

Colorful Songs

Titolo del progetto: Colorful Songs
Alunno/a: Lorenzo Berther, Sebastiano De Bertoldi, Simone Demarchi, Kamil Siddiqui
Classe: Info I3A
Anno scolastico: 2024/2025
Docente responsabile: Michel Palucci

1	Introduzione	4
1.1	Informazioni sul progetto	4
1.2	Abstract	4
1.3	Scopo	4
2	Analisi	5
2.1	Analisi del dominio	5
2.2	Analisi e specifica dei requisiti	5
2.2.1	Spiegazione elementi tabella dei requisiti:.....	8
2.3	Use case	9
2.4	Pianificazione	10
2.4.1	Primo Sprint	11
2.4.2	Secondo Sprint	11
2.4.3	Terzo Sprint.....	11
2.4.4	Quarto Sprint.....	11
2.4.5	Quinto Sprint	11
2.5	Analisi dei mezzi.....	12
2.5.1	Software	12
2.5.2	Librerie	12
2.5.3	Hardware	12
3	Progettazione	12
3.1	Design dell'architettura del sistema	12
3.2	Design dei dati e database.....	13
3.3	Design delle interfacce	14
3.3.1	Design Login	14
3.3.2	Design Admin Page	15
3.3.3	Design Leaderboard	15
3.4	Design della mappa	16
3.5	Design degli ostacoli	16
3.5.1	Percorso con pedane finte	16
3.5.2	Percorso con pedane invisibili e soluzione sul muro	17
3.5.3	Portali finti	17
3.5.4	Percorso con cannoni	18
3.5.5	Percorso con parkour.....	18
3.6	Design dei nemici	19
3.6.1	Tamburo.....	19
3.6.2	Trombetta.....	19
3.7	Generazione dei Dungeon	20
3.8	Design procedurale	20
3.8.1	ActivityDiagramPrincipale	20
3.8.2	AD_Gioco.....	20
3.8.3	AD_LeaderBoard	21
3.8.4	AD_SezioneAmici	21
4	Implementazione	22
4.1	Classe TerrainGenerator.....	22
4.2	Classe ParkourGenerator	29
4.3	Classe FakeFloorGenerator.....	31
4.4	Classe WallPedanaGenerator.....	33
4.5	Classe CannoBall.....	35
4.6	Classe CannoGenerator	35
4.7	Classe PortalGenerator.....	39
4.8	Classe Score	42
4.9	Classe API_CALL.....	44
4.10	Telecamera.....	48
4.10.1	Camera Controller.....	48
4.11	Audio.....	49
4.11.1	Audio Manager.....	50
4.12	Personaggio.....	52
4.12.1	Componenti.....	52

4.12.2	Player Movement	53
4.12.3	Player Collision	57
4.12.4	HealthManager	59
4.13	Nemici.....	62
4.13.1	Tamburo.....	62
4.13.2	Trombetta.....	65
4.14	Global Volume (Effetti Visivi)	67
4.15	Leaderboard	69
4.15.1	Funzione index.....	69
4.15.2	Funzione radioFilter	70
4.15.3	Funzione searchFilter	71
4.15.4	Funzione showFriend.....	72
4.15.5	Funzione getData.....	73
4.15.6	Funzione getDataByFriendId	73
4.15.7	Funzione getDataByUsername.....	74
4.15.8	Funzione newFriend	74
4.15.9	Funzione friendRequestWithFriendId	74
4.15.10	Struttura della tabella	75
4.16	API	76
4.16.1	UserController	76
4.16.2	UserGateway	77
4.16.3	Index	78
5	Test.....	79
5.1	Protocollo di test.....	79
5.2	Risultati test.....	92
5.3	Mancanze/limitazioni conosciute.....	92
6	Consuntivo.....	93
6.1	Primo Sprint.....	93
6.2	Secondo Sprint.....	94
6.3	Terzo Sprint.....	94
6.4	Quarto Sprint	95
6.5	Quinto Sprint	95
6.6	Sesto Sprint.....	95
6.7	Confronto.....	96
7	Conclusioni	96
7.1	Sviluppi futuri.....	96
7.2	Considerazioni personali.....	97
7.2.1	Lorenzo	97
7.2.2	Sebastiano	97
7.2.3	Simone	97
7.2.4	Kamil	98
8	Glossario	99
9	Indice delle figure	100
10	Bibliografia	101
10.1	Sitografia.....	101
11	Allegati	101

1 Introduzione

1.1 Informazioni sul progetto

- Allievi: Lorenzo Berther, Sebastiano De Bertoldi, Simone Demarchi, Kamil Siddiqui
- Docente responsabile: Michel Palucci
- Scuola: Centro Professionale Tecnico Trevano, sezione informatica
- Data di inizio: 29.01.2025
- Data di Termine: 28.05.2025

1.2 Abstract

Video game companies nowadays release the same popular franchises year after year with reiterations; they bring little new innovation. The one constant has been change, as prices continue to rise with each new title. It's because of this that indie studios have started gaining more traction; the refreshing, creative alternatives break away from the usual.

Created by four passionate students, the indie game Colorful Songs bears the mark of '80s design; this group puts unique gameplay into one package with this kind of game: a combination of parkour, fighting, speedrunning, and obstacle navigation. Always keeping it fast and appealing, these form the very heart of the gameplay.

An innovative aesthetic journey that involves a musician-composer who needs to find and recover the lost broken melody, which he intends to make even more exquisite previously is what the game is set to accomplish. Such heavy narration is interwoven with gameplay, relating that every level stands for a depiction of the composer's emotional as well as creative stride.

Another one of the features of Colorful Songs is that it carries a strong nostalgic tone with it. The game takes players back to the 1980s in terms of graphics as it holds bright colors, neon lights, and pixel art visuals that are closely associated with the classic games of the era. The sound makes a main contribution- not as a soundtrack but as a vital part of the story making synthesize type of sounds to take the player into such an immense atmosphere.

This project proves that a small team can create a great gaming experience — one that both retro lovers and players seeking new, different gameplay will enjoy. Colorful Songs is more than just a tribute to the '80s; it's an invitation to find again the simple, heartfelt joy of storytelling, with dynam-ic, multi-faceted gameplay to back it up.

1.3 Scopo

Lo scopo didattico di questo progetto è sviluppare le competenze nella collaborazione di gruppo utilizzando le risorse scolastiche, acquisire familiarità con Unity e comprenderne le principali funzionalità e migliorare le nostre competenze tecniche.

Lo scopo operativo è la creazione di un gioco per PC di genere Roguelike/Platform, in cui l'obiettivo principale è superare i 5 livelli più velocemente possibile. Il gioco è ambientato negli anni '80, sia per aderire al tema di un contest, sia per garantire la compatibilità con computer meno recenti. Questa scelta ha facilitato la gestione del tempo, grazie alla grafica semplice e minimale.

2 Analisi

2.1 Analisi del dominio

Il videogioco Colorful Songs è destinato a qualcuno fascia d'età. Non ha nessuno requisito specifico e sarà giocabile da PC con tastiera o con un controller, è disponibile la funzione di Login/Creazioni account se si vuole salvare il proprio punteggio e accedere alla classifica globale. Possedere un account permette anche di aggiungere altri utenti fra i propri amici e visualizzare una classifica solo fra amici, dalla classifica globale sarà possibile inviare richieste di amicizia.

L'obiettivo principale è quello di creare un videogioco platform/ roguelike in Unity, dove il giocatore deve superare 5 livelli per completare il gioco. I livelli sono composti da una mappa generata casualmente con 2 ostacoli casuali e 1 nemico casuale. Lo scopo sarebbe di completare questi 5 livelli nel minor tempo possibile per competere con altri giocatori.

2.2 Analisi e specifica dei requisiti

ID: REQ-001	
Nome	Gestione utente
Priorità	1
Versione	1.0
Sotto requisiti	
001	Creare Database 'ColorfulSongs'
002	Creare le tabelle 'User' e 'Friend'

ID: REQ-002	
Nome	Effettuare un login in game
Priorità	1
Versione	1.0
Note	Dipende dal requisito REQ-001 (Gestione Utente)
Sotto requisiti	
001	Deve spuntare una tastiera virtuale (per controller)

Colorful Songs

ID: REQ-003	
Nome	Effettuare una registrazione in game
Priorità ID: REQ-003	1
Versione	1.0
Note	Dipende dal requisito REQ-001 (Gestione Utente)
Sotto requisiti	
001	Controlli per non creare utenti identici a quelli già esistenti
002	Deve spuntare una tastiera virtuale (per controller)

ID: REQ-004	
Nome	Leaderboard web
Priorità	1
Versione	1.0
Note	Dipende dal requisito REQ-001 (Gestione utente)
Sotto requisiti	
001	Creare la tabella 'Leaderboard'
002	Creare una pagina web che prenda gli highscore degli utenti
003	Nel gioco deve esserci un collegamento alla pagina

ID: REQ-005	
Nome	Login nella Leaderboard Web
Priorità	1
Versione	1.0
Note	Dipende dal requisito REQ-001 (Gestione utente)
Sotto requisiti	
001	Creare una pagina di login

Colorful Songs

ID: REQ-006	
Nome	Creare gli ostacoli
Priorità	1
Versione	1.0
Sotto requisiti	
001	Creare il " Percorso con pedane finte "
002	Creare il " Percorso con pedana invisibile e soluzione sul muro "
003	Creare i " Portali finti "
004	Creare il " Percorso con cannoni "
005	Creare il " Percorso con parkour "

ID: REQ-007	
Nome	Creazione del dungeon casuale
Priorità	1
Versione	1.0
Sotto requisiti	
001	Generare le stanze e gli ostacoli.
002	Creare id per il dungeon
003	Generare casualmente gli NPC.

