do is run from left to right collecting power-ups and jump over obstacles trying to reach the finish line first to win. Seems simple, doesn't it?

Success!

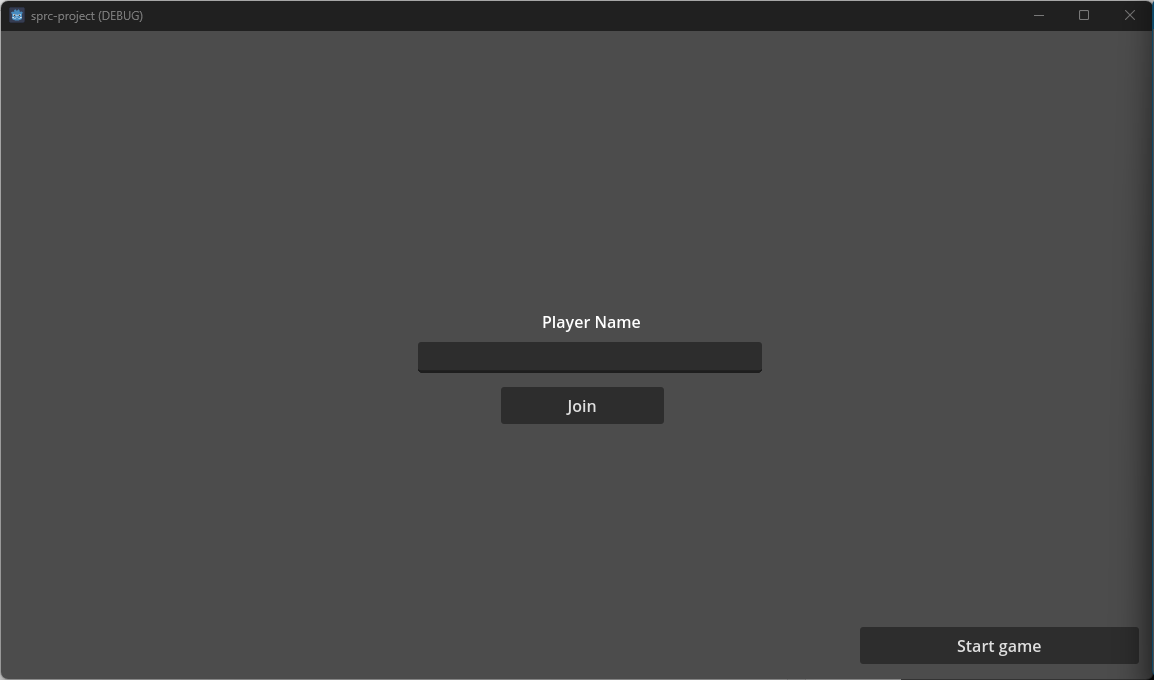
## Launching the Server



The server can be started using the Start\_Server.bat file in the server subfolder. This will run the executable for the dedicated server along with other commands to ensure that only the server console is displayed and not the game interface.

Once the server is launched, players can connect to the dedicated server and play.

## Launching the Application



Once the game is launched, the player is greeted by the "Home" screen which contains a field for entering the name, the Join button (to connect to the server) and the "START GAME" button.

