Target Advanced Game

Sommario

1 Introduzione

1.1 Informazioni sul progetto

- Allievi: Simone Riva, Michael Dobeson, Stefano Ceschi
- Classe: I3AC-BC
- Docente responsabile: Michel Palucci
- Data inizio: 27.01.2023
- Data fine: 5.05.2023

1.2 Abstract

T.A.G. is a virtual recreation of the Laser Tag game, that is set in the future with a futuristic environment and characters.
In Laser Tag everyone is against everyone and you have to point your laser towards the enemy to disable his laser and earn a point.
In our game there will be a map with randomly generated obstacles where the player has to hit other enemies/other players with their laser pointer. You can also defend yourself using a shield to stop the laser from touching you.
The player is also able to see his scores in real time thanks to a leaderboard in the center of the map, he can also see the top statistics through his registered account on a website with a Database.

1.3 Scopo

Lo scopo del progetto è di avere una variazione virtuale del gioco laser tag con una generazione casuale di oggetti all'interno del campo di gioco per stimolare l'utente ad inventarsi una strategia di gioco diversa ad ogni partita.

2 Analisi

2.1 Analisi del dominio

Il gioco dovrebbe essere il più semplice possibile per gli utenti, andando a riprendere le caratteristiche principali del gioco reale. La particolarità del nostro gioco è la virtualizzazione del tutto tramite videogioco. Essendo uguale all'originale, gli utenti impareranno velocemente come si gioca.

2.2 Analisi e specifica dei requisiti

Priorità 2/3 = opzionale

ID: REQ-01	
Nome	Movimento Giocatore
Priorità	1
Versione	1.0
Note	L'utente può usare dei comandi di tastiera o joystick per muovere il personaggio.
Sotto requisiti	

ID: REQ-02	
Nome	Interfaccia Grafica
Priorità	1
Versione	1.0
Note	L'utente può navigare attraverso l'interfaccia di gioco usando tastiera o joystick.
Sotto requisiti	
001	Interfaccia UI per gestire tutte le impostazioni (es: audio, grafica, account, ...).

ID: REQ-03	
Nome	Ambiente Partita
Priorità	1
Versione	1.0
Note	Deve essere presente una mappa di gioco.
Sotto requisiti	

ID: REQ-04	
Nome	Sito Web
Priorità	1
Versione	1.0
Note	Deve essere presente un sito web.
Sotto requisiti	
001	**L'utente crea un account**

ID: REQ-05	
Nome	Interazioni Giocatore
Priorità	1
Versione	1.0
Note	L'utente può mirare e puntare un laser ed eventualmente parare i laser degli altri usando i comandi di tastiera o joystick.
Sotto requisiti	

ID: REQ-06	
Nome	Ostacoli Casuali
Priorità	1
Versione	1.0
Note	I muri si generano in maniera casuale all'inizio di ogni partita usando l'algoritmo di Manhattan Mapper.
Sotto requisiti	

ID: REQ-07	
Nome	HUD
Priorità	1
Versione	1.0
Note	L'Heads-Up Display (HUD) mostrerà varie informazioni durante la partita.
Sotto requisiti	
001	Tempo rimasto di partita.
002	Punteggio

ID: REQ-08	
Nome	AI Nemico
Priorità	1
Versione	1.0
Note	Un'AI che si comporta come un giocatore.
Sotto requisiti	

ID: REQ-09	
Nome	Database
Priorità	1
Versione	1.0
Note	Il database conterrà tutte le tabelle utili per lo storage di dati e per il multiplayer.
Sotto requisiti	
001	Tabella Leaderboard Globale: sarà una tabella consultabile dal sito web dove verranno mostrati i migliori giocatori di sempre con il punteggio.
002	Tabella Leaderboard della Partita: conterrà i punteggi e i giocatori della partita corrente.

ID: REQ-10	
Nome	Multiplayer
Priorità	1
Versione	1.0
Note	Il giocatore può collegarsi ad una sessione con altri giocatori.
Sotto requisiti	

ID: REQ-11	
Nome	Algoritmo di Cell-Shading
Priorità	2
Versione	1.0
Note	La grafica del gioco è da cartone animata però in 3D.
Sotto requisiti	

ID: REQ-12	
Nome	Video Clip
Priorità	3
Versione	1.0
Note	Le partite vengono registrate e mandati al sito web per essere visualizzati.
Sotto requisiti	

In accordo con il committente durante la fase di sviluppo abbiamo deciso di eliminare il sito web e dare comunque la possibilità all'utente di creare l'account su Unity

ID: REQ-13	
Nome	Creazione Account
Priorità	2
Versione	1.0
Note	L'utente crea un account
Sotto requisiti	

2.3 Use case

I casi d'uso rappresentano l'interazione tra i vari attori e le funzionalità del prodot

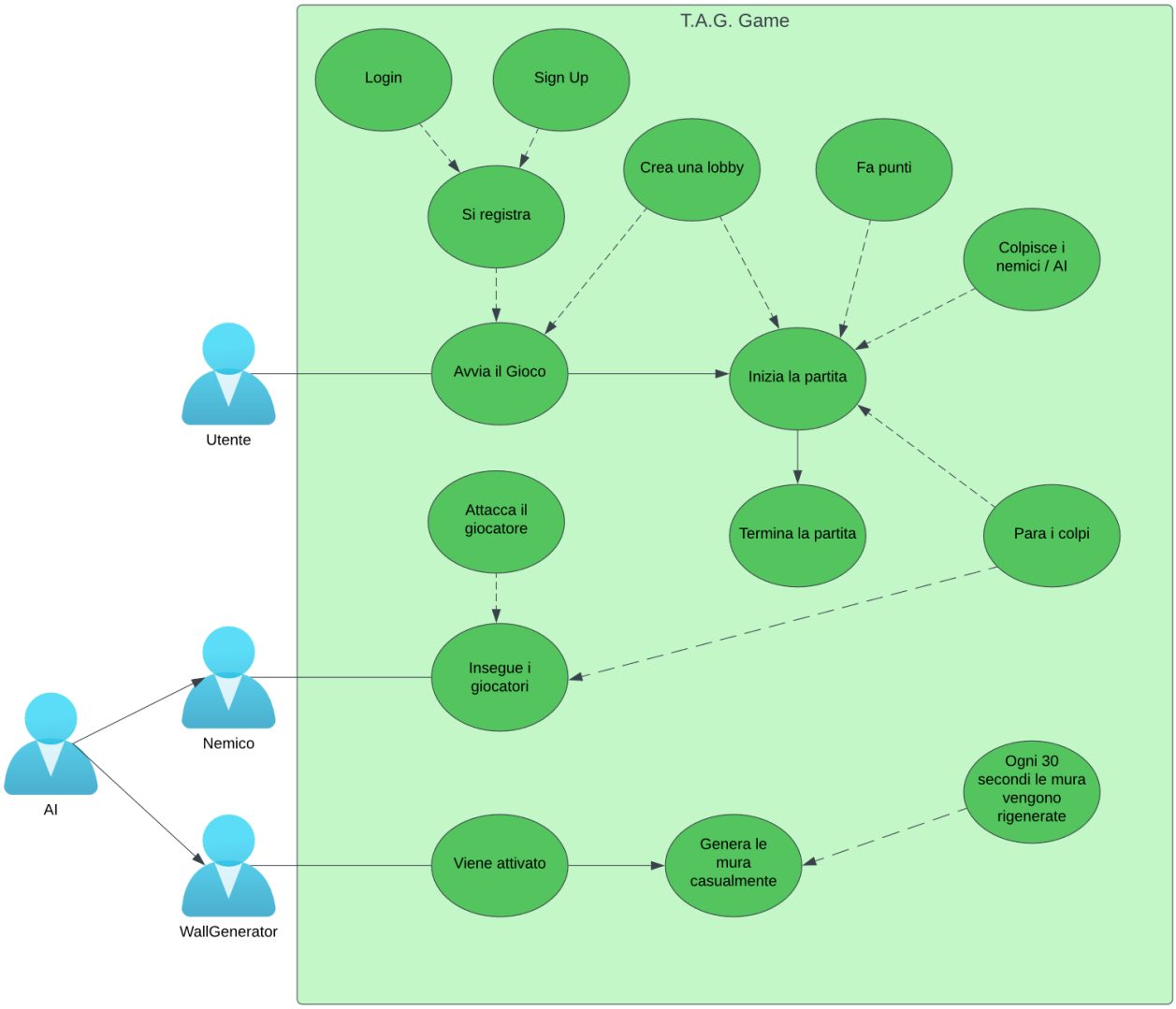
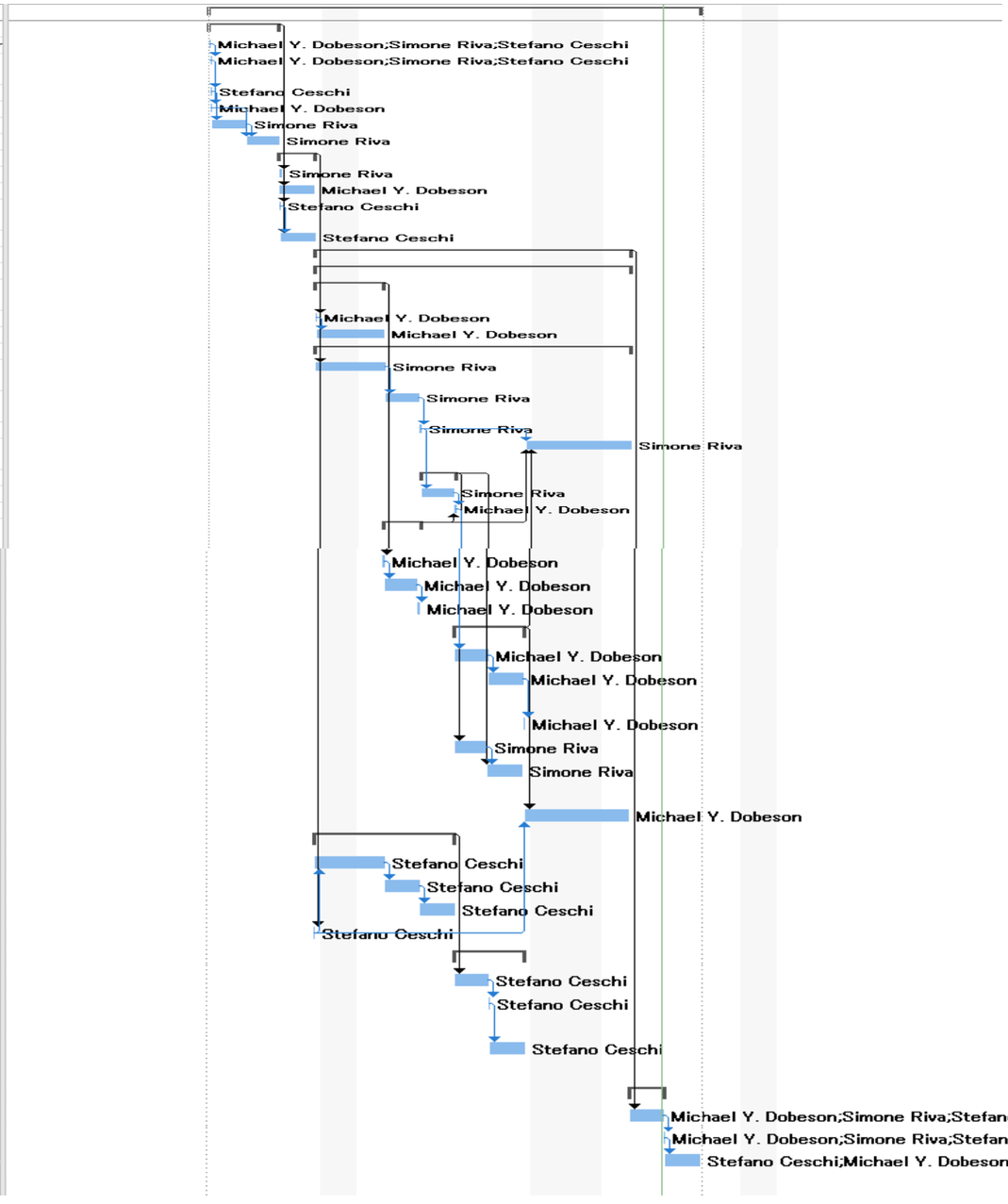