ID: REQ-008	
Nome	Salvataggio highscore
Priorità	1
Versione	1.0

ID: REQ-009	
Nome	Tutorial testuale
Priorità	1
Versione	1.0
Note	Creare una pagina con la descrizione dei comandi

ID: REQ-010	
Nome	Aggiungere amici
Priorità	1
Versione	1.0
Note	Dipende dal requisito REQ-001 (Creazione DB)

ID: REQ-011	
Nome	Giocare con tastiera e/o controller
Priorità	1
Versione	1.0
Note	In caso di controller, generare una tastiera virtuale, per soddisfare requisito REQ-002 e REQ-003

ID: REQ-012	
Nome	API
Priorità	1
Versione	1.0
Note	API per permettere di accedere al database da Unity

2.2.1 Spiegazione elementi tabella dei requisiti:

ID: identificativo univoco del requisito

Nome: breve descrizione del requisito

Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

Versione: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata. Sulla documentazione apparirà solamente l'ultima versione, mentre le vecchie dovranno essere inserite nei diari.

Note: eventuali osservazioni importanti o riferimenti ad altri requisiti.

Sotto requisiti: elementi che compongono il requisito.

2.3 Use case

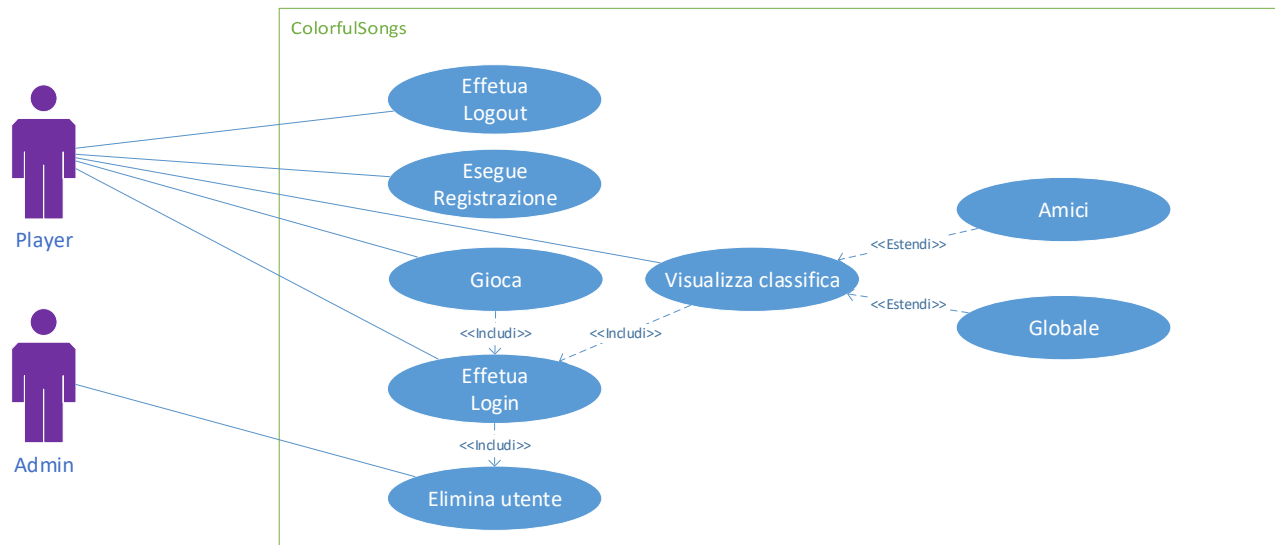


Figura 1: Use Case

Il giocatore può:

- Effettuare un logout (ovviamente se prima ha eseguito un login)
- Registrarsi, in caso non abbia ancora un account
- Eseguire un login, dopo il quale può giocare o visualizzare la classifica, globale o degli amici

L'admin può:

- Eliminare gli utenti

2.4 Pianificazione

Metodo: Scrumban (Agile)

Nome attività	Durata	Inizio	Fine	Nomi risorse
▲ ColorfulSongs	113.5 h	mer 29.01.25	mer 28.05.25	Full Team
▲ Primo sprint	15 h	mer 29.01.25	mer 12.02.25	Full Team
Analisi requisiti	2 h	mer 29.01.25	mer 29.01.25	Full Team
Meeting di kickoff	1 h	mer 29.01.25	mer 29.01.25	Full Team
Kanban	1 h	mer 29.01.25	mer 29.01.25	Full Team
Gantt	2 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
Schema ER	1 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
Creazione database	0.5 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
Use Case	0.5 h	mer 05.02.25	mer 05.02.25	Simone Demarchi;Kamil Siddiqui;Lorenzo Berther
Activity Diagram	2 h	mer 12.02.25	mer 12.02.25	Kamil Siddiqui;Lorenzo Berther
Pianificare la struttura degli ostacoli	1 h	mer 05.02.25	mer 05.02.25	Kamil Siddiqui;Simone Demarchi
Movimento base del player	2 h	mer 12.02.25	mer 12.02.25	Simone Demarchi
Configurazione env PHP	3 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
▲ Mock up interfacce	8.08 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi;Simone Demarchi
GUI Login	2 h	mer 05.02.25	mer 05.02.25	Simone Demarchi
GUI leaderboard	2 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
Creare la pagina per il login web	2 h	mer 12.02.25	mer 12.02.25	Sebastiano De Bertoldi
▲ Secondo sprint	14 h	mer 19.02.25	mer 05.03.25	Full Team
Game menu opzioni	8 h	mer 19.02.25	mer 19.02.25	Simone Demarchi;Lorenzo Berther
▲ Audio	2 h	mer 19.02.25	mer 19.02.25	Simone Demarchi
Implementazione SFX	2 h	mer 19.02.25	mer 19.02.25	Simone Demarchi
Implementazione soundtrack	2 h	mer 19.02.25	mer 19.02.25	Simone Demarchi
Filtri leaderboard	6 h	mer 19.02.25	mer 19.02.25	Sebastiano De Bertoldi
Generazione NPC statici	6 h	mer 19.02.25	mer 19.02.25	Lorenzo Berther;Simone Demarchi
Preparazione demo tecnica	3 h	mer 19.02.25	mer 19.02.25	Lorenzo Berther;Simone Demarchi
Studio API PHP e endpoint	14 h	mer 19.02.25	mer 05.03.25	Sebastiano De Bertoldi
▲ Terzo sprint	12 h	mer 12.03.25	mer 19.03.25	Full Team
Creazione ostacoli	12 h	mer 12.03.25	mer 19.03.25	Simone Demarchi;Kamil Siddiqui;Lorenzo Berther
Generare nelle stanze gli ostacoli	6 h	mer 12.03.25	mer 12.03.25	Kamil Siddiqui
Generare la mappa casualmente	6 h	mer 12.03.25	mer 12.03.25	Kamil Siddiqui
Interazione con L'API in game per account	3 h	mer 12.03.25	mer 12.03.25	Lorenzo Berther
Movimento nemici	6 h	mer 12.03.25	mer 12.03.25	Lorenzo Berther;Simone Demarchi
Gestione Amicizie	12 h	mer 12.03.25	mer 19.03.25	Sebastiano De Bertoldi
▲ Quarto sprint	5.83 h	mer 09.04.25	mer 09.04.25	Full Team
Filtri leaderboard in game	6 h	mer 09.04.25	mer 09.04.25	Kamil Siddiqui
Tutorial Testuale	3 h	mer 09.04.25	mer 09.04.25	Simone Demarchi
Virtual Keyboard in game	6 h	mer 09.04.25	mer 09.04.25	Lorenzo Berther
Score Manager	3 h	mer 09.04.25	mer 09.04.25	Sebastiano De Bertoldi
Env Prod PHP	6 h	mer 09.04.25	mer 09.04.25	Sebastiano De Bertoldi
▲ Quinto sprint	16 h	mer 14.05.25	mer 28.05.25	Full Team
Backend testing	16 h	mer 14.05.25	mer 28.05.25	Kamil Siddiqui;Sebastiano De Bertoldi
Game testing	16 h	mer 14.05.25	mer 28.05.25	Lorenzo Berther;Simone Demarchi

Figura 2: Gantt preventivo

2.4.1 Primo Sprint

Durante il primo sprint abbiamo eseguito l'analisi dei requisiti, seguita dal meeting di kickoff, che ha segnato l'inizio effettivo del progetto. Ci siamo occupati della pianificazione e della creazione del Kanban, del Gantt preventivo, dei vari schemi ERI, del database, dello Use Case Diagram e degli Activity Diagram. Per quanto riguarda Unity, abbiamo pianificato la struttura degli ostacoli che il player incontrerà durante il gioco e abbiamo creato uno script di base per il movimento del personaggio. Abbiamo inoltre pianificato la configurazione dell'ambiente di sviluppo per PHP, per successivamente realizzare il mockup delle varie interfacce del progetto, come la GUI di login, la GUI della leaderboard e la pagina web di login.

2.4.2 Secondo Sprint

Per Unity, abbiamo pianificato la creazione del menu principale con le opzioni di gioco, l'implementazione dell'audio (soundtrack e suoni per il salto). In generale, abbiamo pianificato i filtri per la leaderboard, la generazione di nemici statici, la preparazione della demo tecnica, e abbiamo dedicato del tempo allo studio del funzionamento delle API PHP e dei relativi endpoint.