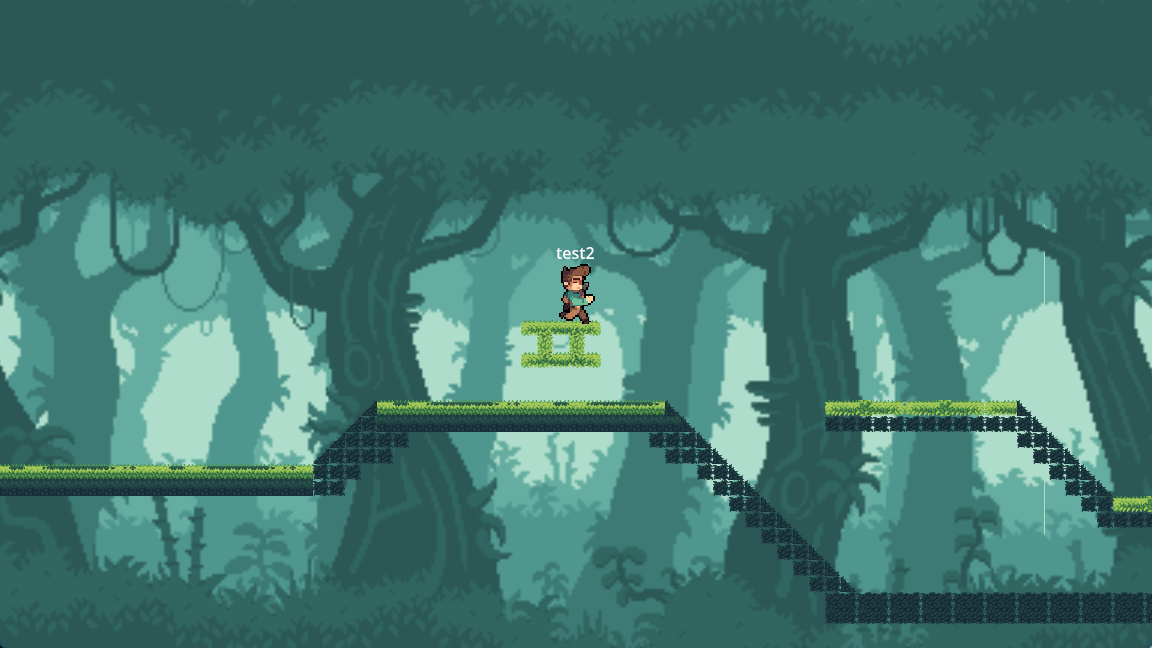
Once the "START GAME" button is pressed, the game will wait until another player connects to the server to start the game.  


Once the second player has connected the game will start instantly.

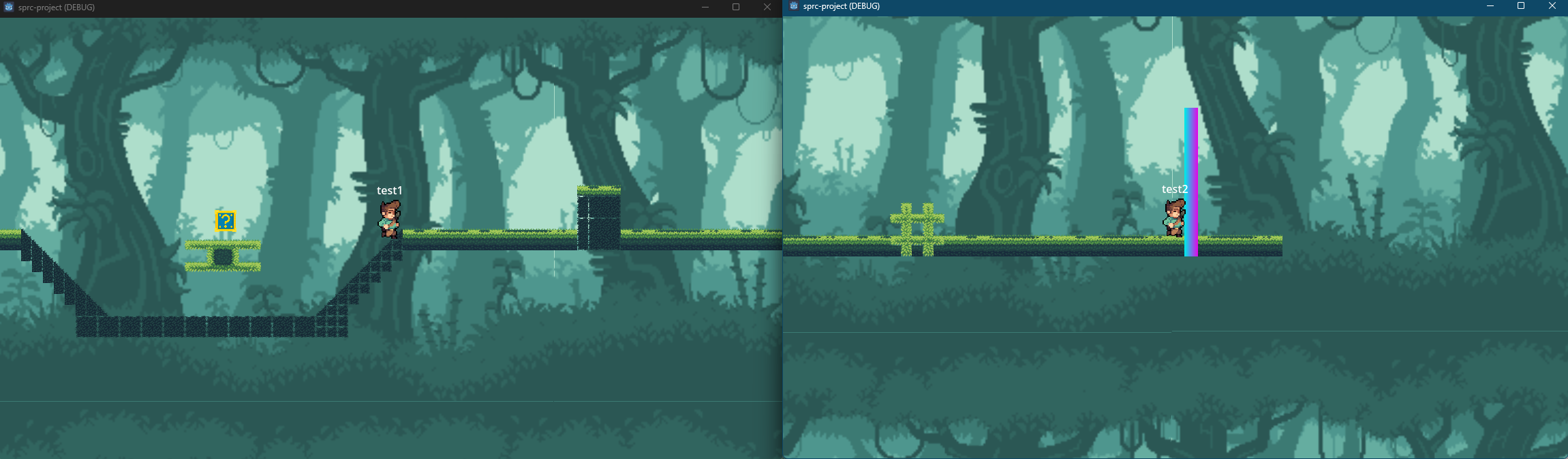
## Level playing



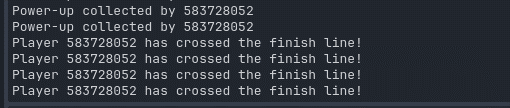
Each level can have a different type of platform and placement of power-ups.