Figura 1 - Use case

2.4 Pianificazione

	T.A.G.	**169.5 h**	**11.75 giorni?**	**ven 27.01.23**	**ven 05.05.23**		
	Analisi	**18 h**	**2 giorni?**	**ven 27.01.23**	**ven 10.02.23**		
	Stesura QdC	1.5 h	0.08 giorni?	ven 27.01.23	ven 27.01.23		Michael Y. Dobeson;Sir
	Requisiti (Documentazione)	3 h	0.17 giorni?	ven 27.01.23	ven 27.01.23	3	Michael Y. Dobeson; Simone Riva;Stefano
	Use Case	3 h	0.5 giorni?	ven 27.01.23	ven 27.01.23	4	Stefano Ceschi
	Gantt Preventivo	3 h	0.5 giorni?	ven 27.01.23	ven 27.01.23	4	Michael Y. Dobeson
	Test Case	6 h	1 giorno?	ven 27.01.23	ven 03.02.23	5	Simone Riva
	Abstract	1.5 h	0.25 giorni?	ven 03.02.23	ven 10.02.23	6;7	Simone Riva
	Progettazione	**16.5 h**	**1.25 giorni?**	**ven 10.02.23**	**ven 17.02.23**		
	Design Interfacce	3 h	0.5 giorni?	ven 10.02.23	ven 10.02.23	2	Simone Riva
	Diagramma di Flusso	6 h	1 giorno?	ven 10.02.23	ven 17.02.23	2	Michael Y. Dobeson
	Diagramma E/R (Database)	1.5 h	0.25 giorni?	ven 10.02.23	ven 10.02.23	2	Stefano Ceschi
	Diagramma di Classi	6 h	1 giorno?	ven 10.02.23	ven 17.02.23	2;12	Stefano Ceschi
	Implementazione	**99 h**	**6.5 giorni?**	**ven 17.02.23**	**ven 21.04.23**		
	Unity	**84 h**	**6.5 giorni?**	**ven 17.02.23**	**ven 21.04.23**		
	Movimeto Giocatore	**4.5 h**	**0.75 giorni?**	**ven 17.02.23**	**ven 03.03.23**		
	Spostamento	1.5 h	0.25 giorni?	ven 17.02.23	ven 17.02.23	9	Michael Y. Dobeson
	Visuale	3 h	0.5 giorni?	ven 17.02.23	ven 03.03.23	17	Michael Y. Dobeson
	Interfaccia Grafica	**19.5 h**	**6.5 giorni?**	**ven 17.02.23**	**ven 21.04.23**		
	Scene di Interfacce	6 h	1 giorno?	ven 17.02.23	ven 03.03.23	9	Simone Riva
	Navigazione con Joystick	4.5 h	0.75 giorni?	ven 03.03.23	ven 10.03.23	20	Simone Riva
	HUD	3 h	0.5 giorni?	ven 10.03.23	ven 10.03.23	21	Simone Riva
	Funzionamento Impostazioni	6 h	1 giorno?	ven 31.03.23	ven 21.04.23	27;31;22	Simone Riva
	Ambiente Partita	**6 h**	**1 giorno?**	**ven 10.03.23**	**ven 17.03.23**		
	Arena	3 h	0.5 giorni?	ven 10.03.23	ven 17.03.23	22	Simone Riva
	Ostacoli Casuali	3 h	0.5 giorni?	ven 17.03.23	ven 17.03.23	25;27	Michael Y. Dobeson
	Interazioni Giocatore	**10.5 h**	**1.75 giorni?**	**ven 03.03.23**	**ven 10.03.23**		
	Puntatore Laser	4.5 h	0.75 giorni?	ven 03.03.23	ven 03.03.23	16	Michael Y. Dobeson
	Mirare Laser	1.5 h	0.25 giorni?	ven 03.03.23	ven 10.03.23	28	Michael Y. Dobeson
	Scudo Anti-Laser	4.5 h	0.75 giorni?	ven 10.03.23	ven 10.03.23	29	Michael Y. Dobeson
	AI	**13.5 h**	**2.25 giorni?**	**ven 17.03.23**	**ven 31.03.23**		
	Algoritmo	4.5 h	0.75 giorni?	ven 17.03.23	ven 24.03.23	26	Michael Y. Dobeson
	Implementazione in Unity	6 h	1 giorno?	ven 24.03.23	ven 31.03.23	32	Michael Y. Dobeson
	Difficoltà	3 h	0.5 giorni?	ven 31.03.23	ven 31.03.23	33	Michael Y. Dobeson
	Audio	3 h	0.5 giorni?	ven 17.03.23	ven 24.03.23	24	Simone Riva
	Modelli ed Animazioni	6 h	1 giorno?	ven 24.03.23	ven 31.03.23	24;35	Simone Riva
	Unity Leaderboard	3 h	0.5 giorni?	ven 31.03.23	ven 21.04.23	42;31	Michael Y. Dobeson
	Multiplayer	**18 h**	**3 giorni?**	**ven 17.02.23**	**ven 17.03.23**		
	Ricerca	6 h	1 giorno?	ven 17.02.23	ven 03.03.23	42	Stefano Ceschi
	Prove	6 h	1 giorno?	ven 03.03.23	ven 10.03.23	39	Stefano Ceschi
	Implementazione	6 h	1 giorno?	ven 10.03.23	ven 17.03.23	40	Stefano Ceschi
	Database	3 h	0.5 giorni?	ven 17.02.23	ven 17.02.23	9	Stefano Ceschi
	Sito Web	**12 h**	**2 giorni?**	**ven 17.03.23**	**ven 31.03.23**		
	Struttura	3 h	0.5 giorni?	ven 17.03.23	ven 24.03.23	38	Stefano Ceschi
	Funzionamento JavaScript	3 h	0.5 giorni?	ven 24.03.23	ven 24.03.23	44	Stefano Ceschi
	Funzionamento PHP	6 h	1 giorno?	ven 24.03.23	ven 31.03.23	45	Stefano Ceschi
	Conclusione	**18 h**	**1 giorno?**	**ven 21.04.23**	**ven 28.04.23**		
	Test Case	9 h	0.5 giorni?	ven 21.04.23	ven 28.04.23	14	Michael Y. Dobeson;Sir
	Risolvere Problemi	9 h	0.5 giorni?	ven 28.04.23	ven 28.04.23	48	Michael Y. Dobeson;Sir
	Documentazione	18 h	1 giorno?	ven 28.04.23	ven 05.05.23	49	Stefano Ceschi;Michae



2.5 Analisi dei mezzi

PC scolastici:

- **CPU:** *Intel Core I7 7th generation*
- **GPU:** *Intel HD Graphics 730 (abbiamo avuto problemi con questa scheda video)*
- **RAM:** *16GB 3.60GHz*
- **DISCO:** *Western Digital 500GB (HDD)*

Mezzi esterni:

- 1x SSD SanDisk 500GB
- 1x SSD SanDisk 1TB
- 1x SSD Samsung 256GB

2.5.1 Software

Sviluppo gioco

- Unity 2022.1.f1 (*personal license*)
- Netcode for GameObjects
- Visual Studio 2022 + 2019
- Blender

Sviluppo WEB

- PHPStorm (*school license*)
- Visual Studio Code
- MySQL

Ambiente di test

- XAMPP

2.5.2 Hardware

Il prodotto è disegnato per poter funzionare fluentemente sulla maggior parte dei computer in circolazione, è però consigliato utilizzare hardware performante per evitare sbalzi di *framerate* oppure di prestazioni di rete. Per poter usufruire del nostro prodotto è **consigliato** possedere una connessione ad internet stabile per evitare il *lag*.

Durante lo sviluppo potremmo utilizzare i PC scolastici descritti nel capitolo 2.5.

3 Progettazione

3.1 Design dell'architettura del sistema

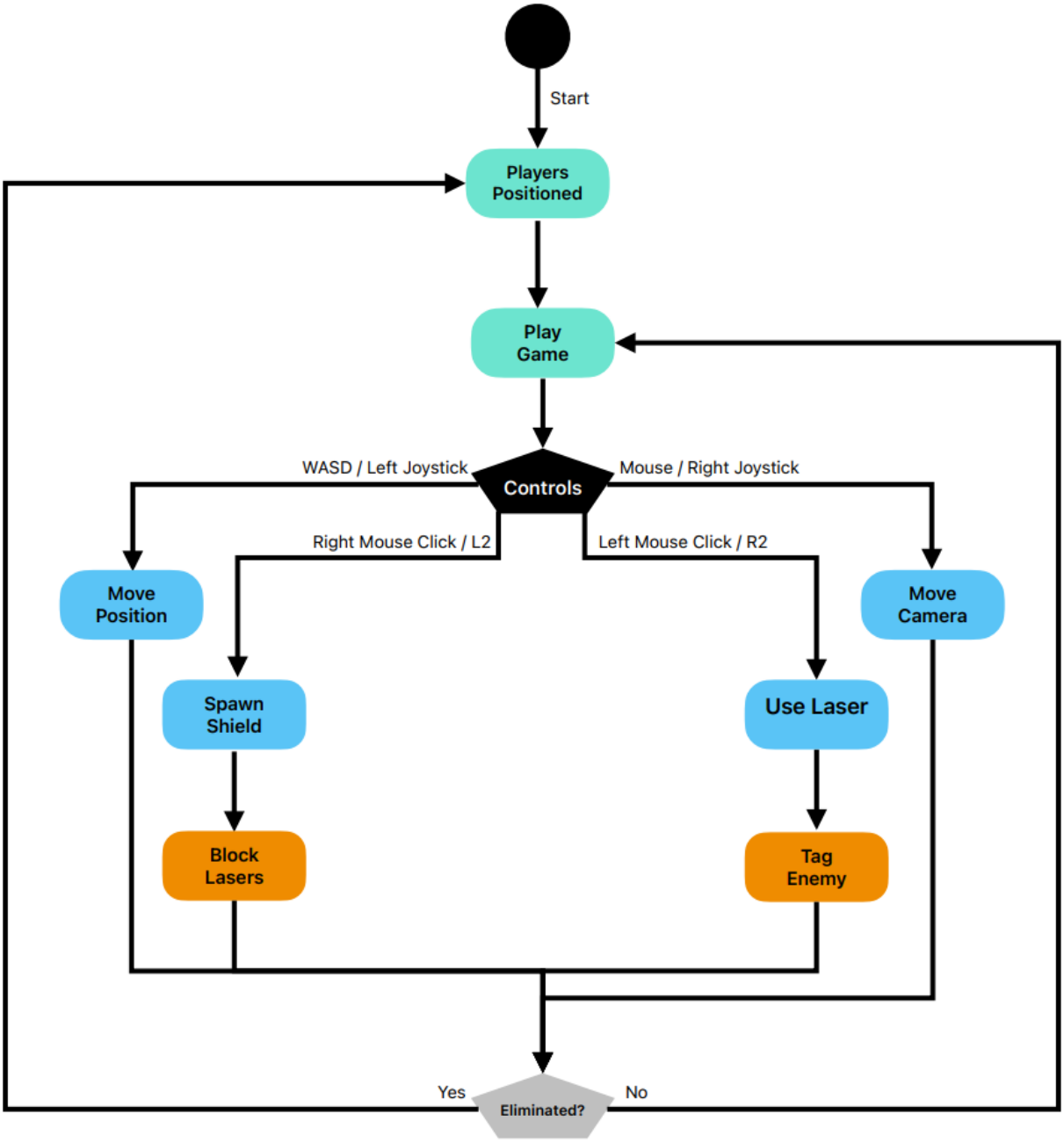


Figura 2 - Diagramma Flusso (Game Mechanics)

3.2 Design dei dati e database

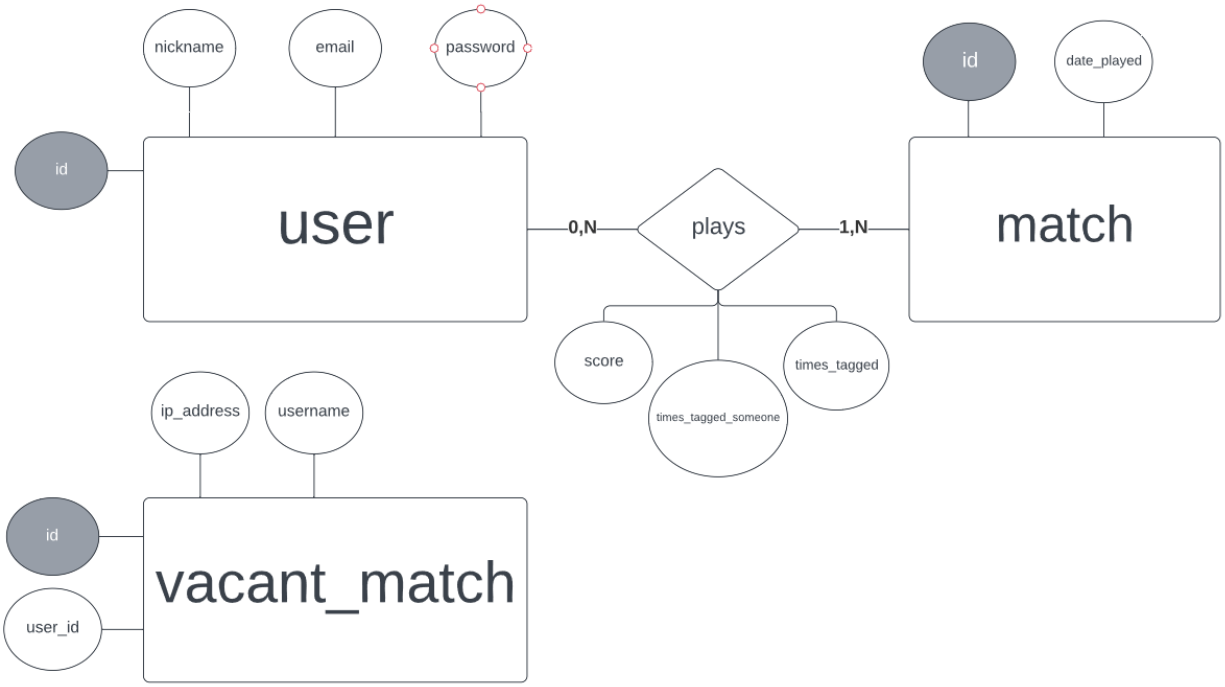


Figura 3 – Diagramma E/R

3.3 Design procedurale

3.3.1 Diagramma classi

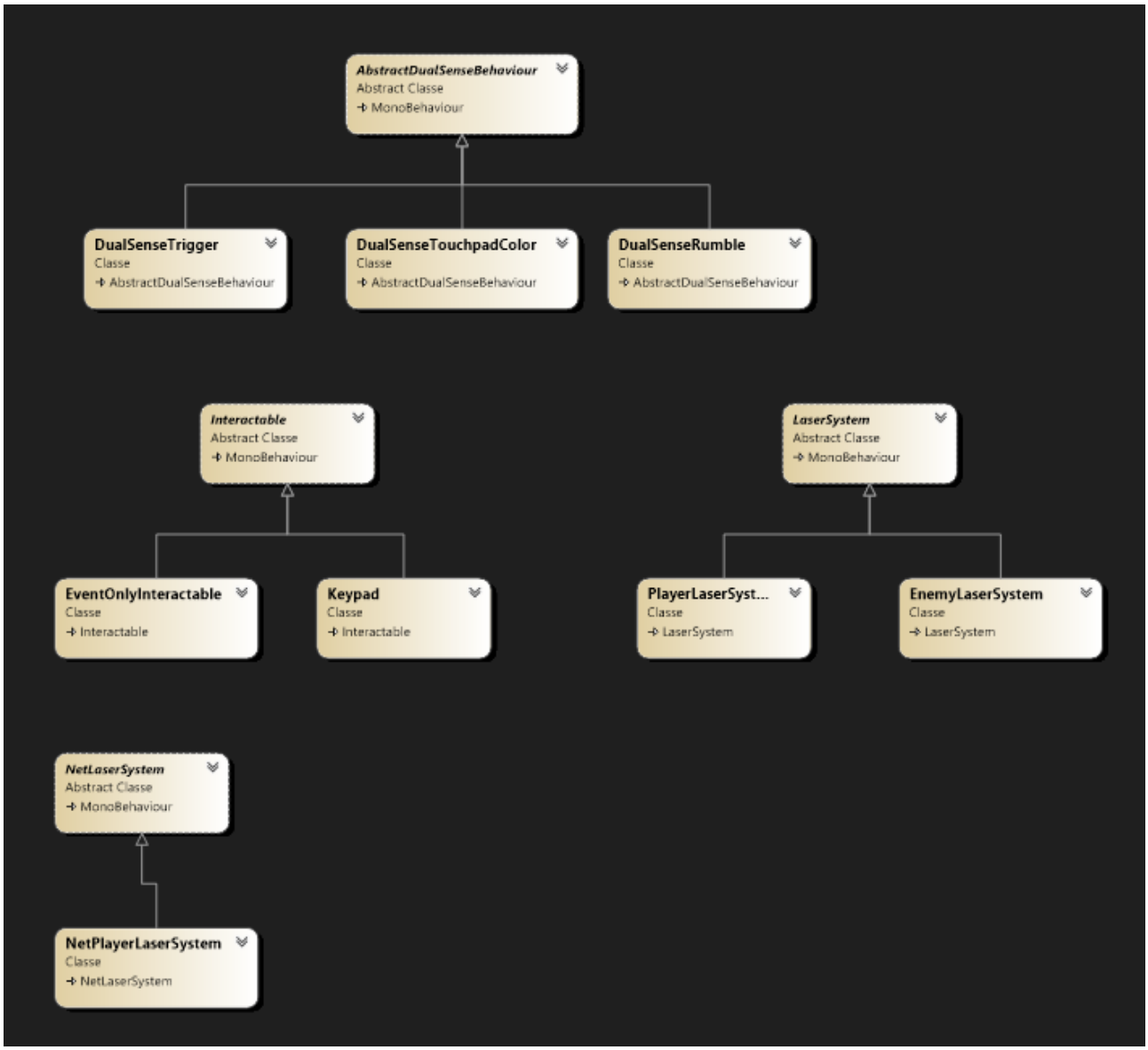


Figura 4 - Diagramma Classi 1.1

Figura 5 - Diagramma Classi 1.2

3.4 Design delle interfacce

Figura 6 - Interfaccia iniziale

La prima interfaccia visualizzata dall'utente comprende:

- Un bottone "Play" che fa iniziare la partita
- Un bottone "Settings" che permette all'utente di modificare delle impostazioni
- Un bottone "How to play" che spiega agli utenti i tasti per poter giocare
- Un bottone "Exit" che permette all'utente di uscire dal gioco

Figura 7 - Schermata selezione partita

Quando l'utente preme sul bottone "Play" arriva all'interfaccia di selezione partita che comprende:

- Il nome della lobby
- La quantità di giocatori già in partita
- Il bottone "Join" che permette all'utente di entrare in partita
- Il bottone "Create" che permette di creare una lobby

Figura 8 - Interfaccia lobby

Se l'utente preme sul bottone "Create" può creare una lobby che comprende:

- Un TextBox per l'inserimento del nome della lobby
- Il bottone "Create" per creare la lobby

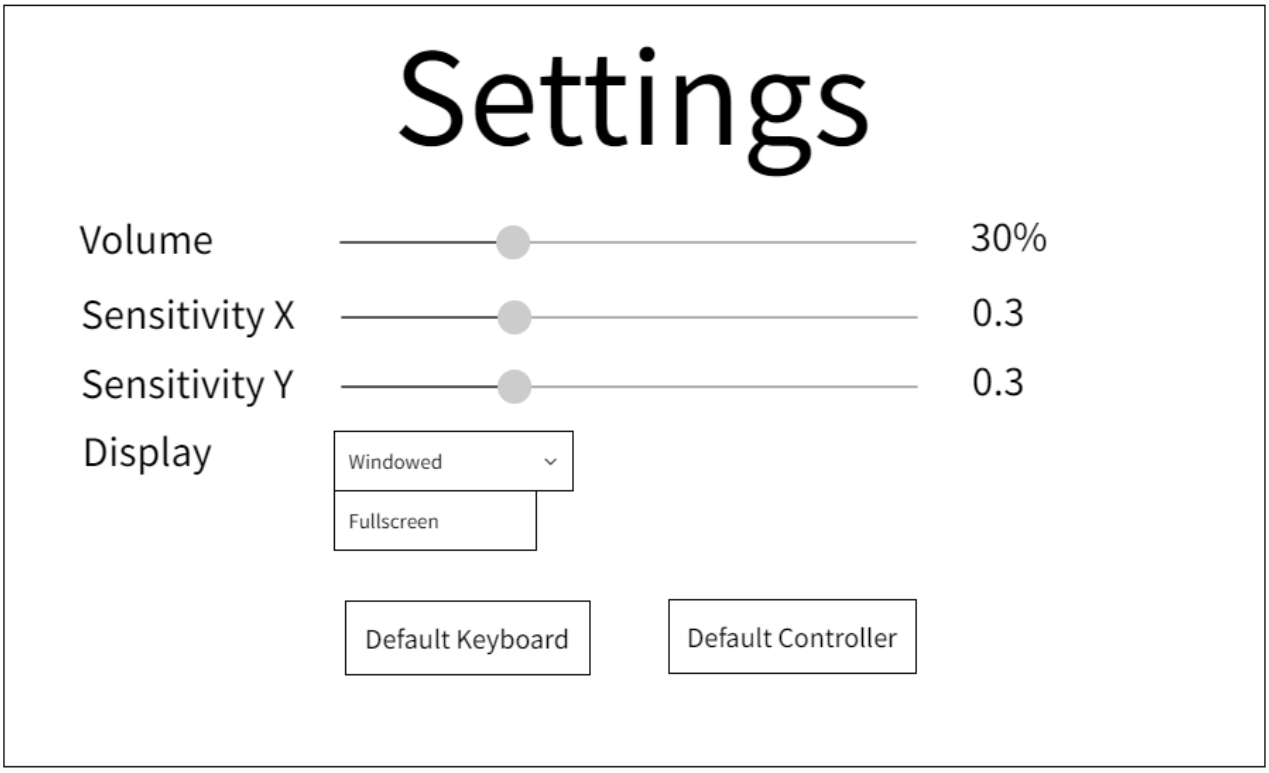


Figura 9 - Interfaccia Impostazioni

La schermata di impostazioni viene visualizzata quando l'utente preme sul bottone "Settings", essa comprende:

- Uno slider pel la regolazione del volume
- Uno slider pel la regolazione della sensibilità orizzontale
- Uno slider pel la regolazione della sensibilità verticale
- Un dropdown per la regolazione del display (a finestra o schermo intero)
- Un bottone che regola cambia gli slider ad un valore predefinito per l'uso della tastiera
- Un bottone che regola cambia gli slider ad un valore predefinito per l'uso della tastiera

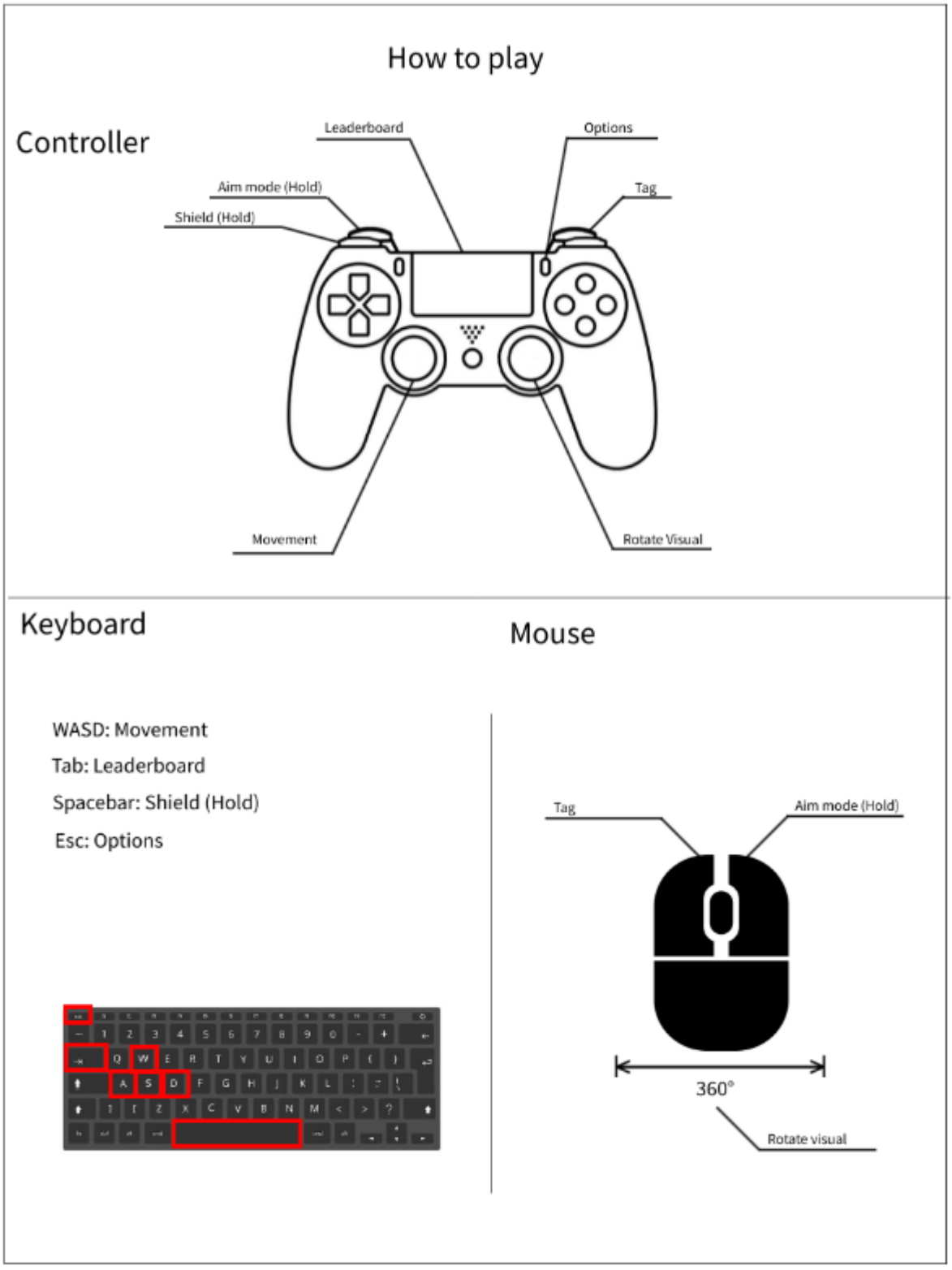


Figura 10 - Mappatura tasti (controller e tastiera)

La schermata How to play viene visualizzata quando l'utente preme sul bottone "How to play", questa schermata permette all'utente di poter visualizzare i tasti per giocare (sia controller che tastiera).

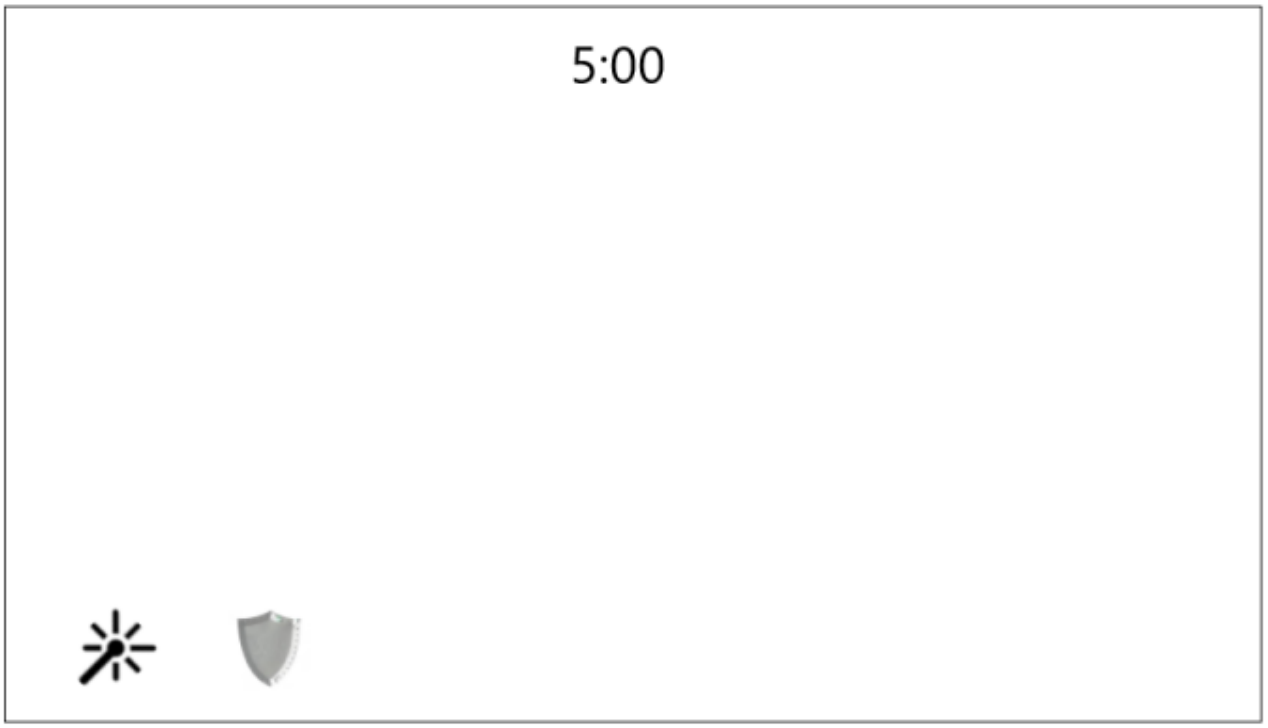


Figura 11 – Interfaccia base di gioco

Questa interfaccia appare quando il giocatore preme il bottone “Play”. Vengono visualizzati la possibilità di utilizzare il puntatore laser e lo scudo tramite due tasti del controller/tastiera, inoltre il player vede l’ambiente di gioco. Il tempo in alto è la durata della partita

Figura 12 - Interfaccia puntatore attivo

Questa interfaccia appare quando il giocatore sta utilizzando il puntatore laser impedendo di utilizzare lo scudo, inoltre il player vede l’ambiente di gioco. Il tempo in alto è la durata della partita

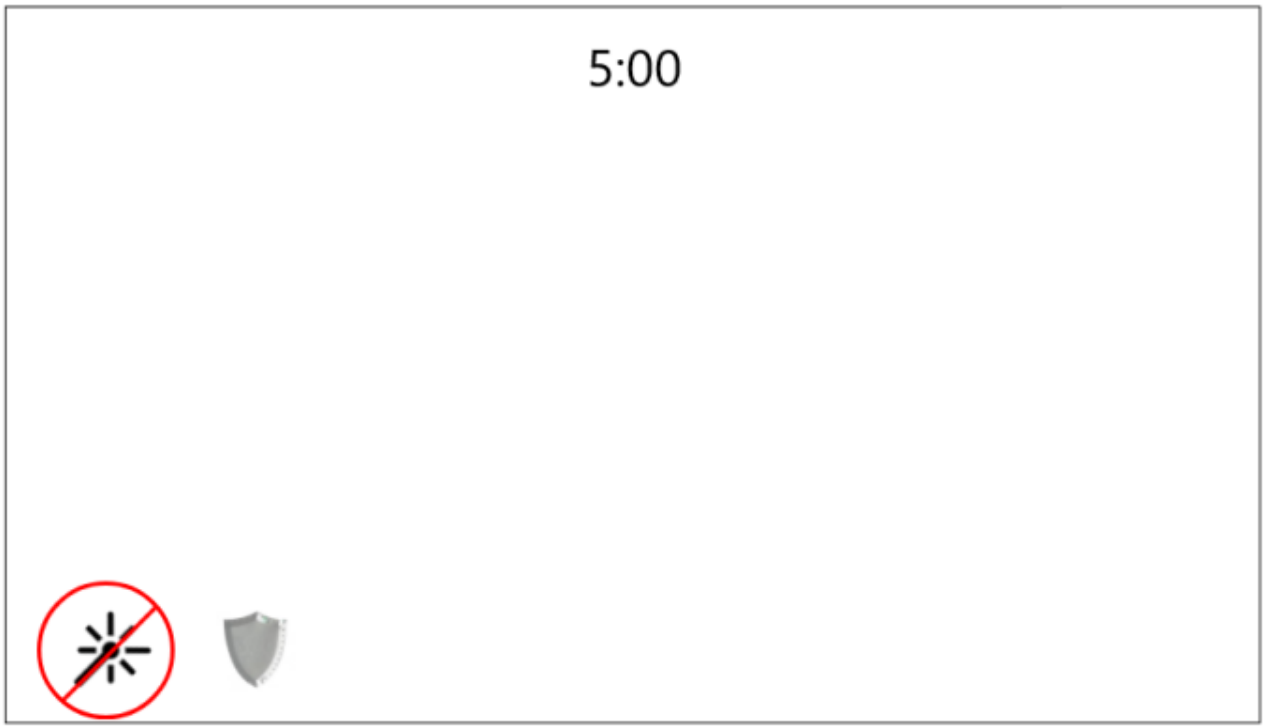


Figura 13 - Interfaccia scudo attivo

Questa interfaccia appare quando il giocatore sta utilizzando lo scudo impedendo di utilizzare il puntatore laser, inoltre il player vede l'ambiente di gioco. Il tempo in alto è la durata della partita