2.4.3 Terzo Sprint

Durante il terzo sprint abbiamo previsto di concentrarci maggiormente su Unity, con la creazione e la generazione automatica degli ostacoli nelle stanze, la generazione randomica della mappa e il movimento dei nemici, per renderli dinamici e non più statici. Per quanto riguarda le API, abbiamo pianificato l'integrazione con il gioco per la gestione degli account, mentre per PHP ci siamo concentrati sulla gestione delle amicizie.

2.4.4 Quarto Sprint

Durante questo sprint abbiamo pianificato l'implementazione dei filtri per la leaderboard in gioco, la creazione di un tutorial testuale per i nuovi giocatori e l'introduzione di una tastiera virtuale per chi gioca con il controller, così da permettere l'inserimento di testo. In ambito PHP, abbiamo previsto la creazione dello Score Manager e dell'ambiente di sviluppo produttivo.

2.4.5 Quinto Sprint

Durante l'ultimo sprint abbiamo pianificato i test del backend e del gioco, oltre ad apportare alcune modifiche e rifiniture alla documentazione.

2.5 Analisi dei mezzi

2.5.1 Software

- Unity 6000.0.35f1
- MS-Office
- MS-Project
- GitHub (Desktop)
- PHP Storm 2024.1.1
- MySQL Workbench 8.0
- Bootstrap 5.2.3
- Figma web
- Postman 11.44.0
- PHP 8.2.12

2.5.2 Librerie

Illuminate/database 12.5.0
Monolog/monolog 3.9.0
VLukas/phpdotenv 2.6.9

2.5.3 Hardware

- PC scolastico:
 - Sistema operativo: Windows
 - CPU: 13th Gen Intel(R) Core(TM) i7-13700
 - Scheda video: NVIDIA T400 4GB
 - Memoria RAM: 32.0 GB DDR5

Server scolastico dove mettere il progetto e il Database

3 Progettazione

3.1 Design dell'architettura del sistema

3.2 Design dei dati e database

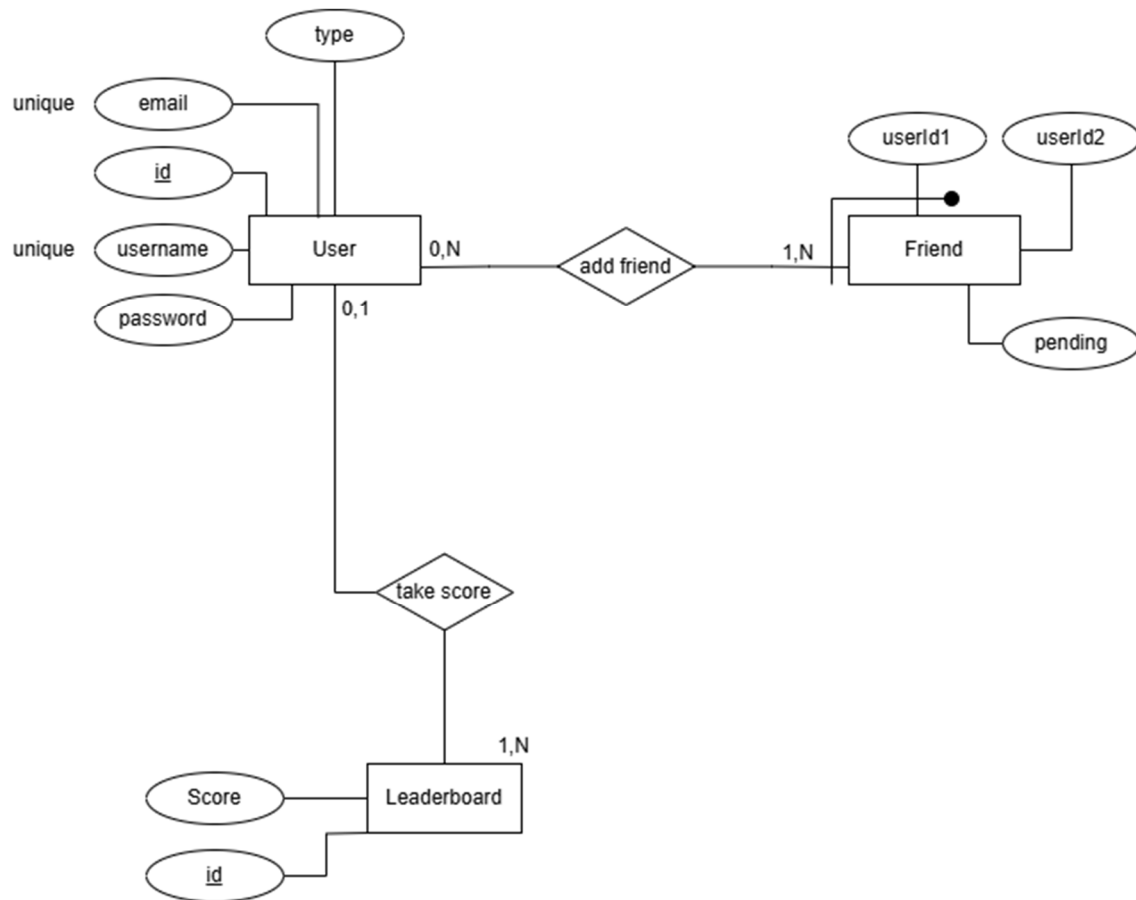


Figura 3: Schema E-R

User	
PK	<u>id int NOT NULL AUTO INCREMENT</u>
	username varchar(64) UNIQUE NOT NULL
	password varchar(256) NOT NULL
	email varchar(256) UNIQUE NOT NULL
	type varchar(10) NOT NULL

Leaderboard	
PK	<u>id int NOT NULL AUTO INCREMENT</u>
FK	user_id int
	score time

Friend	
PK, FK	<u>user_id1 int NOT NULL</u>
PK, FK	<u>user_id2 int NOT NULL</u>
	pending bool

Figura 4: Schema logico

Il database ha come tabella centrale la tabella user, perché è la tabella da cui vengono prese più informazioni. Infatti sia per mostrare i dati nella leaderboard che per la gestione delle richieste di amicizia bisogna per forza passare dalla tabella user, perché nel caso della visualizzazione dei dati nella leaderboard serve vedere il nome dell'utente, mentre nel caso della gestione degli amici serve avere l'id dell'utente. Le relazioni sono $0,1 - 1,N$ per la relazione tra la tabella user e la tabella leaderboard, questo perché un utente può avere o nessun punteggio oppure un punteggio solo che viene aggiornato, e la leaderboard deve avere un minimo di 1 punteggio e può avere come massimo un numero N di punteggi. Mentre per la relazione tra le tabelle user e friend la relazione è $0,N - 1,N$ questo perché un utente può avere da un minimo di 0 amici a un massimo N di amici, invece l'amico può avere da un minimo di 1 amico, questo perché per essere un amico deve avere almeno un amicizia, e come massimo può avere un numero N di amicizie.

3.3 Design delle interfacce

3.3.1 Design Login

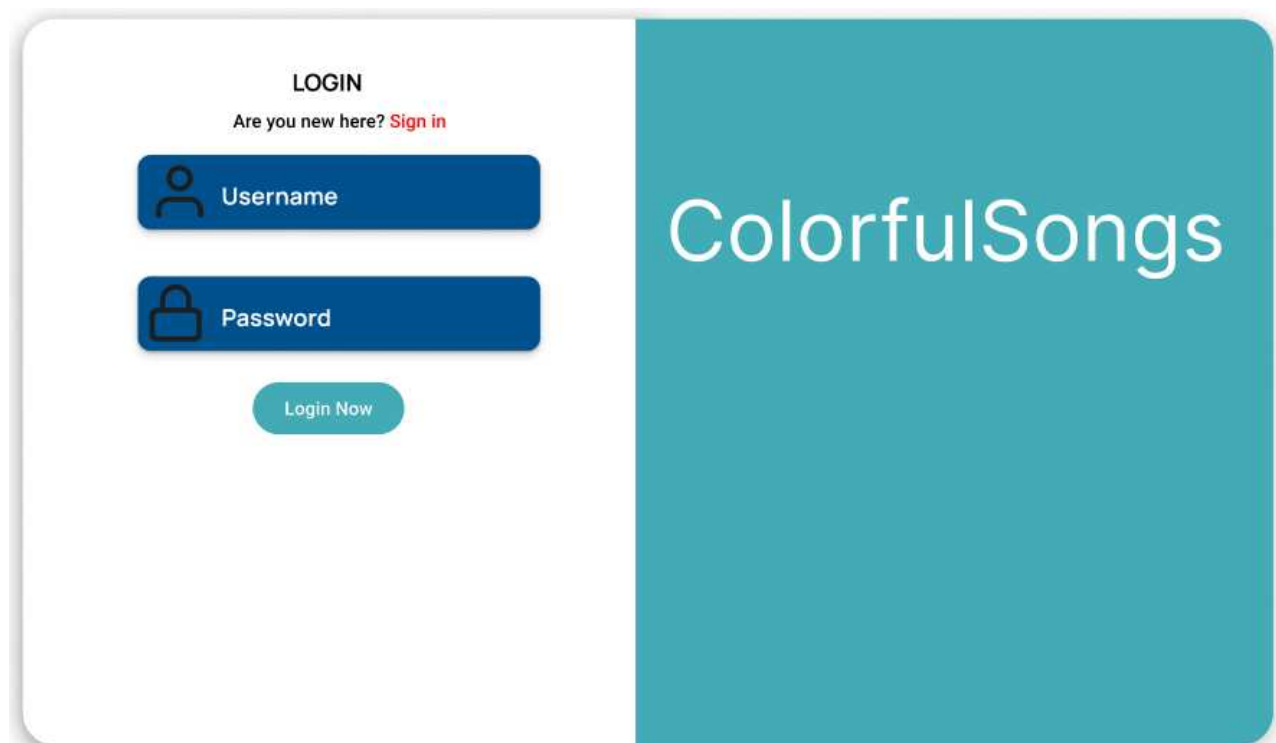


Figura 5: Design interfaccia di Login

Questa schermata, rappresenta l'interfaccia Web per la pagina di Login, permette di accedere alla leaderboard.