Once a player passes over a power-up it will disappear and will no longer be available.



When a player reaches the end of the map, at the finish line it will alert the server that all players should be stopped because one has won.

The winner is announced in the console through a log:  


# Game resources

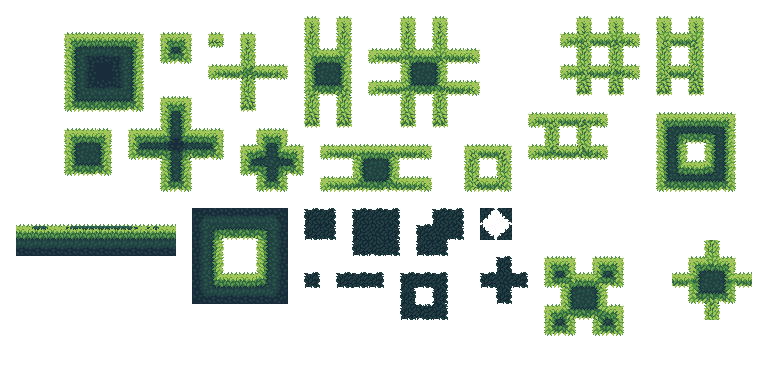
For the game resources I used an indie game dev kit purchased from itch.io

From this website I downloaded and used:

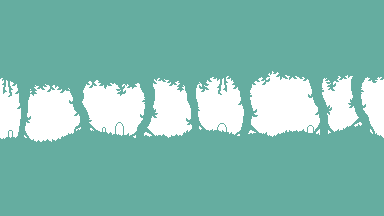
* Character + his animations:

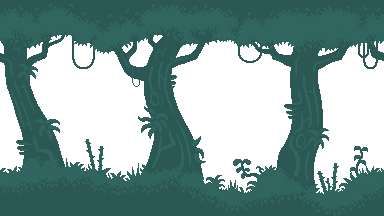


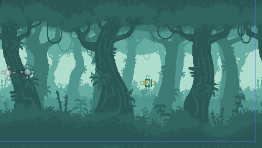
* Tile set for the map:



* Background with parallax:







# Technologies used

**Git:** Git is a distributed version control system (VCS) essential in modern software development. It allows tracking of changes to the source code, collaboration between developers and rollback in case of errors. In our project, Git was used to efficiently manage and organize the game's source code, facilitating collaboration between team members and ensuring a clear development history.

**GitHub:** GitHub is a hosting platform for software projects that use Git. In addition to Git's core functionality, GitHub also offers additional tools for collaboration, such as issue tracking, code review, and project management. In our project, GitHub served as the central source code repository, allowing easy access by all team members and facilitating remote collaboration.

**C# (C Sharp):** C# is a modern, object-oriented programming language developed by Microsoft. It is known for its clear syntax, versatility and high performance. In our project, C# was chosen as the main programming language due to its native support in Godot 4 and its advantages in game development, such as easy object management, automatic garbage collection, and access to powerful libraries.

**Godot 4:** Godot 4 is a free and open-source game engine that offers a wide range of tools and functionality for 2D and 3D game development. It stands out for its intuitive interface, flexible scene system and optimized performance. In our project, Godot 4 was chosen as the main development platform due to its ease of use, active community and C# support, allowing us to create an engaging and interactive "fun run clone" game.