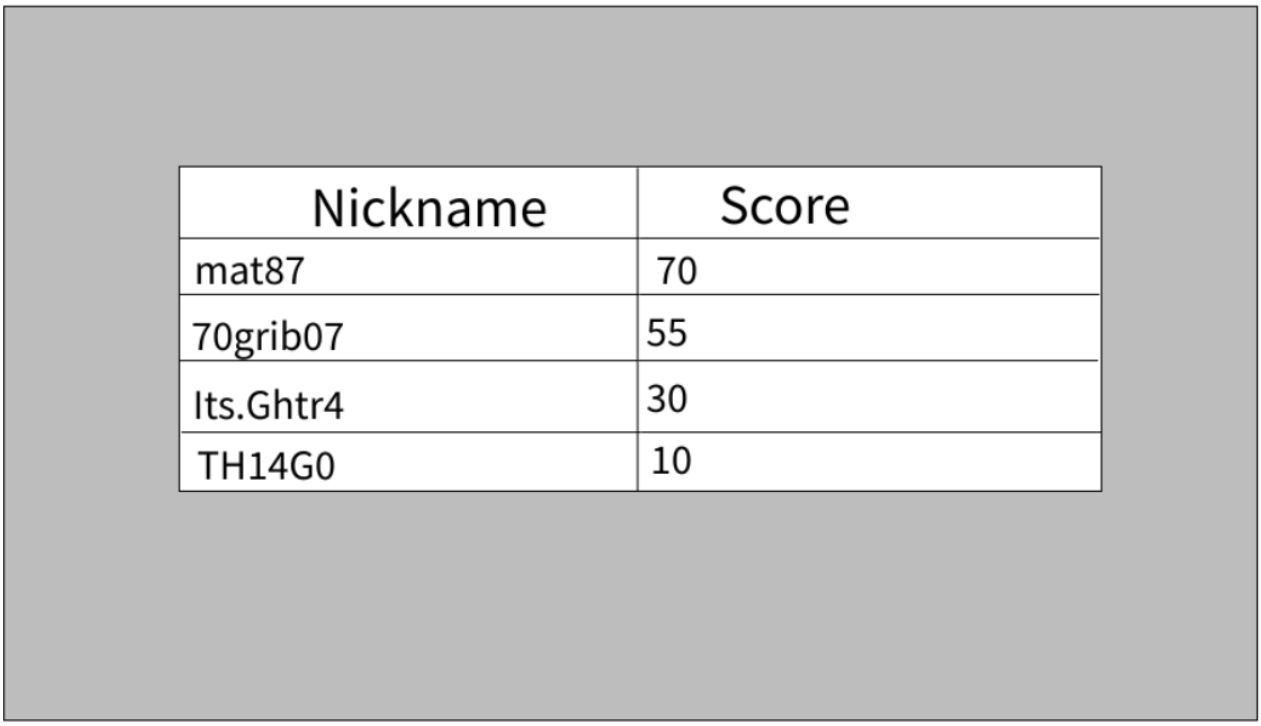


Figura 14 - Interfaccia classifica

Questa schermata appare durante la partita quando il player schiaccia su "Tab", viene visualizzata la classifica della partita. Inoltre il background diventa grigio.

3.5 Design procedurale

Descrive i concetti dettagliati dell'architettura/sviluppo utilizzando ad esempio:

- Diagrammi di flusso e Nassi.
- Tabelle.
- Classi e metodi.
- Tabelle di routing
- Diritti di accesso a condivisioni …

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

4 Implementazione

4.1 Interfacce

4.1.1 Main Scene

Figura 15 - Scena principale

Questa scena è la prima scena che visualizza l'utente subito dopo aver visto il video introduttivo, essa comprende:

- il Bottone “Play” che consente di visualizzare le lobby
- il Bottone “Play offline” che consente di giocare contro tre bot
- il Bottone “How to play” che consente di visualizzare i tasti sia di controller che di mouse e tastiera
- il Bottone “Settings” che consente di impostare le sensibilità di gioco
- il Bottone “Match history” che consente di visualizzare le partite passate
- il Bottone “Exit” che consente di uscire dal gioco
- il Bottone “Login/Logout” che consente di accedere è rispettivamente uscire con il proprio account
- il Bottone “Credits” che visualizza i ”titoli di coda” del gioco

4.1.2 Play Scene

Figura 16 - Scena scelta lobby

Quando l'utente clicca sul bottone play dopo aver fatto login vedrà questa schermata, essa è composta da:

- in alto a destra il Bottone “Load lobbies” che permette di visualizzare le partite non ancora iniziate
- in alto a sinistra un Bottone che permette di tornare indietro la scena iniziale
- il Bottone “Join” che permette all'utente di accedere alla partita dell'utente chiamato “User”
- il Bottone “Create” che permette di creare la partita a nome dell'utente che lo preme

4.1.3 How to Play Scene

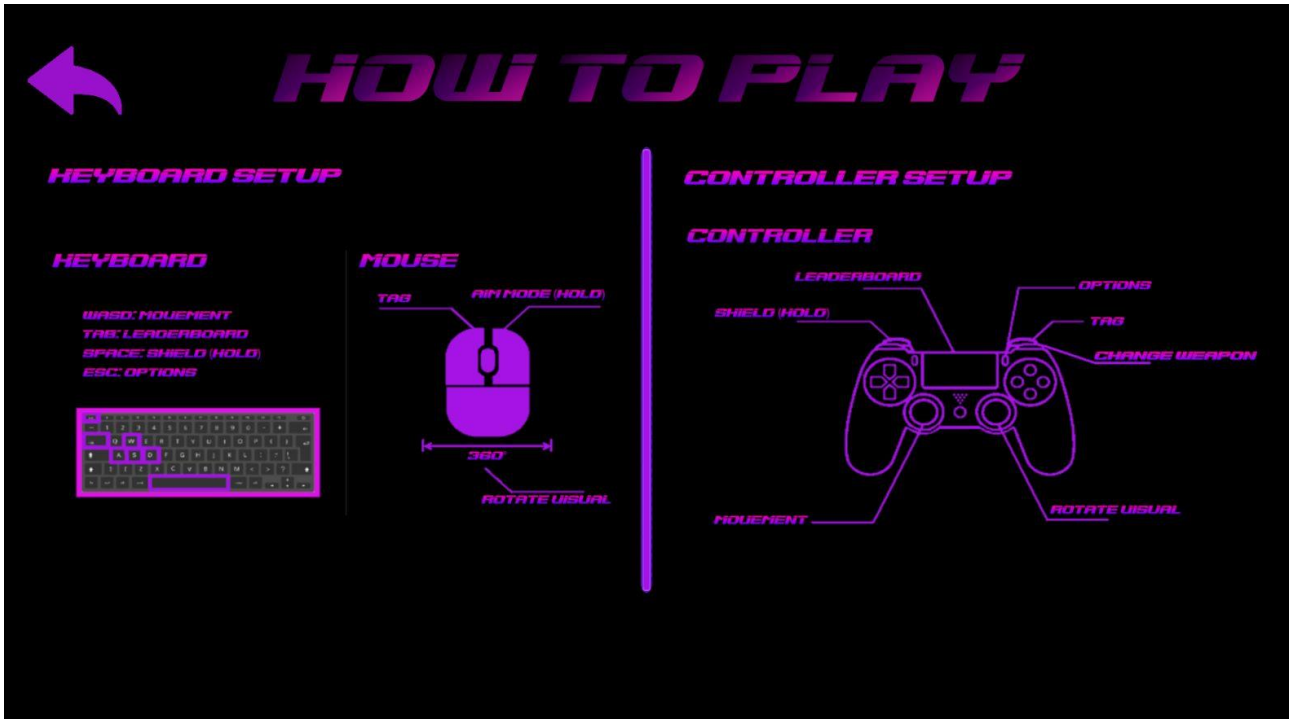


Figura 17 - Scena visualizzazione tasti

La schermata how to play verrà visualizzata quando l'utente clicca sul bottone how to play presente nella scena iniziale, essa è composta dalla mappatura dei tasti sia di mouse e tastiera sia del controller per permettere all'utente di visualizzare i tasti. Inoltre in alto a sinistra è presente un bottone per tornare alla scena iniziale.

4.1.4 Settings Scene

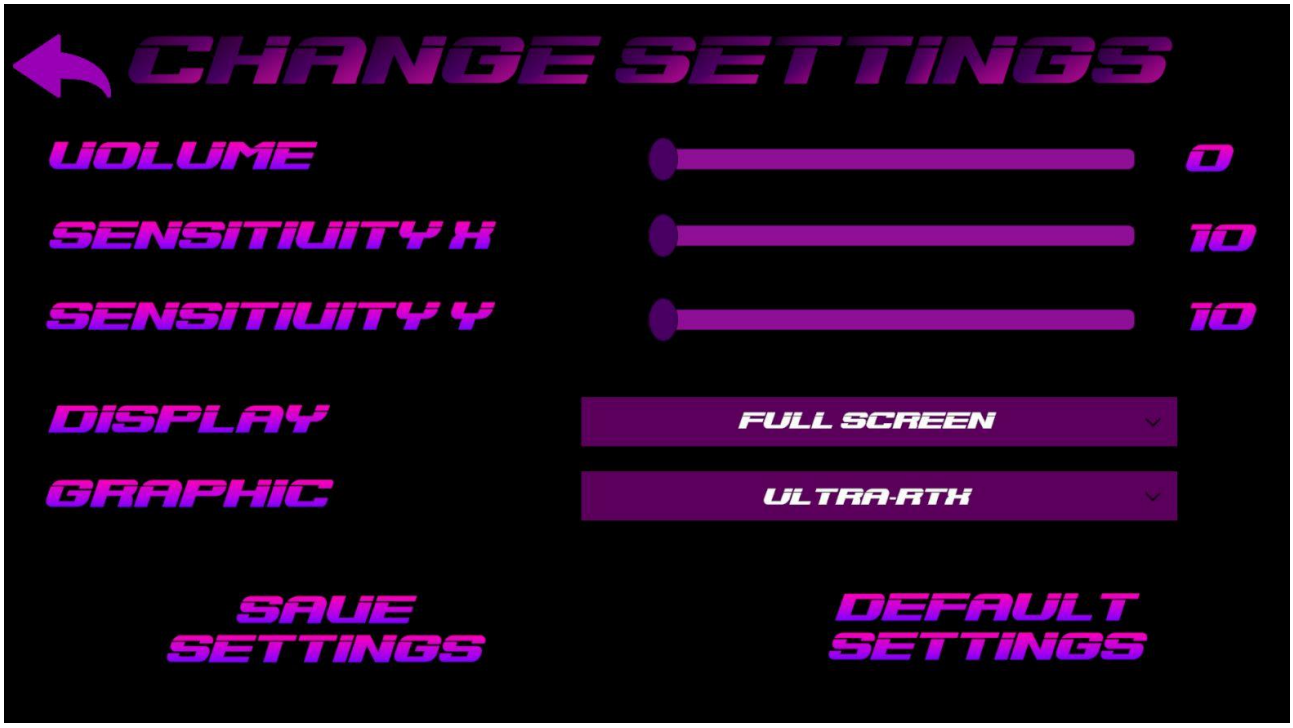


Figura 18 - Scena personalizzazione sensibilità

La scena “Change Settings” viene visualizzata quando l'utente premerà sul bottone settings nella scena iniziale, essa comprende:

- Uno Slider “Volume” che permette di modificare il volume di gioco
- Uno Slider “Sensitivity X” che permette di modificare la sensibilità X
- Uno Slider “Sensitivity Y” che permette di modificare la sensibilità Y
- Un Dropdown “Display” che permette di modificare il tipo di finestra (Full screen o Windowed)
- Un Dropdown “Graphics” che permette di modificare il tipo di garafica
- Un Pulsante “Save Settings” che permette di salvare le impostazioni selezionate
- Un Pulsante “Default Settings” che imposta automaticamente le impostazioni predefinite
- In alto a sinistra un Pulsante che permette di tornare alla scena iniziale

4.1.5 Match History Scene

Figura 19 - Scena Partite Passate

La scena “Match History” viene visualizzata quando l'utente preme sul bottone “Match History” nella scena iniziale essa è composta da una serie di tabelle contenenti tutte le informazioni riguardanti le partite passate, inoltre contiene in alto a sinistra un bottone che permette di tornare indietro alla scena iniziale.

4.1.6 Login Scene

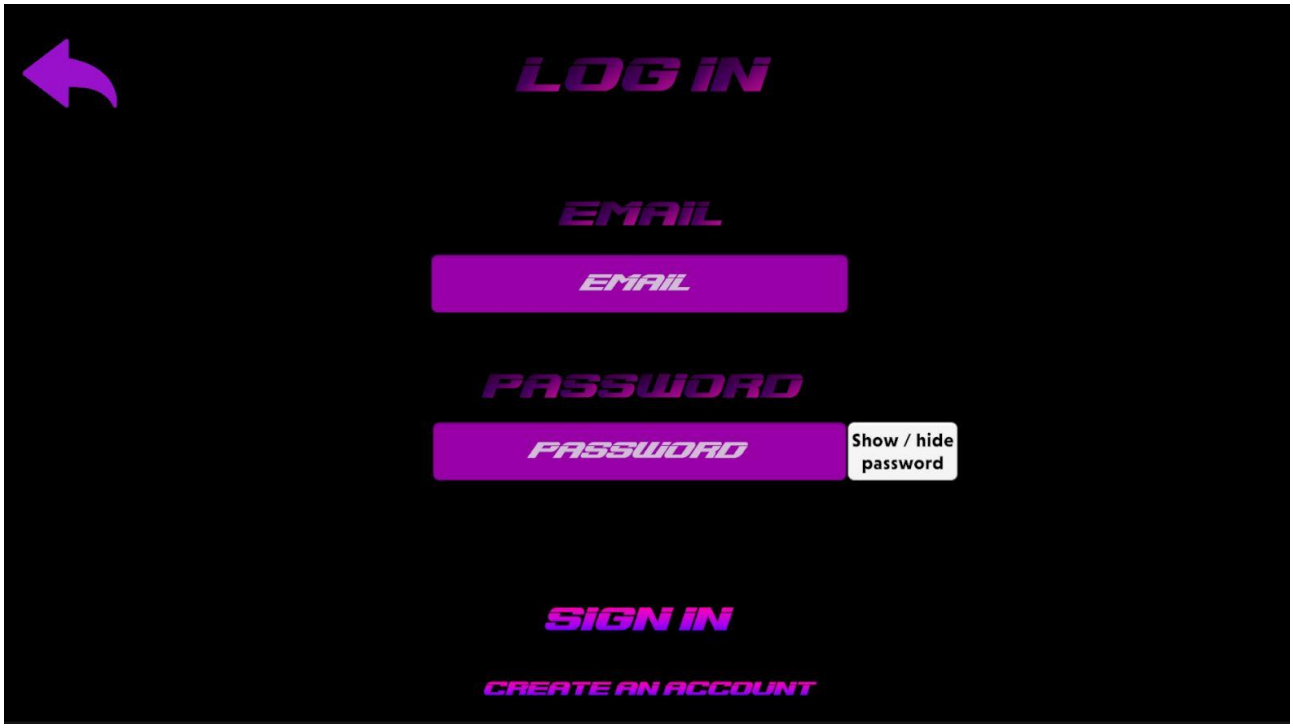


Figura 20 - Scena di Login

La schermata “Login” verrà visualizzata quando l'utente premerà sul pulsante “Login/Logout” presente nella scena iniziale, essa comprende:

- Un Textbox dove l'utente deve inserire la propria email
- Un Textbox dove l'utente deve inserire la password che può visualizzare anche in chiaro tramite il bottone “Show/Hide Password”
- Il pulsante “Sign in” che permette all'utente di accedere all'account
- Il pulsante “Create un account” che permette all'utente di creare un account
- In alto a sinistra un pulsante che permette di tornare indietro alla scena iniziale

4.1.7 Create Account Scene

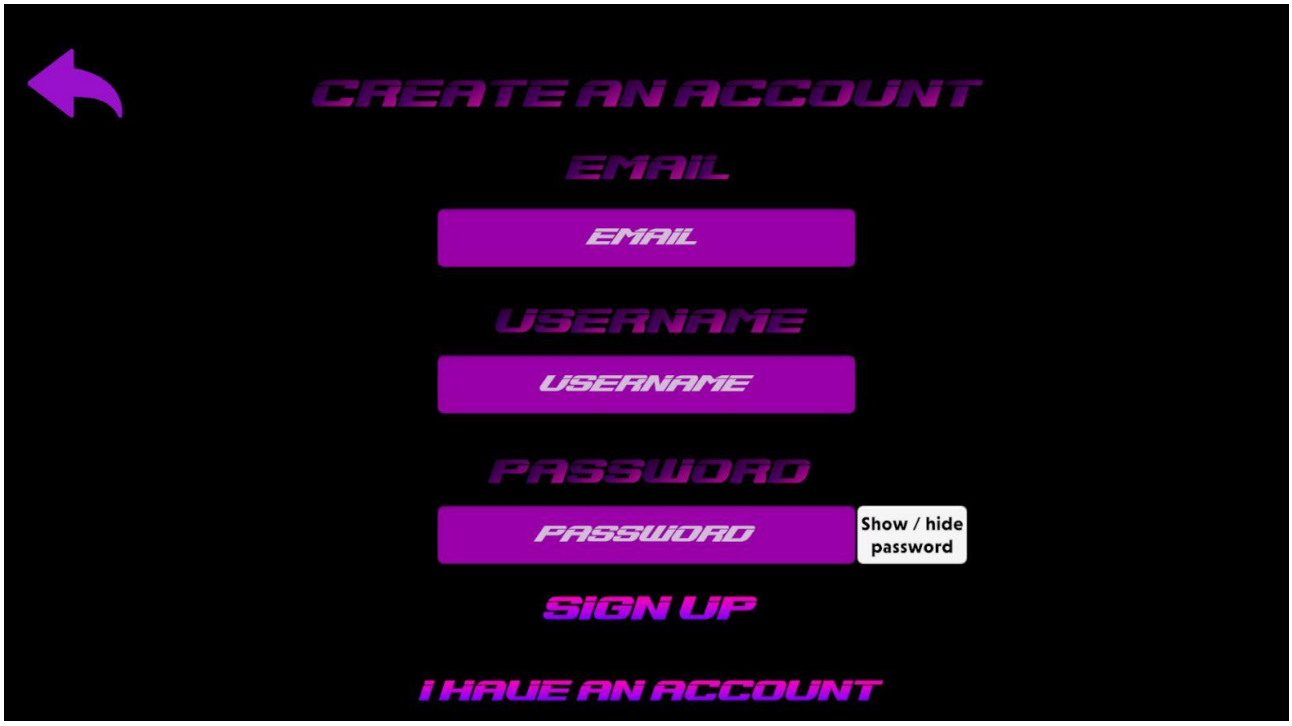


Figura 21 - Scena Creazione Account

La schermata di creazione account viene visualizzata quando l'utente preme sul pulsante “Create an Account” nella scena di login, essa comprende:

- Un Textbox dove l'utente deve inserire la propria email
- Un Texbox dove l'utente deve inserire il proprio nickname
- Un Textbox dove l'utente deve inserire la propria password che può visualizzare anche in chiaro tramite il bottone “Show/Hide Password”
- Il bottone “Sign up” che permette all'utente di creare l'account
- il bottone I have an account che permette all'utente di andare alla scena di login
- In alto a sinistra un pulsante che permette di tornare indietro alla scena iniziale

4.2 Scripts Interfacce

4.2.1 UIButtonManager

Questo script serve per abilitare l'effetto dei bottoni quando ci si pasa sopra, esso è formato dai seguenti metodi

- Start() va ad impostare l'immagine alla quale dare l'effetto
- MouseOver() va ad applicare l'effetto all'immagine
- MouseNotOver() va a rimuovere l'effetto all'immagine
- ExpandImage() va a rendere fluido l'effetto di ingrandimento immagine
- ShrimkImage() va a rimpicciolire l'immagine in maniera fluida

Questi due metodi servono per sottolineare tramite un'immagine i pulsanti grazie al loro delta x. Il primo metodo verifica se l'immagine non si sta espandendo e se il delta x corrente sia minore di quello impostato. Se questa condizione è vera, l'immagine si espanderà. Il secondo metodo fa esattamente il contrario.
I due metodi vengono chiamati nel MouseOver() e MouseNotOver().

4.2.2 GamepadCursor

Questo script serve per impostare il controller e il suo cursore, rispettivamente il mouse e la tastiera. I metodi che utilizza sono:

- CheckLastInput() serve a cambiare tra Mouse/Tastiera a Controller
- OnEnable() serve ad impostare un VirtualMouse per il controller
- UpdateMotion() serve a modificare la posizione del cursore del controller
- AnchorPosition() serve ad ancorare il cursore nella posizione corrente del controller

4.2.3 BackManager

Questo script serve per tornare indietro nelle interfacce, i suoi metodi sono:

- Start() va a prendere un oggetto di tipo InputManager
- Update() va ad impostare la scena se il pulsante è stato premuto
- SetInt(int id) imposta l'id della scena

4.2.4 SettingsManager

Questo script serve per permettere all'utente di impostare le proprie impostazioni preferite, avendo anche la possibilità di impostare quelle di default, esso contiene:

- Start() inserisce gli ultimi valori nelle PlayerPrefs
- Update() serve ad impostare le impostazioni scelte dall'utente
- Save() salva le impostazioni
- DefaultSettings() imposta le impostazioni di default

4.2.5 ShowFreeLobbies

Questo script serve per mostrare all'utente quali sono le lobby non ancora iniziate, essa comprende:

- DisplayLobbies() va a mostrare quali sono le lobby libere
- CreateTable() prende l'output dell'applicativo web e lo rende visibile dall'utente nell'applicativo client
- GetLobbiesFromDb() Tramite una richiesta prende dal DB i dati riguardanti le lobby

```
private IEnumerator GetLobbiesFromDb()
{
    WWWForm form = new WWWForm();
    UnityWebRequest www = UnityWebRequest.Post(GlobalVars.BASE_URL +
"matchManager/manageVacant/getLobbies", form);
    yield return www.SendWebRequest();
    if (www.result == UnityWebRequest.Result.Success)
    {
        jsonlobbies = www.downloadHandler.text;
        StartCoroutine(ReloadView());
    }
    else
    {
        jsonlobbies = "error";
    }
}
```

- ReloadView() serve a ricaricare la pagina contenente le partite

4.2.6 Logout

Questo metodo serve all'utente per poter fare logout, esso contiene un singolo metodo, ovvero LogOut() che elimina tutte le PlayerPrefs e torna alla scena iniziale
Game

4.2.7 GameEndedManager

Questa classe serve a caricare la schermata di fine gioco, è formata:

- Start() visualizza il punteggio dei player
- ReturnToMenu() serve a tornare alla scena iniziale

4.3 Core Scripts

Ora spiego le principali script per il funzionamento del gioco.

4.3.1 PlayerInput

Per impostare i tasti per giocare, abbiamo usato il “New Input System” di Unity che ci permetteva di assegnare in modo diretto e semplice i tasti della tastiera e controller al gioco.
Di seguito si possono vedere le sezioni create per gli input: “OnFoot” per il movimento e “OnAction” per le azioni con i tasti assegnati.

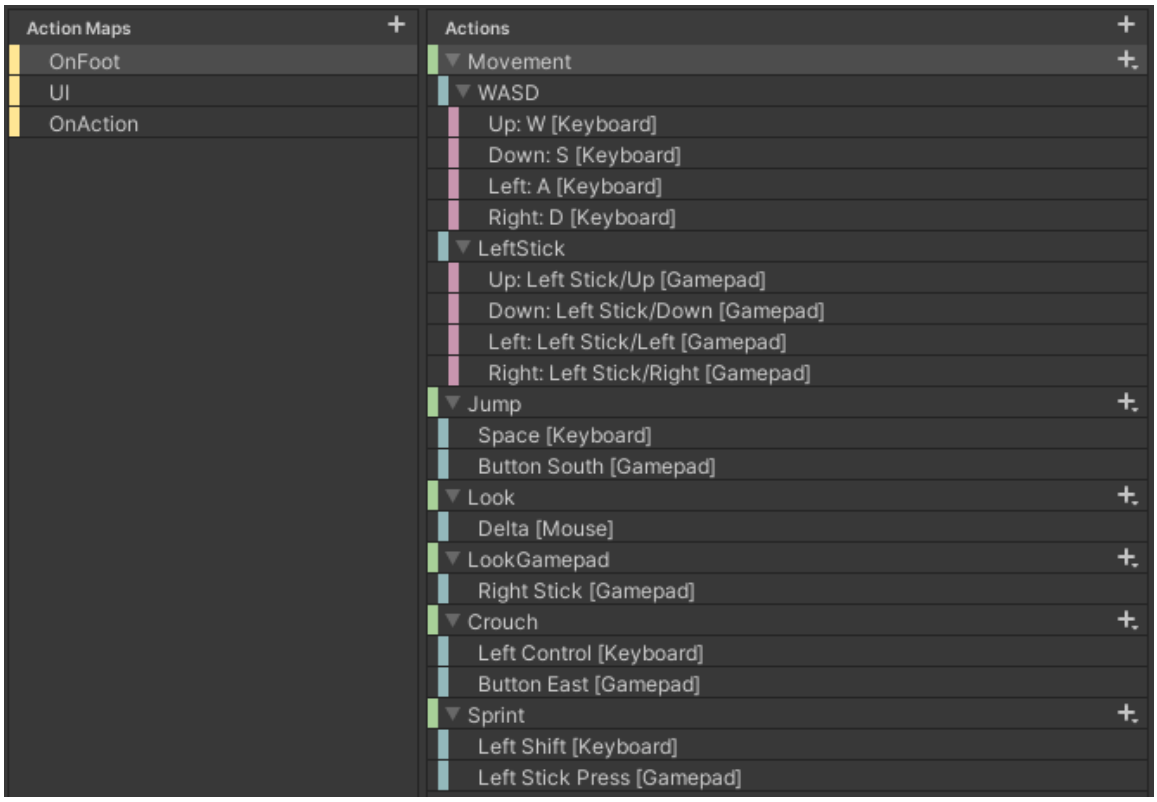


Figura 22 - Input Manager (Movimento)

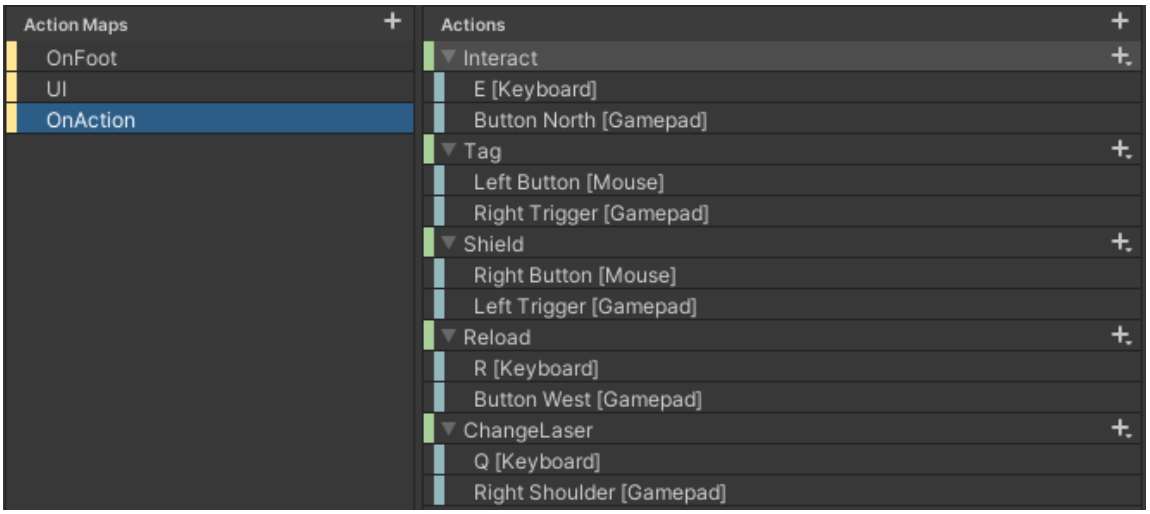


Figura 23 - Input Manager (Azioni)

Queste Action Maps vengono salvate in uno script compilato chiamato “PlayerInput.cs”.

4.3.2 InputManager

Serve per prendere i valori dal PlayerInput e usando gli script PlayerMotor e PlayerLook (prossimi 2 capitoli) convertirlo in funzionamenti.

Metodi:

- **Awake**: istanzio il PlayerInput e i componenti PlayerMotor e PlayerLook e aggiungo i metodi Jump e Crouch come delle azioni dal PlayerInput.
- **FixedUpdate**: dico al PlayerMotor di muovere usando i valori dall’azione “Movement”.
- **LateUpdate**: dico al PlayerLook di muovere la telecamera usando i valori di “Look” e “LookGamepad” (LookGamepad, semplicemente avrà una sensibilità più alta).
- **OnEnable**: serve per attivare i comandi PlayerInput.
- **OnDisable**: serve per disattivare i comandi PlayerInput.

4.3.3 PlayerMotor

Lo script PlayerMotor serve per i movimenti basi del player.

Metodi:

- **Start**: istanzio i componenti.
- **Update**: cambio velocità per lo sprint e/o l’altezza per il crouch.
- **ProcessMove**: serve per muovere il giocatore usando l’input preso dal giocatore e calcolando lo spostamento a dipendeza della gravità e la velocità e lo assegna al Character Controller.

```
public void ProcessMove(Vector2 input)
{
    Vector3 moveDirection = Vector3.zero;
    moveDirection.x = input.x;
    moveDirection.z = input.y;
    controller.Move(transform.TransformDirection(moveDirection) * speed *
Time.deltaTime);
    playerVelocity.y += gravity * Time.deltaTime;
    if(isGrounded && playerVelocity.y < 0)
    {
        playerVelocity.y = -2f;
    }
    controller.Move(playerVelocity * Time.deltaTime);
}
```

- **Jump**: serve per saltare; solo se non sei già nell’aria.
- **Crouch**: serve per cambiare l’altezza del giocatore.

4.3.4 PlayerLook

Lo script PlayerLook serve i movimenti della telecamera del giocatore.