3.3.2 Design Admin Page

Leaderboard		
Username	High Score	Delete User

Figura 6: Design interfaccia di admin

Questa interfaccia rappresenta la gestione degli utenti, è solo per l'admin e gli permette di gestire gli utenti, nella colonna delete user c'è un pulsante delete che permette di eliminare l'utente.

3.3.3 Design Leaderboard

Filter <input type="radio"/> Friends <input checked="" type="radio"/> Global Maps <input type="text" value="Map Code"/> <input type="button" value="Q"/>	Leaderboard		
	Username	High Score	Add Friend

Figura 7: Design interfaccia leaderboard

Questa interfaccia rappresenta la leaderboard, questa permette di consultare i dati dei giocatori che hanno partecipato al gioco. Questi dati sono: username, High score ottenuto e un pulsante add friend, che permette di aggiungere quel giocatore alla lista amici (se l'altro giocatore accetta l'amicizia).

3.4 Design della mappa

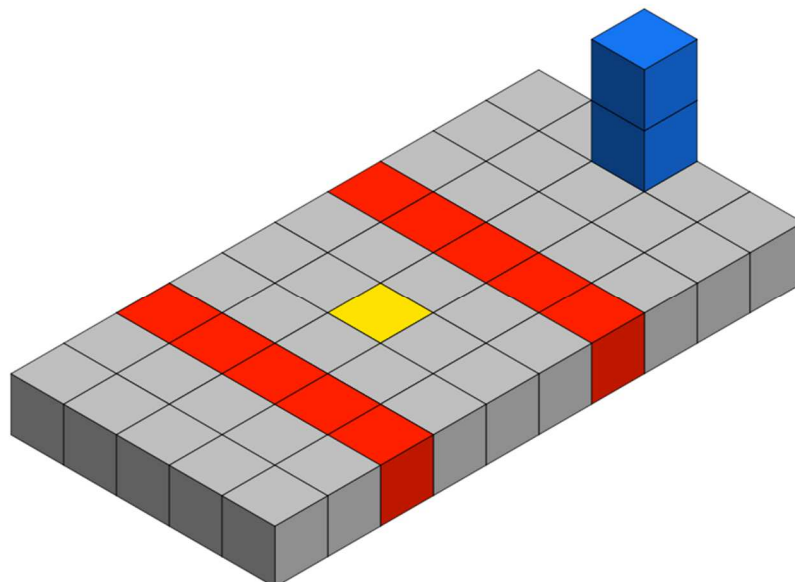


Figura 8: Design di una stanza

Questo design rappresenta la mappa di gioco, le righe rosse sono dove ci saranno gli ostacoli generati casualmente, invece il punto giallo è dove si genera il nemico. Infine i due blocchetti blu sono il portale per passare al livello successivo.

3.5 Design degli ostacoli

3.5.1 Percorso con pedane finte

In questo percorso il giocatore dovrà percorrere un percorso, dove alcune pedane saranno finte. Se l'utente ci passerà sopra, cadrà, verrà riportato all'inizio della stanza e la pedana sparirà, lasciando un buco. Questo percorso occuperebbe uno spazio di colore rosso sulla mappa. Il percorso da seguire avrà una texture diversa.

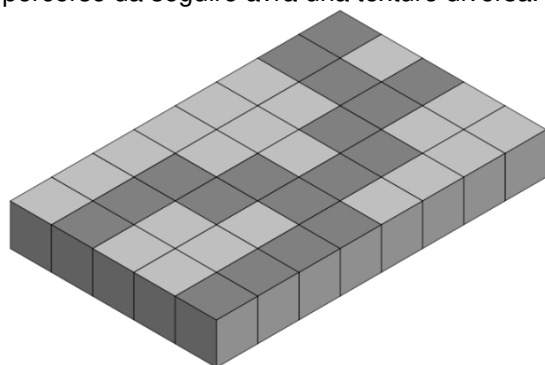


Figura 9: Design del percorso con pedane finte

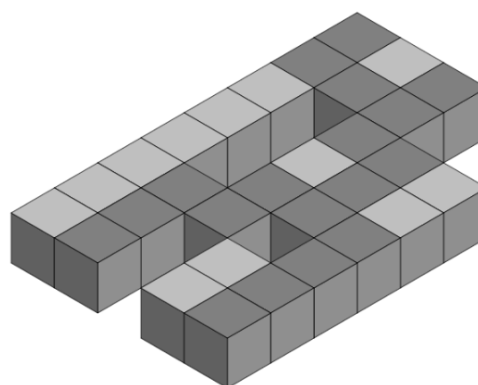


Figura 10: Design della pedana (Come apparirebbe all'utente dopo qualche tentativo)

3.5.2 Percorso con pedane invisibili e soluzione sul muro

Questo percorso è simile all'ostacolo "Percorso con pedane finte", ma a sua differenza le pedane sono invisibili e il percorso da percorrere è disegnato sul muro.

Questo percorso occuperebbe uno spazio di colore rosso sulla mappa.

Ecco un esempio di come un'ipotetica pedana potrebbe essere:

La parte rossa e bianca hanno solo i bordi visibili. Sul muro grigio vediamo la strada corretta da percorrere (bianca).

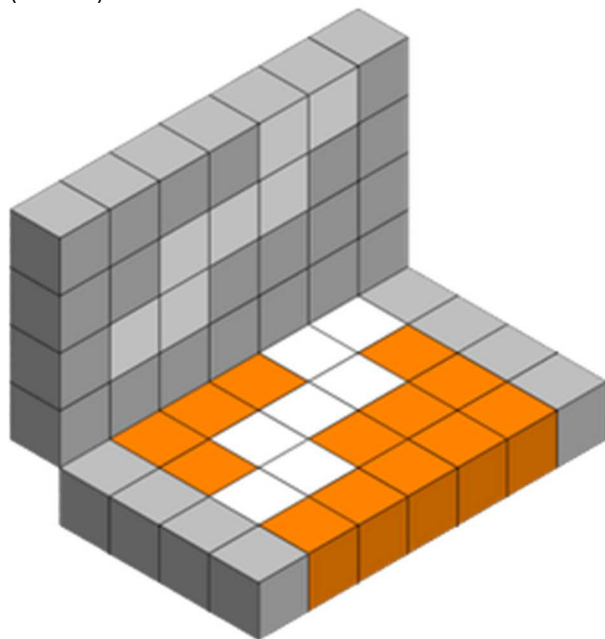


Figura 11: Design dell'ostacolo "Percorso con pedana invisibile e soluzione sul muro"

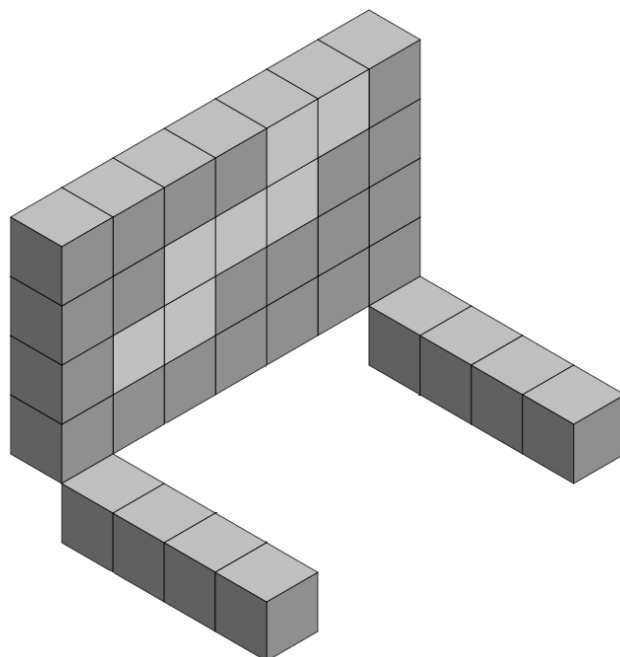


Figura 12: Design dell'ostacolo "Percorso con pedana invisibile e soluzione sul muro" visto dall'utente

3.5.3 Portali finti

Nella stanza saranno presenti dei portali, solo uno di questi porterà il player dalla parte opposta del percorso. Se l'utente prende il portale sbagliato, il portale si distruggerà e toglierà un cuore, per riconoscere il portale giusto bisognerà mettersi davanti il portale e ascoltare l'audio o vedendo dalla texture diversa, in caso di problemi d'udito.