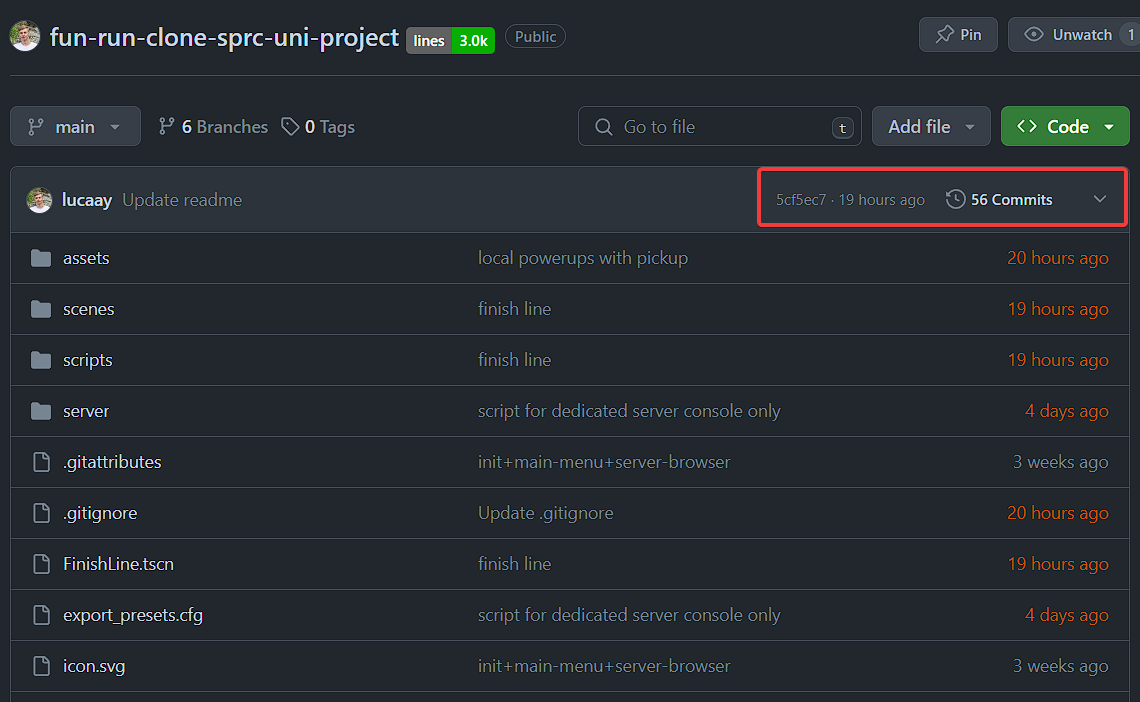
# View the source code

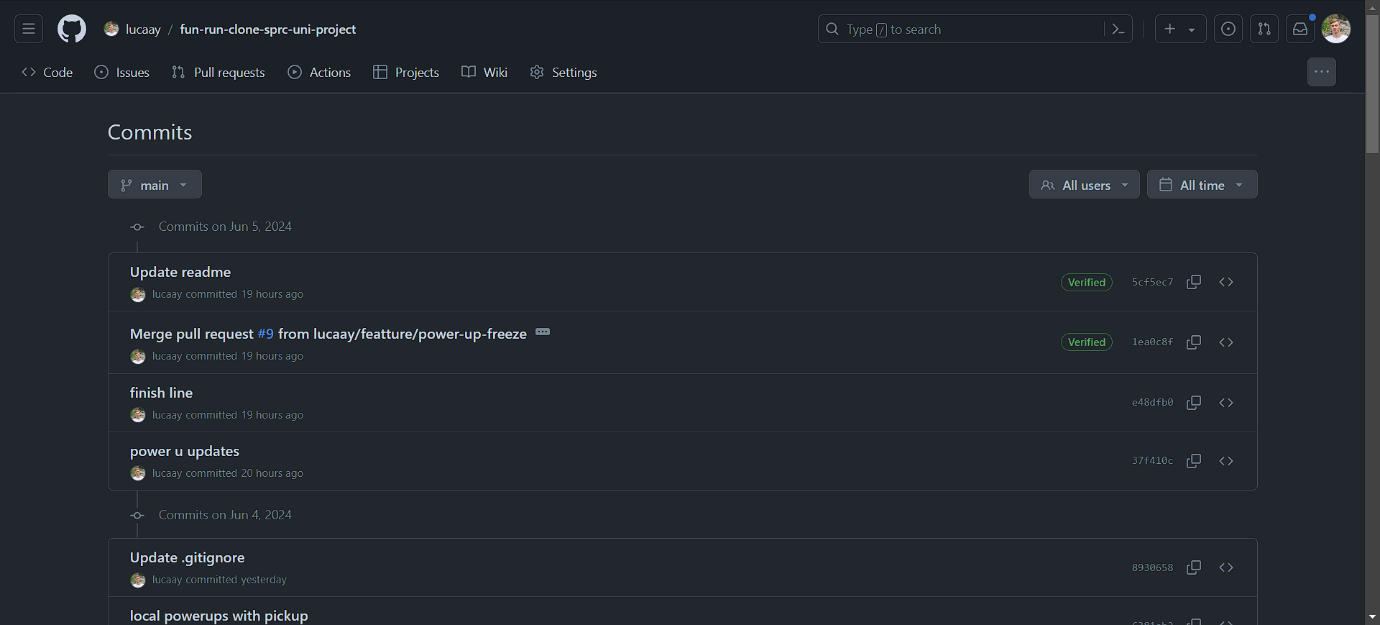
The source code can be