Metodo:

- **ProcessLook**: prende l’input del giocatore e calcola la rotazione della telecamera (vedi sotto) e prende se il giocatore sta usando un controller oppure tastiera e poi aumenta la sensibilità.

```
public void ProcessLook(Vector2 input, bool isGamepad)
{
    [..]
    float mouseX = input.x;
    float mouseY = input.y;
    // Calculate camera rotation for looking up and down
    xRotation -= (mouseY * Time.deltaTime) * ySensitivity;
    xRotation = Mathf.Clamp(xRotation, -80f, 80f);

    // Apply this to our camera transform
    cam.transform.localRotation = Quaternion.Euler(xRotation, 0, 0);

    // Rotate player to look left and right
    transform.Rotate(Vector3.up * (mouseX * Time.deltaTime) * xSensitivity);
}
```

4.3.5 CameraShake

È un piccolo script con il metodo “Shake” che fa scuotere lo schermo per magnitudine e durazione specificata nei parametri.

4.3.6 LaserSystem

Lo script LaserSystem è uno degli script più essenziali per il gioco. Serve per far andare i laser e perciò la base del gioco; è una classe abstract di cui i nemici AI (solo offline) e player ereditano da.
Per cambiare il tipo di laser ho creato i seguenti attributi per differenziare ogni laser:

- Damage: danno.
- Time Each Burst: ogni volta che si invia un/i laser.
- Spread: la distanza massima dal punto in cui i laser colpisce (così è più realistico).
- Range: distanza.
- Reload Time: tempo di ricarica.
- Magazine Size: quanti laser si possono usare prima di ricaricare.
- Bullets Per Tap: quanti laser escono ogni volta che si preme il pulsante/tasto
- Allow Button Hold: se bisogna preme ogni volta per il laser oppure lasciare tenere premuto.

Metodi

- **Shoot**: è il metodo che quando eseguito invia un laser. In seguito c’è il codice con i passaggi commentati.

```
public void Shoot()
{
    readyToShoot = false;

    // Spread
    float x = Random.Range(-spread, spread);
    float y = Random.Range(-spread, spread);

    // Calculate Direction with Spread
    Vector3 raycastDirection = direction + new Vector3(x, y, 0);
```

```
        // Check what's in front using RayCast
        if (Physics.Raycast(position, raycastDirection, out rayHit, range))
        {
            // If it's another player/AI deal damage
            if (rayHit.collider.CompareTag("Enemy") ||
rayHit.collider.CompareTag("Player"))
            {
                rayHit.collider.GetComponent<PlayerHealth>().TakeDamage(damage);
                // Display bullet holes effect very briefly
                DisplayBulletHoleEffect(0.2f);
            }
            // If you hit a shield than it turns red
            else if (rayHit.collider.CompareTag("Shield"))
            {
                // Display bullet holes effect very briefly
                DisplayBulletHoleEffect(0.2f);
                StartCoroutine(rayHit.collider.GetComponent<ShieldMaterial>().chang
eMaterial());
            }
            else
            {
                // Display bullet holes effect very briefly
                DisplayBulletHoleEffect(5f);
            }
        }

        // Display laser effect briefly
        StartCoroutine(DisplayLaserEffect());
        lasersLeft--;
        lasersShot--;

        // If it's burst then allow more shots
        Invoke("ResetShot", timeEachBurst);

        // Keep on shooting after small period
        if (lasersShot > 0 && lasersLeft > 0)
            Invoke("Shoot", timeBetweenShots);
    }
```

- **ResetShot**: rimette a pronto lo sparo.
- **Reload**: blocca lo sparo e invoca il metodo ReloadFinished dopo il tempo di ricarico.
- **ReloadFinished:** avvisa che la ricarica è terminata
- **DisplayLaserEffect:** mostra l'effetto del laser (un pallino rosso sulla punta del laser nel nostro caso)
- **DisplayBulletHoleEffect**: mostra il punto colpito con il laser con un effetto (anche qua l'effetto è uno pallino rosso).
- **ChangeWeapon:** cambia il tipo di laser e rimette a 0 i colpi così non si riesci a barare.

4.3.7 PlayerLaserSystem

Come detto prima, eredità da LaserSystem e si adatta per funzionare con il giocatore.

Metodi:

- **Awake**: faccio i riferimenti agli oggetti nel gioco (ad esempio il testo per mostrare la vita) e istanzio i valori di default.
- **Update**: imposto il testo per le munizioni, imposto la direzione e posizione da dove dovrebbe inviare il laser e controllo gli Input del giocatore attraverso MyInput.
- **MyInput**: controlla cosa fa il giocatore, come se cambia l'arma, invia un laser oppure ricarica.

4.3.8 LaserPresets

È una classe in cui imposto 3 diverse preset di laser.

4.3.9 PlayerShield

Controlla se il pulsante per attivare lo scudo e premuto e poi attiva lo scudo (lo fa nell'Update).

4.3.10 ShieldMaterial

È uno script con il metodo ChangeMaterial che serve per cambiare il materiale dello scudo, per quando viene colpito dal laser.

4.3.11 DualSense

Ho importato un pacchetto per aggiungere le funzionalità del DualSense (UniSense) è sono 7 script per gestirlo.

4.4 Single Player Scripts

Ora spiego gli script che sono stati usati per il single player e perciò non implementati per il multiplayer.

4.4.1 Enemy

Il più grosso cambiamento tra il single player e multi player è che c'è l'AI nemico.

In questo script uso un NavMesh Agent per gestire il nemico.

Metodi:

- **Awake**: isanzio il PlayerManager
- **Start & Update**: assegno un target per l'agent (AI) usando il metodo GetClosestPlayer
- **GetClosestPlayer**: ritorna il giocatore/nemico più vicino a sé. Di seguito puoi vedere lo script commentato:

```
public GameObject GetClosestPlayer()
    {
        List<GameObject> players = playerManager.GetPlayers();

        // Simple calculation to transform a position to a relative number
        float agentPosition = Mathf.Abs(agent.transform.position.x) +
Mathf.Abs(agent.transform.position.z);
        float previousDistance = 0;
        GameObject closestPlayer = null;

        // Check all players/AIs
        for (int i = 0; i < players.Count; i++)
        {
            // Other player (not itself)
            if (players[i].transform.position != agent.transform.position && i !=
0)
            {
                float playerPosition = Mathf.Abs(players[i].transform.position.x) +
Mathf.Abs(players[i].transform.position.z);

                // Confront agent with other player / enemy
                float remainingDistance = Mathf.Abs(playerPosition -
agentPosition);

                // Check who is closest
                if (remainingDistance < previousDistance)
                {
                    previousDistance = remainingDistance;
                    closestPlayer = players[i];
                }
            }else if(i == 0)
            {
```

```
                previousDistance = Mathf.Abs(players[i].transform.position.x) +
Mathf.Abs(players[i].transform.position.z);
                closestPlayer = players[i];
            }
        }
        return closestPlayer;
    }
```

4.4.2 EnemyLaserSystem

Come detto prima, eredità da LaserSystem e si adatta per funzionare con i nemici.

Metodi:

- **Awake**: faccio i riferimenti agli oggetti nel gioco (ad esempio il nemico che è stato assegnato lo script).
- **Start**: istanzio i valori di default e dò il target ed un laser random usando ChangeWeapon ed eseguo il metodo EnemyTag.
- **Update**: aggiorno la direzione del nemico facendolo puntare il giocatore/nemico più vicino e lo faccio sparare dei laser usando MyInput.
- **MyInput**: controlla che è pronto per usare il laser e poi lo spara.
- **EnemyTag**: appena viene eseguito viene impostato quando può sparare facendo andare una variabile da true a false all'infinito ogni tot secondi.
- **ChangeWeapon**: faccio l'override del metodo del parent ed assegno un laser random.

4.4.3 MatchManager

È uno dei primi script eseguiti e semplicemente, attraverso il metodo Start assegno le posizioni di ogni giocatore/nemico ad un Respawn Point nell'arena.

```
void Start()
    {
        playerManager = new PlayerManager();
        players = playerManager.GetPlayers();
        int i = 0;
        // Position Players in arena
        foreach(GameObject player in players)
        {
            if(player.tag == "Player")
                player.transform.position =
playerManager.GetRespawnPositions()[i].transform.position;
            else
                player.transform.position =
player.GetComponent<NavMeshAgent>().nextPosition =
playerManager.GetRespawnPositions()[i].transform.position;
            i++;
        }
    }
```

4.4.4 PlayerManager

È una classe con dei metodi utili inerente ai player/nemici.

Metodi:

- **GetPlayers**: ritorna una lista di GameObject di tutti i player/nemici nella partita.
- **DespawnPlayer**: viene passato il player/nemico nel metodo per poi eseguire ChangePlayerState oppure ChangeEnemyState a dipendenza di che richiede questo metodo e passandogli i valori di disabilitare.
- **RespawnPlayer**: viene passato il player/nemico nel metodo per poi eseguire ChangePlayerState oppure ChangeEnemyState a dipendenza di che richiede questo metodo e passandogli i valori di abilitare.
- **ChangeEnemyState**: viene passato il nemico e lo disabilità/abilità a dipendenza del secondo valore passato.
- **ChangePlayerState**: viene passato il player e lo disabilità/abilità a dipendenza del secondo valore passato.

Questi 2 metodi sono divisi, perché i componenti da disabilitare tra il nemico e il player sono diversi.

- **GetRespawnPositions**: ritorna tutti i GameObjects che sono RespawnPoint
- **GetRandomRespawnPoint**: ritorna un valore Vector3 di una delle posizioni delle RespawnPoint.

4.4.5 PlayerHealth

Serve per gestire la vita dei player/nemici con un metodo visivamente più estetico del multi player.

Metodi:

- **Start**: vengono inseriti i valori di default e piazzate le barre della vita usando il metodo SpawnHealthBars.
- **SpawnHealthBars**: piazza la barra della vita più grande per il player e poi altri più piccoli per i nemici.
- **Update**: aggiorno di continuo tutte le barre della vita usando il metodo UpdateHealthUI e controllo se la vita è a zero, se si viene tolto usando DespawnPlayer di PlayerManager e poi rigenerato dopo 5 secondi usando WaitToRespawnPlayer. Se viene semplicemente colpito allora se sei il giocatore appare un effetto viola sul tuo schermo per indicare che sei a bassa vita.
- **UpdateHealthUI**: aggiorno la vita del giocato facendo un piccolo calcolo per dare un bellissimo effetto quando la tua vita diminuisce o aumenta e mostrando il valore sullo schermo.
- **TakeDamage**: passandogli un numero del danno, quando viene eseguito viene dedotto quel valore di danno ogni volta che viene eseguito questo metodo. Sei se il giocatore allora viene mostrato un effetto viola sullo schermo e viene scuotata.
- **RestoreHealth**: aggiunge vita al giocatore usando il parametro come valore d'aggiunta.
- **GetHealth**; serve per richiamare health da altri script.
- **WaitToRespawnPlayer**: dandogli un tempo in secondi come parametro il player/nemico viene rigenerato dopo quel tempo e ridato vita piena ed è invulnerabile per lo stesso tempo dopo essere rigenerato con una barra della vita blu.

4.4.6 Interactables

Ho degli script che ho preso da online per interagire con oggetti per eseguire altri codici. Lo usiamo per aprire la porta d'inizio gioco usando un GameObject pulsante dentro il gioco.

4.5 Multiplayer Scripts

4.5.1 NetManager

Questa classe gestisce la creazione di istanze di gioco (quindi gestisce la creazione Host e Client)

4.5.1.1 Metodo Start()

```
private void Start()
    {
        textIP.text = "IP: " + GetLocalIPAddress();
        string mode = PlayerPrefs.GetString("multiplayerMode");
        print(mode);
        switch (mode)
        {
            case "server":
                Debug.LogError("Impossible to instance a server! Not Supported!");
                Application.Quit();
                break;
            case "host":
                textIP.text += " (HOST)";
                //Setto le impostazioni di rete
                NetworkManager.Singleton.GetComponent<UnityTransport>().SetConnectionData(
                    "0.0.0.0", //dove connettersi. è host, quindi non server. lasciare
0.0.0.0
                    (ushort)6973, //porta di ascolto
                    PlayerPrefs.GetString("connectIp") //indirizzo ip del host.
                 );
                //Avvio il processo di rete e mi connetto
                NetworkManager.Singleton.StartHost();
                break;
            case "client":
                textIP.text += " (CLIENT)";
                //Setto le impostazioni di rete
                NetworkManager.Singleton.GetComponent<UnityTransport>().SetConnectionData(
                    PlayerPrefs.GetString("connectIp"), //indirizzo ip del host.-
                    (ushort)6973 //porta del host
                 );
                //Avvio il processo di rete e mi connetto
                NetworkManager.Singleton.StartClient();
                break;
        }
    }
```

In questo metodo, a seconda del **playerprefs** che si è passato dallo script StartGame.cs viene avviato un *Singleton* per l'host oppure viene reindirizzato al *Singleton* per la creazione del client.

4.5.1.2 GetLocalIPAddress

```
    //serve a ottenere l'indirizzo privato.
    //in futuro cambiarlo e ottenere quello pubblic
    public static string GetLocalIPAddress()
    {
        foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
        {
            if (ni.NetworkInterfaceType != NetworkInterfaceType.Loopback &&
ni.OperationalStatus == OperationalStatus.Up)
            {
                IPInterfaceProperties ipProps = ni.GetIPProperties();
                if (ipProps.GatewayAddresses.Count > 0)
                {
                    foreach (UnicastIPAddressInformation ip in ipProps.UnicastAddresses)
                    {
                        if (ip.Address.AddressFamily == AddressFamily.InterNetwork)
                        {
                            return ip.Address.ToString();
                        }
                    }
                }
            }
        }

        return "Indirizzo IP locale non trovato!";
    }
```

Questo metodo ritorna l'indirizzo IP locale della scheda ethernet principale (quindi ignora quelle secondarie)

4.5.2 NetworkMatchManager

Questa classe si occupa di gestire il match lato server, per questo eredita da **NetworkBehavior.**

4.5.2.1 Metodi

- StartMatch: si occupa di avviare il match
- EndMatch: si occupa di terminare la partita
- StartMatchServerRpc: (eseguito solo dal server) → Si occupa di richiamare una coroutine per avviare la partita
- EndMatchServerRpc: (eseguito solo dal server) → Si occupa di chiamare la coroutine per terminare la partita.

```
    [ServerRpc]
    private void EndMatchServerRpc()
    {
        DamageManager dm = null;
        foreach (GameObject item in GameObject.FindGameObjectsWithTag("Player"))
        {
            if (item.GetComponent<NetworkPlayer>().IsOwner)
            {
                dm = item.GetComponent<DamageManager>();
                break;
            }
        }
        scores = dm.GetPlayerScores();
        SetScoresClientRpc(scores);
        isMatchStarted.Value = false;
        InsertScoresOnDbClientRpc() }
```

- StartGameClientRpc: (eseguito solo dal client) → Si occupa di gestire lo start del match lato client
- InsertScoresOnDbClientRpc: (eseguito solo dal client) → Si occupa di gestire l'inserimento dei dati dei punteggi nel DB

4.5.3 ServerLife

Si occupa di gestire la durata della partita, quando la classe NetworkMatchManager avvia la partita il timer scende, quando arriva a zero viene notificata la classe antecedentemente menzionata.

4.5.3.1 Update()

```
private void Update()
{
    if (FindObjectOfType<NetworkMatchManager>().isMatchStarted.Value && IsHost)
    {
        timeRemaining.Value -= Time.deltaTime;
        if(timeRemaining.Value <= 0)
            FindObjectOfType<NetworkMatchManager>().EndMatch();
    }
    if(FindObjectOfType<NetworkMatchManager>().isMatchStarted.Value)
        text.text = "Time remaining: " + (int)timeRemaining.Value + " seconds";
}
```

Questo metodo si occupa di aggiornare il tempo di gioco e di visualizzarlo.