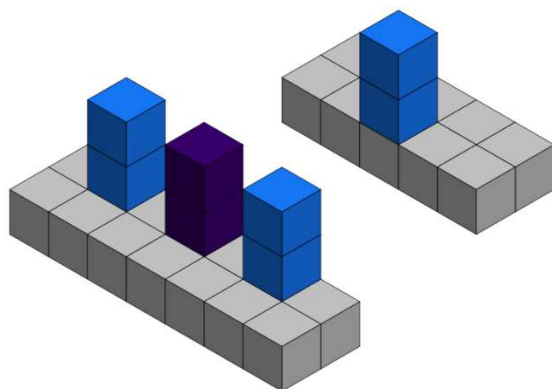


Figura 14: Design portali finti

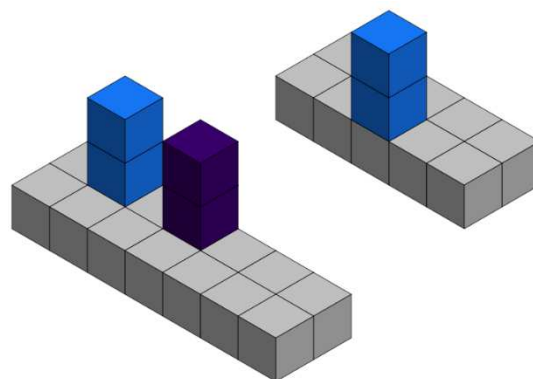


Figura 13: Design portali finti (dopo errore)

3.5.4 Percorso con cannoni

Questo percorso avrà sui muri dei cannoni che spariranno a intervalli regolari, alternandosi tra i cannoni in posizione pari con quelli in posizione dispari, se si verrà colpiti da cannoni si perderà la partita

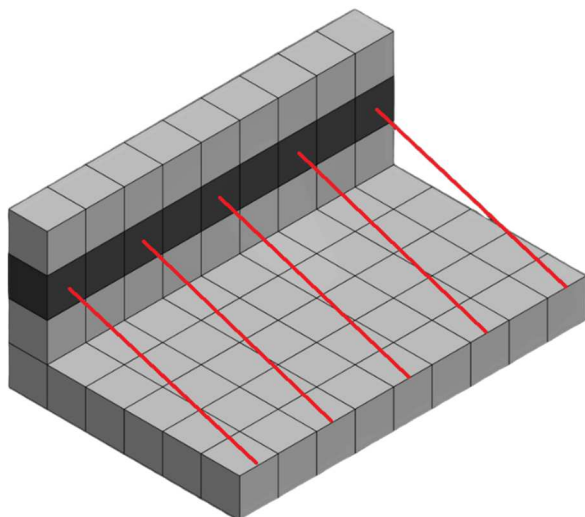


Figura 16: Cannoni pari

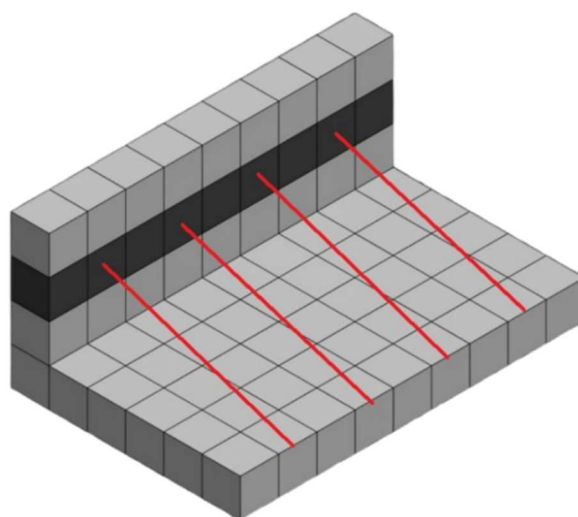


Figura 15: cannoni dispari

3.5.5 Percorso con parkour

Questo percorso sarà un semplice parkour, l'utente dovrà superarlo usando la meccanica del salto. Se cadrà perderà la partita.

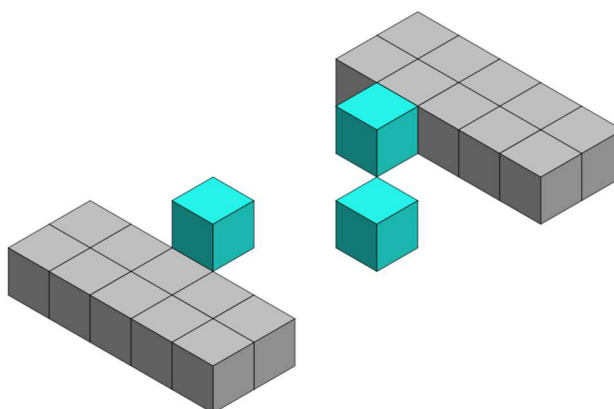


Figura 17: Parkour

3.6 Design dei nemici

3.6.1 Tamburo

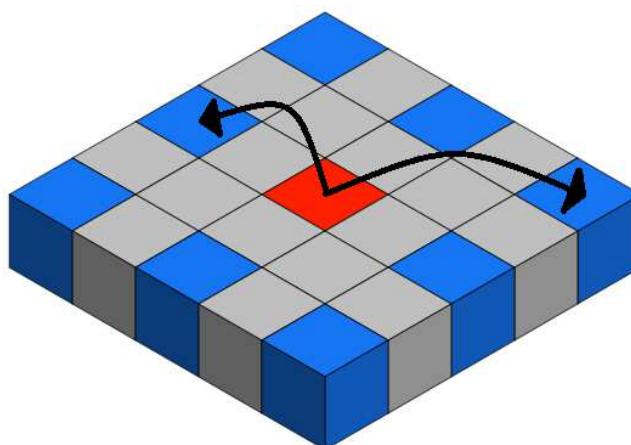


Figura 18: Pattern Tamburo

Il tamburo è un nemico che si può incontrare durante il gioco. Non ha un pattern specifico, ma viene generato casualmente. Il tamburo appare sulla pedana rossa e può saltare sulle piattaforme blu, impedendo al giocatore di attraversare alcuni blocchi, poiché infligge 1 punto di danno se il giocatore lo tocca.

3.6.2 Trombetta

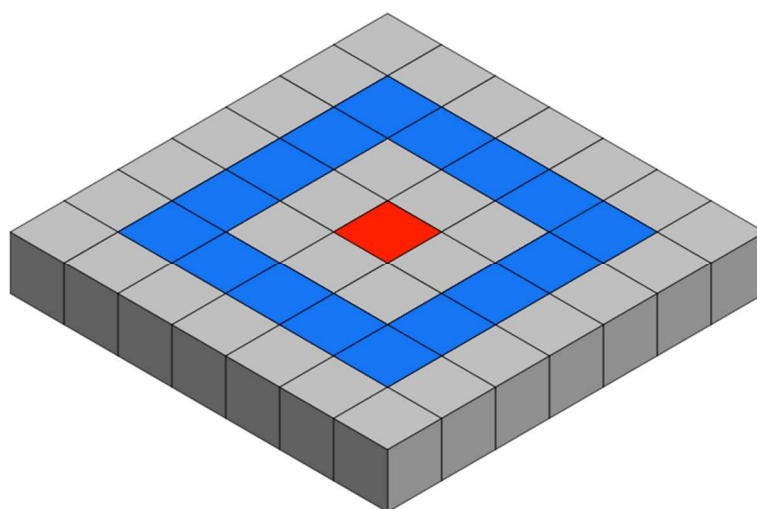


Figura 19: Trombetta

La trombetta è un nemico che si può incontrare durante il gioco. Il suo pattern è sparare a 360° senza mai fermarsi, creando così un “turbine” di proiettili. La trombetta appare sulla pedana rossa e può sparare fino alle pedane blu, aumentando così la difficoltà di passaggio al giocatore, che dovrà prestare attenzione ai proiettili, infligge 1 punto di danno se il giocatore tocca un proiettile.

3.7 Generazione dei Dungeon

I dungeon sono composti da cinque stanze, ognuna contenenti due ostacoli generati casualmente tra quelli descritti nel punto [3.4](#).

All'interno delle stanze è anche presente un nemico che viene generato anch'esso casualmente tra i nemici descritti nel punto [3.5](#).

3.8 Design procedurale

3.8.1 ActivityDiagramPrincipale

Vedere allegato "7_Allegati\ActivityDiagram\ActivityDiagramPrincipale.png" per dettaglio.

L'utente apre il gioco. Il sistema verifica se l'utente possiede già un account. Se l'utente ha un account, viene indirizzato alla pagina di accesso. Se non ha un account, viene indirizzato alla pagina di registrazione. Nel caso di accesso, l'utente inserisce le credenziali e il sistema verifica la validità dei dati. Se i dati non sono validi, viene mostrato un errore generico. Se i dati sono validi, il sistema controlla se la password inserita è corretta. Se la password non è valida, viene mostrato un errore con il messaggio "Password non valida". Se la password è corretta, l'accesso viene confermato con il messaggio "Login effettuato".

Nel caso di registrazione, il sistema controlla se lo username è già presente nel database. Se lo username esiste, viene mostrato un errore con il messaggio "Username già in uso". Se lo username non esiste, il sistema tenta di creare un nuovo utente. Se la creazione dell'utente fallisce, viene mostrato un errore con il messaggio "Utente non creato, riprovare". Se la creazione va a buon fine, viene effettuato il login con il messaggio "Login effettuato".

Dopo il login l'utente accede alla schermata principale del gioco. Da qui può selezionare diverse opzioni. Se seleziona "Impostazioni", viene aperta la pagina delle impostazioni e l'utente può modificarle e salvarle. Se seleziona "Amici", viene aperta la sezione dedicata agli amici. Se seleziona "Leaderboard", viene aperta una pagina web che mostra la classifica. Se seleziona "Gioca", viene avviata una nuova partita. Se seleziona "Chiudi il gioco", l'applicazione viene chiusa.

3.8.2 AD_Gioco

Vedere allegato "\7_Allegati\ActivityDiagram\AD_Gioco.png" per dettaglio.

All'inizio della partita, vengono generate casualmente cinque stanze. All'interno di ciascuna stanza, vengono generati casualmente due enigmi. Il giocatore viene quindi posizionato all'inizio della prima stanza, in un'area priva di oggetti.