viewed on the following link:

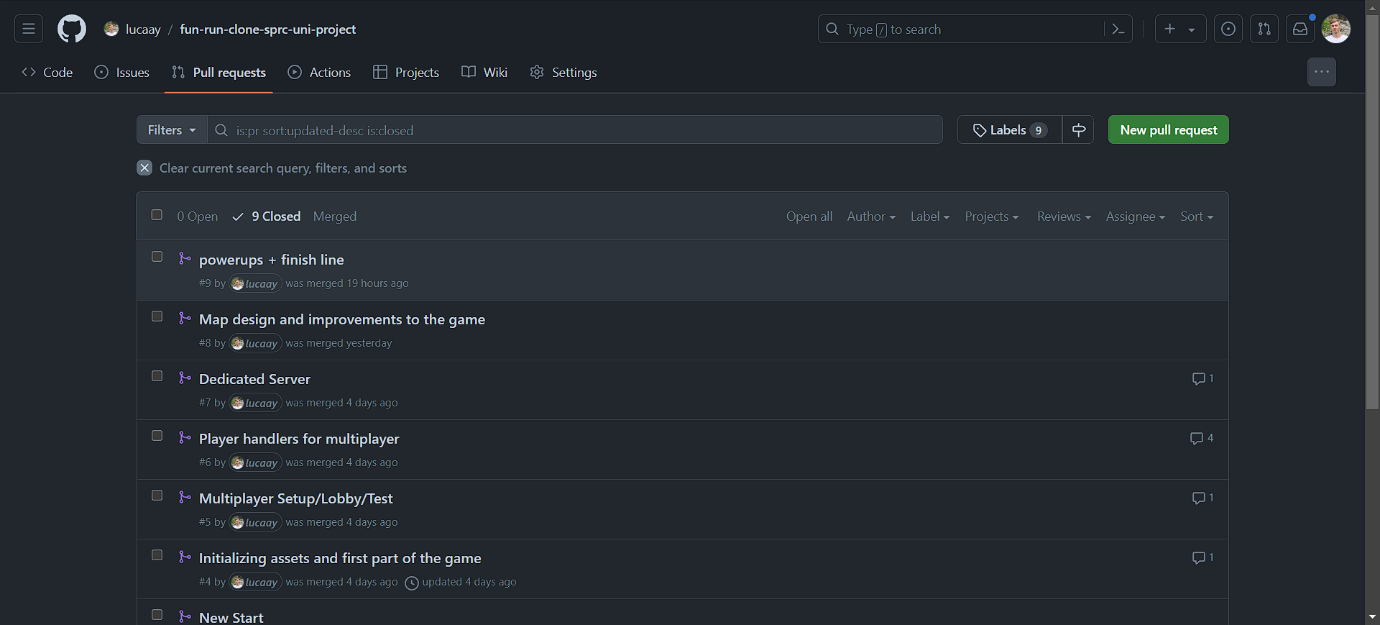
<https://github.com/lucaay/fun-run-clone-sprc-uni-project>