4.5.4 DamageManager

Questa classe si occupa di gestire lato server tutto quello che riguarda la ricezione del laser, ovvero toglie punti vita al giocatore.

4.5.4.1 Metodi

- Awake: istanzia tutte le **NetworkVariables**
- Update: aggiorna la vista della vita lato client
- GetPlayerScores: ottiene tutti i punteggi dei giocatori
- PlayerHittedServerRpc: (eseguito dal server) → è richiamato dai client quando sanno di aver colpito qualcuno con il laser
- AddDeathPointsServerRpc: (eseguito dal server) → è richiamato dai client quando un client sa che ha fatto terminare i punti vita ad un altro giocatore. Aggiunge i punti
- UpdateTextClientRpc: (eseguito dai client) → è richiamato dal metodo Update() e aggiorna i testi dei punti vita
- PlayerTaggedBySomeoneClientRpc (eseguito dai client) → è richiamato quando un player colpisce quancuno

4.5.5 NetworkPlayer

Questa classe si occupa di gestire il singolo player all'interno della rete di **Netcode for GameObjects**

4.5.5.1 Start

```
private void Start()
    {
        _camera = GetComponentInChildren<Camera>();
        _audioListener = GetComponentInChildren<AudioListener>();
        _playersManagement = FindObjectOfType<PlayersManagement>();

        if (!IsOwner) //Se chi richiama questo metodo non è il proprietario allora...
        {
            _audioListener.enabled = false; //...viene disabilitato l'audiolistener
            _camera.enabled = false; //...viene disabilitata la camera
            _playersManagement.enabled = false; //...viene disabilitato questo script

            //vengono disabilitati questi componeti
            GetComponent<NetworkObject>().enabled = false;
            GetComponent<NetworkPlayer>().enabled = false;
            GetComponent<DamageManager>().enabled = false;
            GetComponent<PlayerMotor>().enabled = false;
            GetComponent<PlayerLook>().enabled = false;
            return;
        }
        NetworkId = GetComponent<NetworkObject>().NetworkObjectId; //vado a prendere dal
server il mio id
        UserDbId = PlayerPrefs.GetInt(PlayerPreference.USER_ID);
        _playersManagement.ClientConnectedServerRpc(NetworkId); //avviso il server che mi
sono connesso
    }
```

In questo metodo vengono cercati ed eventualmente disabilitati componenti al GameObject. Vengono disabilitato solo se il GameObject del player è un clone (ovvero non quello che sto controllando).

4.5.5.2 Metodi

- Update: si occupa di controllare se il server si sta disconnettendo
- DisconnectAllClientsServerRpc: si occupa di disconnetter ogni client se il server crasha o termina la partita
- EndAllConnectionsClientRpc: è il metodo richiamato da quello precedente. Disconnette il client in maniera effettiva
- *Override* OnNetworkDespawn: è l'override del metodo bas:

```
public override void OnNetworkDespawn()
    {
        if (IsClient && IsLocalPlayer)
        {
            SceneManager.LoadScene((int)SceneToId.gameEnded);
            NetworkManager.Singleton.Shutdown();
        }
    }
```

4.5.6 PlayersManagement

Questa classe si occupa di gestire i players in rete

4.5.6.1 ClientConnectedServerRpc

```
[ServerRpc(RequireOwnership = false)]
    public void ClientConnectedServerRpc(ulong clientId) //il player si è connesso
    {
        GameObject[] rawList = GameObject.FindGameObjectsWithTag("Player");
        GameObject clientObj = null;
        foreach (GameObject obj in rawList)
        {
            if(obj.GetComponent<NetworkObject>().NetworkObjectId == clientId)
            {
                clientObj = obj;
                break;
            }
        }
        try
        {
            players.Add(clientId, clientObj);
            UpdateDictCount();
            Debug.Log("Client connected, id: " + clientId);
            ulong cId = clientId;
            if (clientId == 1)
                cId = 0;
            else
                cId -= (cId - 1);
            switch (cId)
            {
                case 0:
                    player1Id.Value = cId;
                    break;

                case 1:
                    player2Id.Value = cId;
                    break;

                case 2:
                    player3Id.Value = cId;
                    break;

                case 3:
                    player4Id.Value = cId;
                    break;

                default:
                    Debug.LogError("Error, clientId > 4! Actual ID: " + cId);
                    break;
            }
            Debug.Log("Joined client " + cId);
        }
        catch { }


    }
```

Questo metodo è chiamato dal client quando si connette

4.5.6.2 ClientDisconnectedServerRpc

```
[ServerRpc(RequireOwnership = false)]
    public void ClientDisconnectedServerRpc(ulong clientId, bool serverCrashed) //client
disconnesso
    {
        if (!IsOwner)
            return;
        Debug.Log("Client disconnected, id: " + clientId);
        if (!serverCrashed)
        {
            players.Remove(clientId);
            UpdateDictCount();
            return;
        }
        Debug.LogError("Server crashed!");
    }
```

Questo metodo viene richiamato quando il client si disconnette. Server anche a sapere se la disconnessione è dovuta al server che è crashato.

4.5.6.3 GetPlayers

```
    public IDictionary<ulong, GameObject> GetPlayers()
    {
        return players;
    }
```

Questo metodo ritorna il dizionario di player (contiene il loro GameObject e il NetworkObjectId → è un id che viene assegnato al GameObject per riconoscerlo in rete).

4.5.7 NetMatchManager

Questo script gestisce lato client il match.

4.5.7.1 Metodi

- Start: si occupa di una lista di players solo se viene eseguito dall'istanza del giocatore e non dai cloni

4.5.8 NetLaserSystem

Questa classe è quasi identica alla sua controparte non net (PlayerLaserSystem) ma con la differenza che qui quando un player viene colpito esegue questo codice:

```
         // If it's another player/AI deal damage
        if (rayHit.collider.CompareTag("Enemy") || rayHit.collider.CompareTag("Player"))
            {
                //Tells server to decrement player's life
                FindObjectOfType<DamageManager>().PlayerHittedServerRpc(
                    rayHit.collider.GetComponent<NetworkPlayer>().NetworkObjectId,

transform.parent.parent.parent.GetComponent<NetworkPlayer>().NetworkObjectId);
                // Display bullet holes effect very briefly
                DisplayBulletHoleEffect(0.2f);
            }
```

4.5.9 MapGenerator

Questa classe si occupa di generare la mappa di gioco in modo casuale.

4.5.9.1 GenerateMapServerRpc

```
[ServerRpc(RequireOwnership = false)]
    public void GenerateMapServerRpc()
    {
        for (int i = 0; i < walls.Length; i++)
        {
            wallsBefore[i] = walls[i]; //copio l'array precedente
            walls[i] = UnityEngine.Random.Range(0, 100) < PROBABILITY; //genero la nuova
mappa
        }
        serializedWalls.Value = ArrayToString(walls, ','); //serializzo l'array nella net
var
        serializedWallsBefore.Value = ArrayToString(wallsBefore, ','); //serializzo l'array
nella net var
        StartCoroutine(SendCommandToClientCooldown()); //attesa
    }
```

Questo metodo server per generare l'array. Lo serializziamo in una stringa perché nei metodi ClientRpc / ServerRpc e nelle NerworkVariables non possono essere passati oggetti ma solo i tipi primitivi e alcune classi di Unity serializzabili.

4.5.9.2 ApplyConfigurationClientRpc

```
[ClientRpc]
    private void ApplyConfigurationClientRpc()
    {
        string[] serializedWallsArray = serializedWalls.Value.ToString().Split(",");
        string[] serializedWallsBeforeArray =
serializedWallsBefore.Value.ToString().Split(",");
        walls = new bool[WALL_COUNT];
        wallsBefore = new bool[WALL_COUNT];
        wallInfoDebug.text = "ClientWalls:" + "\r\n" + serializedWalls.Value; //debug
        for (int i = 0; i < walls.Length; i++) //vado a riformare l'array di bool dalla
stringa serializzata
        {
            if (serializedWallsArray[i].Equals("True"))
            {
                walls[i] = true;
            }
            else
            {
                walls[i] = false;
            }
            if (serializedWallsBeforeArray[i].Equals("True"))
            {
                wallsBefore[i] = true;
            }
            else
            {
                wallsBefore[i] = false;
            }
        }
        for (int i = 0; i < walls.Length; i++) //controllo lo stato precedente e decido se
abbassare o alzare il muro
        {
            if (walls[i] == wallsBefore[i])
            {
                continue;
            }
            if(walls[i] && !wallsBefore[i])
            {
                StartCoroutine(TranslateObject(wallsPhysic[i], .2f, Vector3.zero, 0, deltaH,
0));
            }
            else
            {
                StartCoroutine(TranslateObject(wallsPhysic[i], .2f, Vector3.zero, 0, -
deltaH, 0));
            }
        }
    }
```

Questo metodo si occupa di *applicare* la configurazione generata dal metodo precedente.

4.6 Differenze Single player e Multiplayer

	Single player	Multiplayer
AI	**X**	
Database e Leaderboard		**X**
Tempo		**X**
Muri che si alzano		**X**
Vita visualmente estetica	**X**	
Porte interagibili	**X**	
Punteggio		**X**

5 Test

5.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

5.1.1 Test Case Obbligatori

Test Case:	**TC-001**	Nome:	**Movimento Giocatore**
Riferimento**:**	**REQ-01**		
Descrizione:	**L'utente può usare dei comandi di tastiera o joystick per muovere il personaggio.**		
Procedura:	**1. L'utente apre il gioco** **2. Inizia una partita** **3. Si sposta con controller o tastiera**		
Risultati attesi:	**L'utente è in grado di spostarsi in giro per la mappa sia con controller che con tastiera**		

Test Case:	**TC-002**	Nome:	**Interfaccia grafica**
Riferimento**:**	**REQ-02**		
Descrizione:	**L'utente può navigare attraverso l'interfaccia di gioco usando tastiera o joystick.**		
Procedura:	**1. L'utente apre il gioco** **2. Seleziona I vari oggetti della schermata**		
Risultati attesi:	**L'utente è in grado di interagire con gli oggetti delle schermate sia con controller che con tastiera**		

Test Case:	**TC-003**	Nome:	**Ambiente Partita**
Riferimento**:**	**REQ-03**		
Descrizione:	**Deve essere presente una mappa di gioco.**		
Procedura:	**1. L'utente apre il gioco** **2. Preme sul bottone "Play"** **3. Entra in partita**		
Risultati attesi:	**L'utente visualizza la mappa di gioco con il personaggio**		

Test Case:	**TC-004**	Nome:	**Sito web**
Riferimento**:**	**REQ-04**		
Descrizione:	**Deve essere presente un sito web**		
Procedura:	**1. L'utente apre un browser** **2. Entra nel sito web**		
Risultati attesi:	**L'utente visualizza il sito web**		

Test Case:	**TC-005**	Nome:	**Creazione account**
Riferimento**:**	**REQ-13**		
Descrizione:	**L'utente crea un account su Unity**		
Procedura:	**1. L'utente entra nel gioco** **2. Crea un account**		
Risultati attesi:	**L'utente ha un account personale su Unity**		

Test Case:	**TC-006**	Nome:	**Interazioni giocatore**
Riferimento**:**	**REQ-05**		
Descrizione:	**L'utente può mirare e puntare un laser ed eventualmente parare i laser degli altri usando i comandi di tastiera o joystick.**		
Procedura:	**1. L'utente avvia una partita** **2. Mira e punta un avversario** **3. Si para con lo scudo**		
Risultati attesi:	**L'utente è in grado di utilizzare tutte le mosse del personaggio**		

Test Case:	**TC-007**	Nome:	**Ostacoli casuali**
Riferimento**:**	**REQ-06**		
Descrizione:	**I muri si generano in maniera casuale all'inizio di ogni partita usando l'algoritmo di Manhattan Mapper.**		
Procedura:	**1. L'utente avvia una partita** **2. Visualizza la posizione casual dei muri**		
Risultati attesi:	**L'utente può giocare su delle mappe tutte diverse**		

Test Case:	**TC-008**	Nome:	**HUD**
Riferimento**:**	**REQ-07**		
Descrizione:	**L'Heads-Up Display (HUD) mostrerà varie informazioni durante la partita.**		
Procedura:	**1. L'utente avvia una partita** **2. L'utente guarda il tempo scorrere**		
Risultati attesi:	**L'utente può giocare per un tot di tempo una determinate partita**		

Test Case:	**TC-009**	Nome:	**Incremento punteggio leaderboard**
Riferimento**:**	**REQ-07**		
Descrizione:	**Ad ogni colpo del puntatore laser su un avversario il punteggio aumenta**		
Procedura:	**1. L'utente avvia una partita** **2. Punta un avversario** **3. Guarda nella leaderboard il punteggio**		
Risultati attesi:	**L'utente può visualizzare il suo punteggio incrementato ad ogni colpo corretto**		

Test Case:	**TC-010**	Nome:	**AI Nemico**
Riferimento**:**	**REQ-08**		
Descrizione:	**Un'AI che si comporta come un giocatore.**		
Procedura:	**1. L'utente avvia una partita** **2. L'utente gioca senza altri 3 giocatori** **3. Vengono generate le AI che giocano come una persona**		
Risultati attesi:	**L'utente può giocare anche con solamente AI**		

Test Case:	**TC-011**	Nome:	**Database**
Riferimento**:**	**REQ-09**		
Descrizione:	**Il database contiene tutte le tabelle utili per lo storage di dati e per il multiplayer.**		
Procedura:	**1. L'utente avvia una partita** **2. Punta un avversario** **3. Controlla che il punteggio si sia incrementato**		
Risultati attesi:	**L'utente può visualizzare il suo punteggio incrementato ad ogni colpo corretto**		

Test Case:	**TC-012**	Nome:	**Tabella Leaderboard Globale**
Riferimento**:**	**REQ-09**		
Descrizione:	**tabella consultabile dal sito web dove verranno mostrati i migliori giocatori di sempre con il punteggio**		
Procedura:	**1. L'utente Accede al sito web** **2. Va nella sezione leaderboard** **3. Controlla la leaderboard**		
Risultati attesi:	**L'utente può visualizzare la classifica dei migliori punteggi**		

Test Case:	**TC-013**	Nome:	**Multiplayer**
Riferimento**:**	**REQ-10**		
Descrizione:	**Il giocatore può collegarsi ad una sessione con altri giocatori.**		
Procedura:	**1. L'utente seleziona una lobby** **2. Avvia una partita** **3. Vede gli altri giocatori**		
Risultati attesi:	**L'utente può giocare con altri giocatori**		

5.2 Risultati test

TC-xxx	Risultato	Descrizione errore	Data
TC-001	Passato		05.05.2023
TC-002	Passato		05.05.2023
TC-003	Passato		05.05.2023
TC-004	Non Passato	Non fatto in accordo con il Committente	05.05.2023
TC-005	Passato		05.05.2023
TC-006	Parzialmente Passato	Nel multiplayer non funziona	05.05.2023
TC-007	Passato		05.05.2023
TC-008	Passato		05.05.2023
TC-009	Passato		05.05.2023
TC-010	Passato		05.05.2023
TC-011	Passato		05.05.2023
TC-012	Passato		05.05.2023
TC-013	Passato		05.05.2023

Per controprova visualizzare allegato “Risultati_Test_TAG.docx”

5.3 Mancanze/limitazioni conosciute

5.3.1 Sito WEB

L’idea iniziale era che il prodotto sarebbe stato accompagnato da un sito web dove sarebbe stato possibile gestire il proprio account e vedere i risultati dei match (anche in live). Durante il progetto ci siamo accorti di un nostro piccolo errore nella progettazione, quindi non saremo riusciti a terminare la parte web del progetto. Abbiamo quindi deciso, in accordo con il committente, di eliminare il sito web e di integrare queste funzioni direttamente nell’applicativo.