Il giocatore deve affrontare e cercare di superare gli enigmi presenti. Se non riesce a superarli e si tratta di un enigma fatale, viene mostrata una schermata di sconfitta insieme alla visualizzazione del punteggio ottenuto. Se invece supera gli enigmi e l'enigma non è fatale, deve affrontare gli NPC presenti nella stanza. Se riesce a sconfiggerli e a raccogliere l'oggetto necessario, può raggiungere la porta della stanza ed uscire. Il sistema verifica a questo punto se il giocatore si trova nella quinta e ultima stanza. Se è così, viene mostrata la schermata di vittoria con la visualizzazione del punteggio. Se non è ancora l'ultima stanza, il ciclo prosegue con la stanza successiva.

Alla fine della partita (in caso di vittoria), viene controllato se il punteggio ottenuto è superiore al punteggio personale più alto (high score). Se il nuovo punteggio è maggiore, viene mostrato un messaggio che indica "Nuovo HighScore" e il sistema procede a sovrascrivere il punteggio nel database.

3.8.3 AD_LeaderBoard

Vedere allegato “\7_Allegati\ActivityDiagram\AD_LeaderBoard.png” per dettaglio.

La procedura inizia con l'accesso alla web page. Il sistema verifica se la pagina è stata raggiunta direttamente dal videogioco. Se sì, viene effettuato il login automaticamente utilizzando i dati passati dal videogioco. In alternativa, l'utente può effettuare il login tramite un form presente sulla pagina web.

Se i dati inseriti non sono validi, viene mostrato un messaggio di errore. Se i dati sono validi, il sistema mostra il messaggio: “Login effettuato con successo”.

A questo punto, il sistema verifica se l'utente è un amministratore (user di tipo “admin”). Se l'utente non è un admin, può visualizzare semplicemente la leaderboard.

Se invece è un admin, gli viene concessa la possibilità di accedere a una pagina con la lista di tutti gli utenti registrati. Da lì può decidere se procedere con l'eliminazione di uno di essi. Se viene selezionato un utente da eliminare, il sistema lo rimuove dal database e visualizza il messaggio: “Eliminazione effettuata con successo”.

3.8.4 AD_SezioneAmici

Vedere allegato “\7_Allegati\ActivityDiagram\AD_SezioneAmici.png” per dettaglio.

Quando l'utente accede alla sezione Amici, ha a disposizione diverse opzioni per gestire le relazioni con gli altri giocatori.

Aggiunta di un amico:

L'utente può decidere di aggiungere un nuovo amico inserendo un codice. Il sistema verifica se il codice è valido.

- Se il codice è corretto, viene inviata una richiesta di amicizia e il database viene aggiornato. L'utente riceve il messaggio: “Richiesta mandata con successo”.
- Se il codice non è valido, viene mostrato il messaggio: “Codice invalido”.

Rimozione di un amico:

L'utente può anche scegliere di rimuovere un amico. In questo caso, il sistema aggiorna il database eliminando il legame tra i due giocatori.

Alla fine dell'operazione, viene mostrato il messaggio: “Amico rimosso con successo”.

Gestione delle richieste in sospeso:

Se ci sono richieste di amicizia ricevute, l'utente può decidere se accettarle o rifiutarle.

- Se accetta, il sistema aggiorna il database aggiungendo l'amico e rimuovendo la richiesta pendente. Il messaggio mostrato è: “Richiesta accettata con successo”.

- Se rifiuta, la richiesta viene semplicemente eliminata, e viene visualizzato il messaggio: “Richiesta rifiutata con successo”.

Al termine delle operazioni, l'utente può chiudere la sezione amici e tornare alla schermata principale.

4 Implementazione

4.1 Classe TerrainGenerator

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class TerrainGenerator : MonoBehaviour
{
    private int x;
    private int z;
    private int h;
    private bool empty;
    private int firstObstacle;
    private int secondObstacle;
    private int enemy;
    private int startingX = 0;
    private int startingZ = 0;

    [Header("Level Size")]
    [SerializeField] private int row;
    [SerializeField] private int column;
    [SerializeField] private int y;
    private int firstTerrainZ;
    private int firstTerrainX;
    [SerializeField] private int wallHeight;
    private int rowLShaped = 20;
    private int columnLShaped = 20;

    [Header("First Obstacle position")]
    [SerializeField] private int startObstacle1X;
    [SerializeField] private int endObstacle1X;
    private int startObstacle1Z;
    private int endObstacle1Z;
    [SerializeField] private int startObstacle1Y;

    [Header("Second Obstacle position")]
    [SerializeField] private int startObstacle2X;
    [SerializeField] private int endObstacle2X;
    private int startObstacle2Z;
    private int endObstacle2Z;
    [SerializeField] private int startObstacle2Y;

    private float enemyX;
    private float enemyZ;
    private float enemyY;
}
```

```
[Header("Prefab")]
[SerializeField] private GameObject floor;
[SerializeField] private GameObject transparent;
[SerializeField] private GameObject wall;
[SerializeField] private GameObject parent;
[SerializeField] private GameObject finishPortal;
[SerializeField] private HealthManager hm;
[SerializeField] private Score sc;
[SerializeField] private GameObject inGameGUI;
[SerializeField] private GameObject winGUI;
[SerializeField] private GameObject deathGUI;
[SerializeField] private GameObject ButtonGUI;

public GameObject[] obstacles;
public GameObject[] enemies;
```

Questa prima parte di codice della classe TerrainGenerator inizializza tutte le variabili che verranno utilizzate.

Le variabili x,z e h sono variabili usate per cicli for per generare i blocchi della stanza.

La variabile bool empty serve per dire al ciclo for quando non generare alcun blocco e lasciare spazio vuoto per gli ostacoli.

firstEnigma e secondEnigma sono due variabili che conterranno la posizione dell'ostacolo nell'indice enigmas[].

startingX e startingZ sono le coordinate dalla quale si inizierà a generare il terreno della stanza.

row, column e y sono le variabili che identificano la grandezza e l'altezza della stanza.

wallHeight è l'altezza dei muri laterali.

startObstacle1X e Z e endObstacle1X e Z sono le variabili con le coordinate limite per la generazione del primo ostacolo.

startObstacle2 X e Z e endObstacle2 X e Z sono invece le variabili per il secondo ostacolo.

EnemyX Z e Y sono la posizione iniziale in cui verrà generato il nemico

I GameObject floor e wall contengono i prefab, blocchetti pre-fabbricati, da usare per generare la stanza.

Il GameObject transparent contiene il prefab che verrà utilizzato per generare dei muri invisibili nella stanza di fine livello.

Il GameObject parent sarà il parente di tutti i blocchi usati per generare la stanza.

Il GameObject finishPortal è un prefab che si inserisce a fine stanza e serve da portale per il prossimo livello.

HealthManager hm è uno script che controlla la vita e la morte del player.

Score sc gestisce i punteggi e formatta i punteggi da mostrare a fine partita.

I GameObject inGameGui, winGui, deathGui e ButtonGui sono tutti object GUI.

L'array GameObject[] obstacles contiene gli oggetti con gli script per generare gli ostacoli.

L'array GameObject enemies contiene i prefab enemies con gli script per muoverli.

```
void Start()
{
    if (PlayerPrefs.GetInt("LevelEnded") == 1 &&
    PlayerPrefs.GetInt("roomGenerated") == 0)
    {
        generateEndRoom();
        PlayerPrefs.SetInt("roomGenerated", 1);
        PlayerPrefs.Save();
    }
    else
    {
        hm.resetInvincible();
        enemyX = endObstacle1X + (startObstacle2X - endObstacle1X) / 2f;
        enemyZ = column / 2 - 0.5f;
        enemyY = 1.5f;

        int l = obstacles.Length;
        int n = enemies.Length;
        do
        {
            firstObstacle = UnityEngine.Random.Range(0, l);
            secondObstacle = UnityEngine.Random.Range(0, l);
            enemy = UnityEngine.Random.Range(0,n);
        } while (firstObstacle == secondObstacle);

        obstacles[firstObstacle].SetActive(true);
        obstacles[secondObstacle].SetActive(true);
        if(enemies[enemy].name != "Drum")
        {
            enemies[enemy].SetActive(true);
        }

        generateStraightTerrain();
    }
}
```

Nella funzione Start() con un if controllo se il player ha vinto (PlayerPrefs("LevelEnded") == 1) e se la stanza di fine livello non è ancora stata generata (PlayerPrefs("LevelEnded") == 0), se true allora richiamo la funzione generateEndRoom(), imposto la PlayerPrefs("roomGenerated") = 1 e salvo le modifiche. Se invece il player non ha vinto, annullo l'invincibilità del player, per evitare che finendo un livello da invincibile il suo status non si riavvii.

Dopodiché calcolo i valori per la posizione del nemico, così che venga generato al centro della zona tra i due ostacoli.

In un ciclo do while scelgo casualmente i due enigmi da generare e il nemico, controllando che i due ostacoli abbiano un indice diverso nell'array.

Attivo i due ostacoli agli index firstObstacle e secondObstacle.

Controllo se il nome dell'enigma è diverso da Drum, se sì attivo il nemico all'indice enemy.

Richiamo la funzione generateStraightTerrain(), che genera la stanza dritta.

Chiamo la funzione generateStraightTerrain() che genera la stanza dritta.

Creo l'oggetto _finishPortal che verrà messo a fine stanza e servirà come portale per la stanza seguente.