The following can be viewed here:

* All game code
* Several commits showing the changes made throughout the development of the game
* The code divided into Pull Requests representing the major Stages in the development of the game.
* A message for each commit explaining the changes made







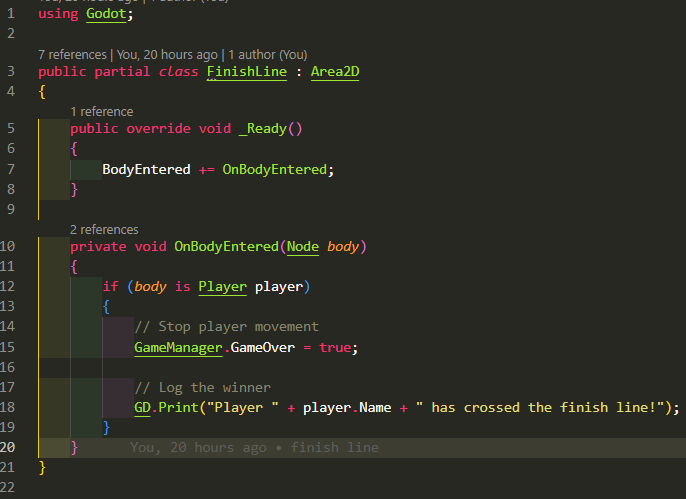
# C# components used

## FinishLine.cs

**Purpose:** Detects when a player hits the finish line and triggers game over.

**Operation:**

* Waits for an object to enter its collision area.
* Checks if the object is a player.
* If so, it signals the end of the game

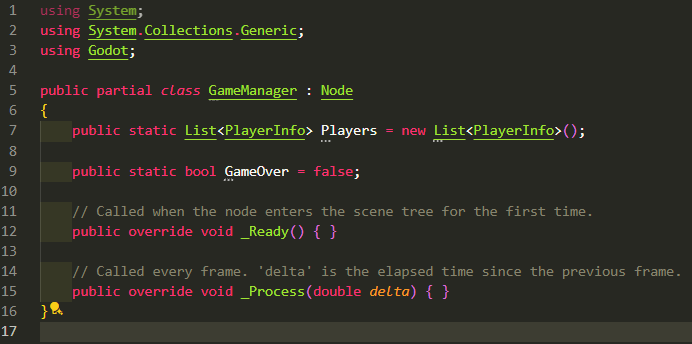


## GameManager.cs

**Purpose:** Manages the overall state of the game.

**Operation:**

* Keeps a list of connected players (Players).
* Holds a boolean variable to indicate whether the game is over (GameOver).



## MultiplayerController.cs

**Purpose:** Manages connection and communication between players in a multiplayer environment.

**Operation:**

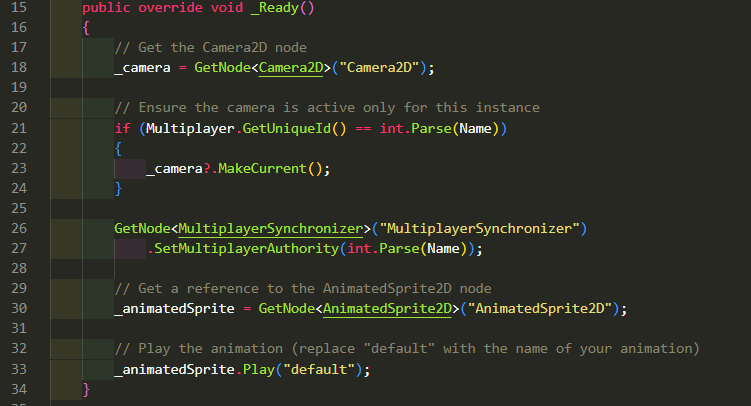
* Allows connection to an existing dedicated server.
* Send and receive information between players such as name, position, actions, etc.
* Handles events such as login, logout or game start.