5.3.2 Funzionamento Impostazioni

Per quanto riguarda il funzionamento delle impostazioni non abbiamo fatto in tempo ad implementarlo, i pulsanti e gli slider funzionano ma non impostano al giocatore le cose.

5.3.3 Funzionamento in WAN

L’idea era quella di fare funzionare tutto in WAN, però non siamo riusciti perché da scuola non si può accedere all’esterno e l’API che volevamo usare non funzionava.

6 Consuntivo

✓		**T.A.G.**	**155 h**	**11.19 giorni?**	**ven 27.01.23**	**ven 05.05.23**		
✓		**Analisi**	**18 h**	**2 giorni?**	**ven 27.01.23**	**ven 10.02.23**		
✓		Stesura QdC	1.5 h	0.08 giorni?	ven 27.01.23	ven 27.01.23		Michael Y. Dobeson;Sim
✓		Requisiti (Documentazione)	3 h	0.17 giorni?	ven 27.01.23	ven 27.01.23	3	Michael Y. Dobeson; Simone Riva;Stefano
✓		Use Case	3 h	0.5 giorni?	ven 27.01.23	ven 27.01.23	4	Stefano Ceschi
✓		Gantt Preventivo	3 h	0.5 giorni?	ven 27.01.23	ven 27.01.23	4	Michael Y. Dobeson
✓		Test Case	6 h	1 giorno?	ven 27.01.23	ven 03.02.23	5	Simone Riva
✓		Abstract	1.5 h	0.25 giorni?	ven 03.02.23	ven 10.02.23	6;7	Simone Riva
✓		**Progettazione**	**16.5 h**	**1.25 giorni?**	**ven 10.02.23**	**ven 17.02.23**		
✓		Design Interfacce	3 h	0.5 giorni?	ven 10.02.23	ven 10.02.23	2	Simone Riva
✓		Diagramma di Flusso	6 h	1 giorno?	ven 10.02.23	ven 17.02.23	2	Michael Y. Dobeson
✓		Diagramma E/R (Database)	1.5 h	0.25 giorni?	ven 10.02.23	ven 10.02.23	2	Stefano Ceschi
✓		Diagramma di Classi	6 h	1 giorno?	ven 10.02.23	ven 17.02.23	2;12	Stefano Ceschi
✓		**Implementazione**	**82.5 h**	**5.83 giorni?**	**ven 17.02.23**	**ven 21.04.23**		
✓		**Unity**	**81.5 h**	**5.83 giorni?**	**ven 17.02.23**	**ven 21.04.23**		
✓		**Movimeto Giocatore**	**4.5 h**	**0.75 giorni?**	**ven 17.02.23**	**ven 03.03.23**		
✓		Spostamento	1.5 h	0.25 giorni?	ven 17.02.23	ven 17.02.23	9	Michael Y. Dobeson
✓		Visuale	3 h	0.5 giorni?	ven 17.02.23	ven 03.03.23	17	Michael Y. Dobeson
✓		**Interfaccia Grafica**	**15 h**	**5.67 giorni?**	**ven 17.02.23**	**ven 31.03.23**		
✓		Scene di Interfacce	5 h	0.83 giorni?	ven 17.02.23	ven 03.03.23	9	Simone Riva
✓		Navigazione con Joystick	8 h	1.33 giorni?	ven 03.03.23	ven 17.03.23	20	Simone Riva
✓		HUD	1 h	0.17 giorni?	ven 10.03.23	ven 10.03.23	21	Simone Riva
✓		Funzionamento Impostazioni	1 h	0.17 giorni?	ven 31.03.23	ven 31.03.23	27;31;22	Simone Riva
✓		**Ambiente Partita**	**6 h**	**1 giorno?**	**ven 10.03.23**	**ven 17.03.23**		
✓		Arena	3 h	0.5 giorni?	ven 10.03.23	ven 17.03.23	22	Simone Riva
✓		Ostacoli Casuali	3 h	0.5 giorni?	ven 17.03.23	ven 17.03.23	25;27	Michael Y. Dobeson
✓		**Interazioni Giocatore**	**10.5 h**	**1.5 giorni?**	**ven 03.03.23**	**ven 10.03.23**		
✓		Puntatore Laser	4.5 h	0.75 giorni?	ven 03.03.23	ven 03.03.23	16	Michael Y. Dobeson
✓		Mirare Laser	3 h	0.5 giorni?	ven 03.03.23	ven 10.03.23	28	Michael Y. Dobeson
✓		Scudo Anti-Laser	3 h	0.5 giorni?	ven 10.03.23	ven 10.03.23	29	Michael Y. Dobeson
✓		**AI**	**10.5 h**	**1.75 giorni?**	**ven 17.03.23**	**ven 31.03.23**		
✓		Algoritmo	4.5 h	0.75 giorni?	ven 17.03.23	ven 24.03.23	26	Michael Y. Dobeson
✓		Implementazione in Unity	6 h	1 giorno?	ven 24.03.23	ven 31.03.23	32	Michael Y. Dobeson
		~~Difficoltà~~	~~3 h~~	~~0.5 giorni?~~	~~ven 31.03.23~~	~~ven 31.03.23~~	~~33~~	~~Michael Y. Dobeson~~
✓		Unity Leaderboard	5 h	0.83 giorni?	ven 31.03.23	ven 21.04.23	40;31	Michael Y. Dobeson
✓		**Multiplayer**	**30 h**	**5 giorni?**	**ven 17.02.23**	**ven 31.03.23**		
✓		Ricerca	6 h	1 giorno?	ven 17.02.23	ven 03.03.23	40	Stefano Ceschi
✓		Prove	12 h	2 giorni?	ven 03.03.23	ven 17.03.23	37	Stefano Ceschi
✓		Implementazione	12 h	2 giorni?	ven 17.03.23	ven 31.03.23	38	Stefano Ceschi
✓		Database	1 h	0.17 giorni?	ven 17.02.23	ven 17.02.23	9	Stefano Ceschi
		~~Sito Web~~	~~12 h~~	~~2 giorni?~~	~~ven 31.03.23~~	~~ven 28.04.23~~		
		~~Struttura~~	~~3 h~~	~~0.5 giorni?~~	~~ven 31.03.23~~	~~ven 21.04.23~~	~~36~~	~~Stefano Ceschi~~
		~~Funzionamento JavaScript~~	~~3 h~~	~~0.5 giorni?~~	~~ven 21.04.23~~	~~ven 21.04.23~~	~~42~~	~~Stefano Ceschi~~
		~~Funzionamento PHP~~	~~6 h~~	~~1 giorno?~~	~~ven 21.04.23~~	~~ven 28.04.23~~	~~43~~	~~Stefano Ceschi~~
✓		**Conclusione**	**18 h**	**1 giorno?**	**ven 21.04.23**	**ven 28.04.23**		
✓		Test Case	9 h	0.5 giorni?	ven 21.04.23	ven 21.04.23	14	Michael Y. Dobeson;Sir
✓		Risolvere Problemi	9 h	0.5 giorni?	ven 21.04.23	ven 28.04.23	46	Michael Y. Dobeson;Sir
✓		Documentazione	20 h	1.11 giorni?	ven 28.04.23	ven 05.05.23	47	Stefano Ceschi;Michae

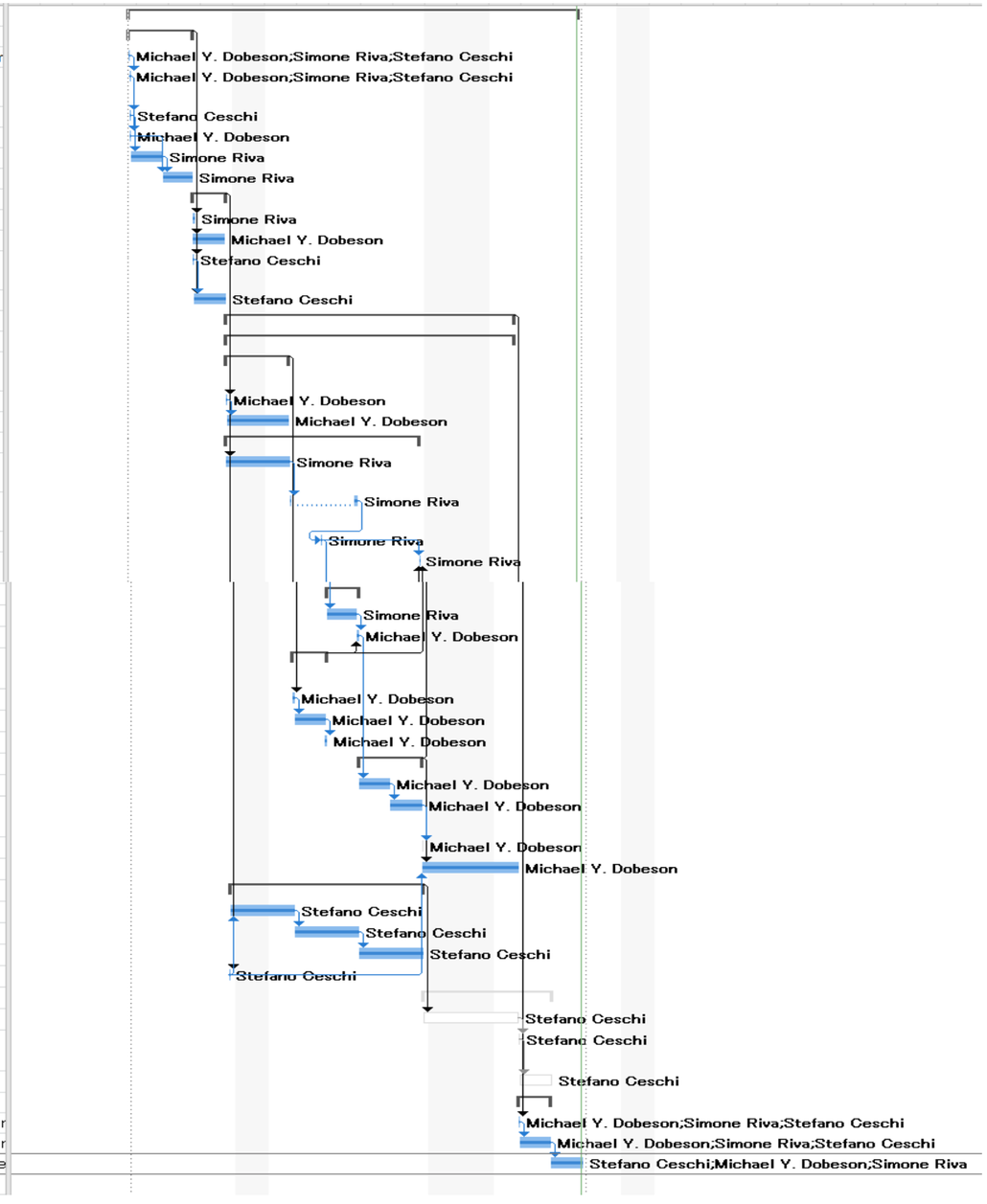


Figura 24 - Gantt Consuntivo

Al termine del progetto il Gantt è cambiato, infatti abbiamo impiegato il doppio del tempo sia per la navigazione con il controller sia per l'implementazione del multiplayer.
Inoltre abbiamo impiegato meno tempo per la realizzazione del DataBase.

Conclusioni

6.1 Sviluppi futuri

In futuro si può migliorare la grafica, applicare il cell-shading, sviluppare il sito web, creare un sistema per un LAN party e sicuramente correzioni di bug.

6.2 Considerazioni personali

Simone
Questo progetto mi ha aiutato a consolidare il lavoro in team e l'organizzazione del lavoro durante le settimane, inoltre mi ha aiutato a comprendere meglio in maniera pratica alcune funzionalità programmate e implementate in C#.
Mi ritengo soddisfatto del nostro lavoro e contento di essere riuscito nello svolgimento del progetto.

Michael
Questo progetto è la seconda volta che metto le mani su Unity, ma era un'esperienza completamente diverse essendo che stavo lavorando in un'ambiente 3D usando funzionalità Network per fare un gioco online.
Per quanto non siamo riusciti ad avere un prodotto raffinato al completo, è funzionale e sono contento di quello che è uscito e di quello che ho imparato.

Stefano
Vorrei concludere questo progetto dicendo prima di tutto che ho veramente imparato molto, soprattutto quello che riguarda il networking dei GameObjects. Questo progetto mi è servito particolarmente per capire che nessuno di noi tre è onnisciente, mi spiego meglio: siamo partiti con l'idea di fare un giocone, pieno di funzioni e con addirittura un sito web, ma ci siamo resi conto circa 3 settimane prima che non avremmo finito se avessimo continuato a quel ritmo. Io mi sono forse concentrato troppo sullo sviluppo e poco sulla documentazione, ne sono consapevole ma voglio ribadire che, a mio avviso, è anche così che si imapara.

7 Glossario

Termine	Descrizione
AI	Intelligenza Artificiale
Character Controller	È uno dei componenti di Unity e serve per aggiungere l'Input Manager ad un Game Object, ad esempio al giocatore.
ClientRpc	Dice al PC che quel metodo deve essere eseguito solo dal client
Controller	Si riferisce al joystick che si usa per giocare ai giochi su le console. Nel nostro caso abbiamo usato un controller PS5 per testare
Framerate	Sequenza di fotogrammi
Game Object	Gli oggetti in Unity.
Lag	Tempi di scambio dati troppo lunghi risetto a quelli desiderati
Leaderboard	Classifica
NavMesh Agent	Questo componente è collegato a un personaggio mobile nel gioco per consentirgli di navigare nella scena utilizzando NavMesh, un componente di Unity per simulare un'AI.
NetworkBehavior	È la classe base di Netcode (da cui ogni script eredita)
NetworkVariable	Variabile che è sincronizzata nella rete Netcode

Respawn Point	I punti predefiniti in cui vieni rigenerato.
ServerRpc	Dice al PC che quel metodo deve essere eseguito solo dal server / host
Vector3	Il tipo di dati Vector3 rappresenta un vettore nello spazio 3D, generalmente utilizzato come punto nello spazio 3D o le dimensioni di un prisma rettangolare.

8 Bibliografia

8.1 Sitografia

https://lucid.app 28.04.2023
https://docs.unity3d.com 28.04.2023
https://stackoverflow.com 28.04.2023
https://learn.microsoft.com/en-us/docs/ 28.04.2023
https://api.ipify.org/ 28.04.2023
https://mockflow.com/ 28.04.2023
https://www.youtube.com/watch?v=Y3WNwl1ObC8 14.04.2023
https://github.com/nullkal/UniSense 14.04.2023

9 Allegati

https://github.com/SimoneRivaSAMT/TAG 05.05.2023
https://trello.com/b/T5sMZ7dY/development 05.05.2023