```

        var _enemy = Instantiate(enemies[enemy], new Vector3(enemyX, enemyY,
enemyZ), Quaternion.identity);
        _enemy.name = "Enemy";
        _enemy.transform.SetParent(parent.transform);
        _enemy.SetActive(true);

        var _finishPortal = Instantiate(finishPortal, new Vector3(row - 3, y
+ 1.5f, column - column / 2), Quaternion.identity);
        _finishPortal.name = "FinishPortal";
        _finishPortal.transform.SetParent(parent.transform);
    }
}

private void Update()
{
    if ( hm.IsDead() || PlayerPrefs.GetInt("CompletedLevels") > 4 &&
PlayerPrefs.GetInt("roomGenerated") == 0)
    {
        if (hm.IsDead())
        {
            PlayerPrefs.SetInt("Dead", 1);
        }
        PlayerPrefs.SetInt("LevelEnded", 1);
        PlayerPrefs.Save();
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}

```

Istanzio il nemico e il portale di fine livello.

Nella funzione Update() controllo se il player è morto, tramite la funzione hm.IsDead(), o se il numero di livelli completati è maggiore di 4.

Se vero, controllo se è morto, e in caso imposto una PlayerPref("Dead") con il valore di 1.

Dopodiché imposto la PlayerPref("LevelEnded") con il valore di 1, salvo il valore e carico come scena attiva quella attuale, così da entrare nel primo if dello start e generare la stanza.

```

public void generateStraightTerrain()
{
    startObstacle1Z = 0;
    endObstacle1Z = column;
    startObstacle2Z = 0;
    endObstacle2Z = column;

    for (int r = startingX; r <= row; r++)
    {

```

```

        x = r;
        if (
            ((r < startObstacle1X || r >= endObstacle1X) &&
            ((r < startObstacle2X || r >= endObstacle2X))
        )
        {
            empty = false;
        }
        else
        {
            empty = true;
        }

        for (int c = startingZ; c <= column; c++)
        {
            z = c;
            if (!empty)
            {
                string name = "pavimento[" + r.ToString() + ";" + y.ToString()
+ ";" + c.ToString() + "];";
                var cube = Instantiate(floor, new Vector3(x, y, z),
Quaternion.identity);
                cube.name = name;
                cube.transform.SetParent(parent.transform);
            }
            if (r == row || c == column)
            {
                for (int a = y; a <= wallHeight; a++)
                {
                    string nameWall = "wall[" + r.ToString() + ";" +
a.ToString() + ";" + c.ToString() + "];";
                    var wallObj = Instantiate(wall, new Vector3(x, a, z),
Quaternion.identity);
                    wallObj.name = nameWall;
                    wallObj.transform.SetParent(parent.transform);
                }
            }
        }
    }
}

```

Tramite due cicli for annidati, tratto la generazione del terreno come una tabella, e con un terzo for annidato gestisco la generazione dei muri.

Controllo se la r (variabile del ciclo for) non è nei range dei due ostacoli, se non lo è la variabile empty è uguale a true.

In un altro ciclo for, genero i blocchi del pavimento.

Al suo interno, con un if verifico se la r o la c sono uguali a row e column (i valori massimi), se sì in un ciclo for genero i muri.

```

public void generateEndRoom()
{
    if (PlayerPrefs.GetInt("roomGenerated") == 0)
    {
        for (int r = 0; r <= 7; r++)
        {
            x = r;
            for (int c = 0; c <= 7; c++)
            {
                z = c;
                string name = "pavimento[" + r.ToString() + ";" +
y.ToString() + ";" + c.ToString() + "]";
                var cube = Instantiate(floor, new Vector3(x, y, z),
Quaternion.identity);
                cube.name = name;
                cube.transform.SetParent(parent.transform);

                if (r == 7 || c == 7)
                {
                    for (int a = y; a <= 5; a++)
                    {
                        string nameWall = "wall[" + r.ToString() + ";" +
a.ToString() + ";" + c.ToString() + "]";
                        var wallObj = Instantiate(wall, new Vector3(x, a, z),
Quaternion.identity);
                        wallObj.name = nameWall;
                        wallObj.transform.SetParent(parent.transform);
                    }
                }
                else if (r == 0 || c == 0)
                {
                    for (int a = y; a <= 5; a++)
                    {
                        string nameWall = "transparentWall[" + r.ToString() +
";" + a.ToString() + ";" + c.ToString() + "]";
                        var wallObj = Instantiate(transparent, new Vector3(x,
a, z), Quaternion.identity);
                        wallObj.name = nameWall;
                        wallObj.transform.SetParent(parent.transform);
                    }
                }
            }
        }
        ButtonGUI.SetActive(true);
        sc.setGUIScore();
        inGameGUI.SetActive(false);
    }
}

```

```

        GameObject.Find("Character").GetComponent<PlayerMovement>().enabled =
false;
        Time.timeScale = 0f;
        if (PlayerPrefs.GetInt("CompletedLevels") >= 5)
        {
            HealthManager.EndScreen("Won!");
        }
        else if (PlayerPrefs.GetInt("Dead") == 1)
        {
            HealthManager.EndScreen("Lost");
        }
    }
}

```

Uguale alla funzione generateStraightTerrain(), cambia soltanto il fatto che genero i muri su tutti i lati, quelli davanti la videocamera sono invisibili, e non devo lasciare spazi vuoti.

Attivo le GUI dei Button per ricominciare il livello o tornare al menu, mostro il punteggio, nascondo la GUI del gioco (Cuori, tempo e livelli completati) e disabilito il movimento del player.

Controllo se il player ha finito perché ha vinto o se è morto, e richiamo la rispettiva funzione per attivare la GUI corretta.

4.2 Classe ParkourGenerator

```
using UnityEngine;

public class ParkourGenerator : MonoBehaviour
{
    private int startingX;
    private int endingX;
    private int startingZ;
    private int endingZ;
    private int startingY;

    private CharacterController characterController;

    [SerializeField] private GameObject parkour;
    [SerializeField] private GameObject parent;
    [SerializeField] private GameObject generator;

    void Start()
    {
        startingX =
generator.GetComponent<TerrainGenerator>().getStartX("ParkourGenerator");
        startingZ =
generator.GetComponent<TerrainGenerator>().getStartZ("ParkourGenerator");
        startingY =
generator.GetComponent<TerrainGenerator>().getStartY("ParkourGenerator");
        endingX =
generator.GetComponent<TerrainGenerator>().getEndX("ParkourGenerator");
        endingZ =
generator.GetComponent<TerrainGenerator>().getEndZ("ParkourGenerator");

        int oldZ = 0;
        int z = 0;
        int x = 0;
        bool first = true;
    }
}
```

In questa prima parte di codice, inizializziamo variabili private che serviranno come limitatori per saper dove generare il parkour.

Il GameObject parkour è un prefab, ovvero i blocchi su cui il player dovrà saltare.

Il GameObject generator è un oggetto su cui c'è lo script "TerrainGenerator" che, quando richiamato, restituisce i valori per limitare il parkour.

Nella funzione Start richiamiamo il generator e assegniamo alle variabili prima inizializzate i valori StartX (l'X di partenza da dove il parkour deve venir generato), StartZ (la Z di partenza da dove il parkour deve venir generato), StartY (La Y di partenza), EndX (L'ultimo valore di X in cui bisogna generare il parkour) e EndZ (L'ultimo valore di Z in cui bisogna generare il parkour).

Inizializziamo poi int oldZ, che terrà conto della Z dell'ultimo blocco generato, int z e x che saranno le coordinate del blocco e bool first, che servirà a sapere se si sta generando il primo blocco oppure no.

```

while(x < endingX)
{
    if (first){
        x = startingX + Random.Range(0,3);
        z = Random.Range(startingZ, endingZ);
        first = false;
    }
    else
    {
        while(z>=endingZ || z < 0 || z==oldZ)
        {
            z = Random.Range(oldZ-2, oldZ + 3);
        }
    }
    oldZ = z;

    var cube = Instantiate(parkour, new Vector3(x, startingY, z),
Quaternion.identity);
    cube.name = "parkour[" + x + "; "+z+"]";
    cube.transform.SetParent(parent.transform);
    x += Random.Range(2,4);
    z = -1;
}
}
}

```

Qui inizia la generazione del parkour, in un ciclo while controllo che x sia minore di endingX. Finché questo while è vero, controllo se first è vero, e se sì, genero una x aggiungendo un numero casuale da 0 a 2 alla startingX, così che anche il primo blocco del parkour sia casuale, e non sia sempre attaccato alla pedana. Genero anche la prima z, che avrà come limiti gli stessi del parkour, essendo il primo blocco è raggiungibile da ogni posizione.

Se non è il primo blocco, quindi first è false, genero una z casualmente, ma per renderla raggiungibile la limito a oldZ-2 a oldZ+3 (escluso), e nel mentre, tramite un while, controllo che il valore di Z non sia al di fuori dei limiti del parkour.

Salvo il valore di z in oldZ e genero il cubo del parkour

4.3 Classe FakeFloorGenerator

```
public class FakeFloorGenerator : MonoBehaviour
{
    private int startingX;
    private int endingX;
    private int startingZ;
    private int endingZ;
    private int startingY;

    [SerializeField] private GameObject realFloor;
    [SerializeField] private GameObject fakeFloor;
    [SerializeField] private GameObject parent;
    [SerializeField] private GameObject generator;

    void Start()
    {
        startingX =
generator.GetComponent<TerrainGenerator>().getStartX("FakeFloorGenerator");
        startingZ =
generator.GetComponent<TerrainGenerator>().getStartZ("FakeFloorGenerator");
        startingY =
generator.GetComponent<TerrainGenerator>().getStartY("FakeFloorGenerator");
        endingX =
generator.GetComponent<TerrainGenerator>().getEndX("FakeFloorGenerator");
        endingZ =
generator.GetComponent<TerrainGenerator>().getEndZ("FakeFloorGenerator");
    }
}
```

Questo primo pezzo del codice è identico a quello della classe [ParkourGenerator](#), cambia soltanto i GameObject realFloor e fakeFloor, che sono i due Prefab utilizzati per generare il percorso effettivo da seguire e i blocchi da non calpestare.