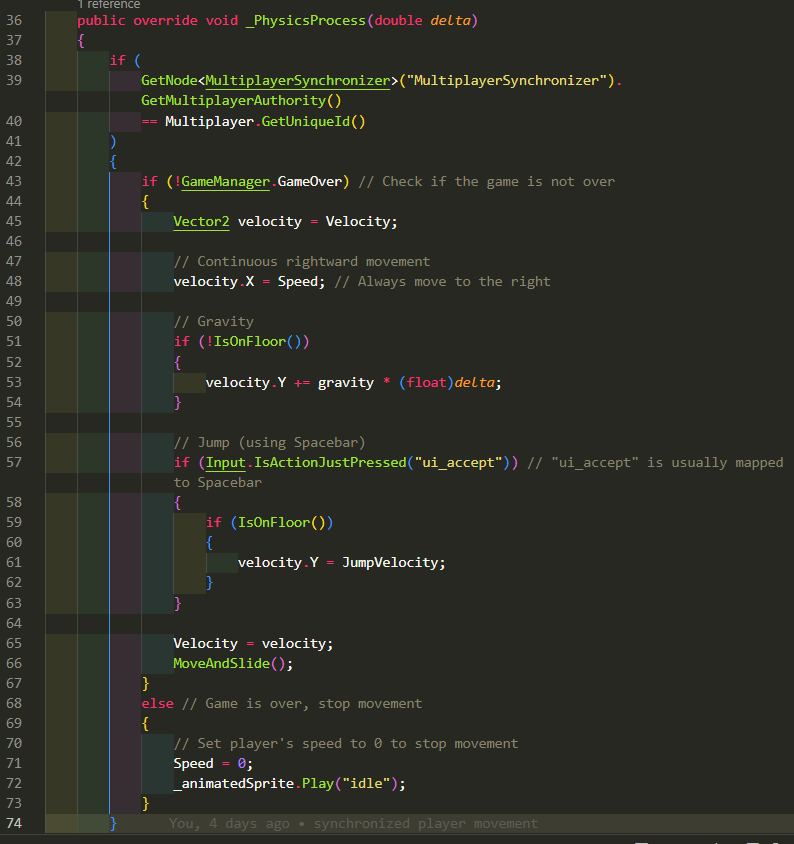
## Player.cs

**Purpose:** Controls a player's movement and actions.

**Operation:**

* Ensures continuous movement of the player to the right.
* Allows the player to jump.
* Handles collisions with the ground and other objects.
* Synchronizes player movement and actions in multiplayer.





## PlayerInfo.cs

**Purpose:** Stores information about a player.

**Operation:**

* Contains properties for player name and id.
* It is used to pass player information between the server and clients.

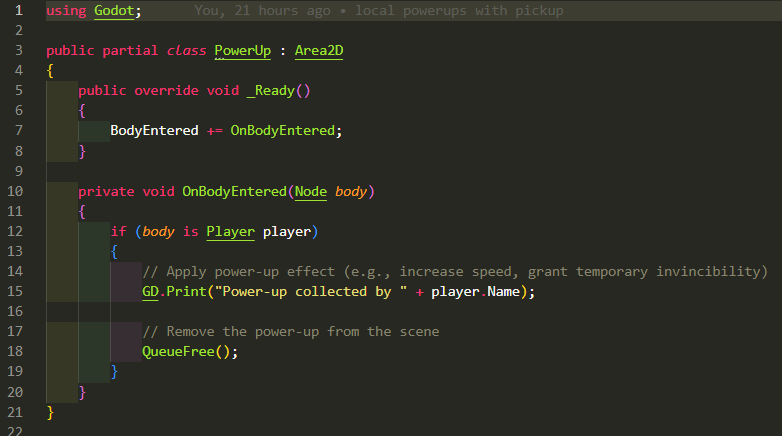


## PowerUp.cs

**Purpose:** Defines the behavior of a power-up.

**Operation:**

* Waits for an object to enter its collision area.
* Checks if the object is a player.
* If so, it applies the power-up's effect to the player and self-destructs.

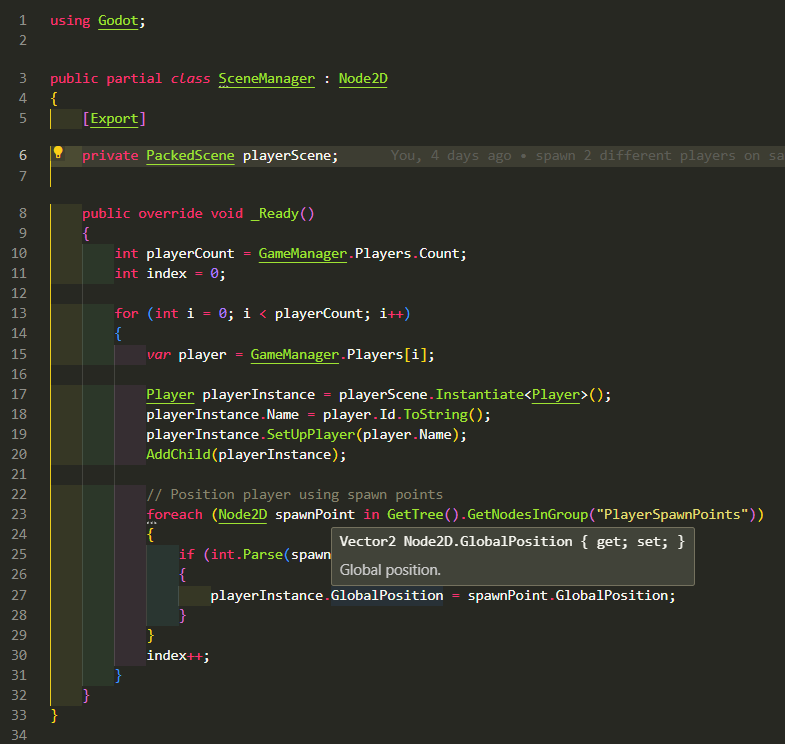


## SceneManager.cs

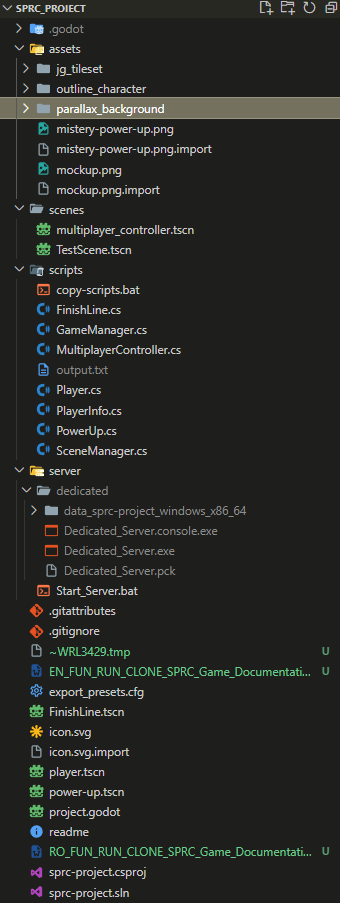
**Purpose:** Manages the loading and placement of elements in the scene, especially players.

**Operation:**

* Creates player instances for each connected player.
* Position players on designated spawn points.



# System architecture and structure



# Bibliography, resources and bookstores

1. <https://godotengine.org/>
2. <https://itch.io/game-assets>
3. GO: <https://code.visualstudio.com/>
4. GIT Version Control: <https://github.com/lucaay/>
5. GIT Client: <https://desktop.github.com/>