```
int[] posZ = new int[2];
int oldZ = 0;
bool first = true;
int zBlock = 0;
```

In questo pezzo di codice inizializziamo un array che conterrà le due posizioni Z, essendo che ci saranno due blocchi per ogni coordinata X.

```

for (int x = startingX; x < endingX; x++)
{
    if (first)
    {
        for (int i = 0; i < 2; i++)
        {
            zBlock = Random.Range(startingZ + 1, endingZ);
            posZ[i] = zBlock;
        }

        first = false;
    }
    else
    {
        for (int i = 0; i < 2; i++)
        {
            oldZ = posZ[i];
            do
            {
                zBlock = Random.Range(oldZ - 1, oldZ + 2);
            } while (zBlock >= endingZ || zBlock <= startingZ);

            posZ[i] = zBlock;
        }
        oldZ = zBlock;

        for (int z = startingZ; z <= endingZ; z++)
        {
            if (posZ.Contains(z))
            {
                var percorso = Instantiate(realFloor, new Vector3(x, startingY,
z), Quaternion.identity);
                percorso.name = "realFloor[" + x + "; " + z + "]";

                percorso.transform.SetParent(parent.transform);
            }
            else
            {
                var cube = Instantiate(fakeFloor, new Vector3(x, startingY, z),
Quaternion.identity);
                cube.name = "fakeFloor[" + x + "; " + z + "]";

                cube.transform.SetParent(parent.transform);
            }
        }
    }
}

```

Qui inizia la generazione del percorso, in un ciclo for controllo che x sia minore di endingX. Finché x è minore, controllo se first è vero, e se sì, dentro un ciclo for genero anche le prime 2 z, che avranno come limiti gli stessi della piattaforma, essendo i primi blocchi sono raggiungibili da ogni posizione. Non controllo se le Z sono diverse, essendo che i due percorsi possono intrecciarsi. Se non è il primo blocco, quindi first è false, genero una z casualmente, ma per renderla raggiungibile la limito a oldZ-1 a oldZ+2 (escluso), e nel mentre, tramite un while, controllo che il valore di Z non sia al di fuori dei limiti del percorso. Salvo il valore di z (posZ[i]) in oldZ e genero i cubi del percorso.

4.4 Classe WallPedanaGenerator

```
private int startingX;
private int endingX;
private int startingZ;
private int endingZ;
private int startingY;

[SerializeField] private GameObject realBlock;
[SerializeField] private GameObject fakeBlock;
[SerializeField] private GameObject firstBlock;
[SerializeField] private GameObject wall;
[SerializeField] private GameObject parent;
[SerializeField] private GameObject generator;

void Start()
{
    startingX =
generator.GetComponent<TerrainGenerator>().getStartX("WallPedanaGenerator");
    startingZ =
generator.GetComponent<TerrainGenerator>().getStartZ("WallPedanaGenerator");
    startingY =
generator.GetComponent<TerrainGenerator>().getStartY("WallPedanaGenerator");
    endingX =
generator.GetComponent<TerrainGenerator>().getEndX("WallPedanaGenerator");
    endingZ =
generator.GetComponent<TerrainGenerator>().getEndZ("WallPedanaGenerator")-3;

    int[] posX = new int[endingX - startingX];
    int[] posZ = new int[endingX - startingX];

    int oldZ = 0;
    bool first = true;
    bool generateFirst = true;
    int i = 0;
    int blockZ = 0;
```

Come per le altre classe generator, inizializzo le variabili limitatori per la generazione dell'ostacolo e i prefab per generare i blocchi invisibili finti, veri, i blocchi soluzione sul muro e il primo blocco.

I due array posX e posZ contengono le coordinate dei veri blocchi, che serviranno dopo per generare la soluzione sul muro.

A endingZ togliamo 3 perché vogliamo che le prime due colonne attaccate al muro non vengano calcolate nella generazione dell'ostacolo.

```
for (int x = startingX; x < endingX; x++)
{
    if (first)
    {
        blockZ = Random.Range(startingZ + 1, endingZ);
        first = false;
    }
    else
    {
        do
        {
            blockZ = Random.Range(oldZ - 1, oldZ + 2);
        } while (blockZ >= endingZ || blockZ <= startingZ);
    }
    oldZ = blockZ;
```

Come per il generatore del fakefloor, controllo in un ciclo for che x sia minore di endingX, nel mentre genero le Z del blocco

```
for (int z = startingZ; z <= endingZ; z++)
{
    if (z == blockZ)
    {
        var block = gameObject;
        if (generateFirst)
        {
            block = firstBlock;
            generateFirst = false;
        }
        else
        {
            block = realBlock;
        }
        var percorso = Instantiate(block, new Vector3(x, startingY, z),
Quaternion.identity);
        percorso.name = "realBlock[" + x + "; " + z + "]";
        posX[i] = x;
        posZ[i] = z;

        percorso.transform.SetParent(parent.transform);
        i++;
    }
    var cube = Instantiate(fakeBlock, new Vector3(x, startingY, z),
Quaternion.identity);
    cube.name = "fakeBlock[" + x + "; " + z + "]";
    cube.transform.SetParent(parent.transform);
}
}
```

Questa parte di codice genera l'ostacolo, in un ciclo for genero una Z con valore da startingZ a endingZ. Controllo che z sia uguale a blockZ, ovvero la variabile contenente la coordinata del blocco Z. Se equivale, controllo se sia il primo blocco, se sì il valore della variabile block è il prefab firstBlock, ovvero un blocco visibile, altrimenti è un blocco invisibile ma calpestabile (realBlock). Salvo la x e la z negli array posX e posZ, così da poter rigenerare il percorso sul muro più tardi. Se z è diverso da blockZ, allora genero un blocco con prefab fakeblock.

```
for (int j = 0; j < posX.Length; j++)
{
    var cube = Instantiate(wall, new Vector3(posX[j], posZ[j]+1, endingZ +
2.9f), Quaternion.identity);
    var wallX = startingX + j;
    cube.name = "realBlockWall[" + wallX + "; " + posX[j] + " ; " + endingZ
+ "]";
    cube.transform.SetParent(parent.transform);
}
```

Qui genero la soluzione sul muro, essendo che bisogna raffigurare la soluzione sul pavimento, la variabile posZ prende il valore di Y, e il valore di Z è endingZ + 2.9f, così che il blocco utilizzato esca dal muro di 0.1f e sia visibile.

4.5 Classe CannoBall

```
using UnityEngine;

public class CannonBall : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        Destroy(this.gameObject);
        HealthManager hm = HealthManager.Instance;
        hm.LooseOneHeart();
    }
}
```

Questo script è associato al proiettile sparato dai cannoni, chiamato "CannonBall".

Quando il proiettile collide con un altro oggetto, rilevato dal metodo OnTriggerEnter(Collider other):

Il proiettile stesso viene distrutto immediatamente con Destroy(this.gameObject), per rimuoverlo dalla scena e liberare risorse.

Viene quindi recuperata l'istanza singleton dello script HealthManager tramite HealthManager.Instance.

Viene chiamato il metodo LooseOneHeart() sul gestore della salute, per far perdere al giocatore un punto vita in seguito al colpo subito.

In pratica, questo script gestisce l'impatto del proiettile con il giocatore o con qualsiasi altro collider abilitato, attivando la perdita di salute e facendo sparire il proiettile.

4.6 Classe CannoGenerator

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

public class CannonGenerator : MonoBehaviour
{
    [SerializeField] private GameObject generator;
    [SerializeField] private GameObject cannonPrefab;
    [SerializeField] private GameObject bulletPrefab;
    [SerializeField] private GameObject parent;
    [SerializeField] private GameObject floor;
    [SerializeField] private int cannonCount;
    [SerializeField] private float fireInterval;
    [SerializeField] private float fireSpeed;
    [SerializeField] private float fireHeight;
    [SerializeField] private float bulletLifetime;
    private int startingX;
    private int endingX;
    private int startingZ;
    private int endingZ;
    private int startingY;

    private GameObject[] cannons;
    private List<GameObject> bullets = new List<GameObject>();
}
```

```

void Start()
{
    startingX =
generator.GetComponent<TerrainGenerator>().getStartX("CannonGenerator");
    startingZ =
generator.GetComponent<TerrainGenerator>().getStartZ("CannonGenerator");
    startingY =
generator.GetComponent<TerrainGenerator>().getStartY("CannonGenerator");
    endingX =
generator.GetComponent<TerrainGenerator>().getEndX("CannonGenerator");
    endingZ =
generator.GetComponent<TerrainGenerator>().getEndZ("CannonGenerator");

    int[] posX = new int[cannonCount];
    cannons = new GameObject[cannonCount];
    for (int i = 0; i < cannonCount; i++)
    {
        Vector3 cannonPosition = new Vector3(startingX, startingY, endingZ-
0.1f) + new Vector3(i * 2.0f, fireHeight, 0); //distanza fra i cannoni (2.0f)
        cannons[i] = Instantiate(cannonPrefab, cannonPosition,
Quaternion.identity);
        cannons[i].name = "Cannone" + i;
        cannons[i].transform.SetParent(parent.transform);
    }
    StartCoroutine(ShootCannons());

    for (int x = startingX; x < endingX; x++)
    {
        for (int z = startingZ; z <= endingZ; z++)
        {
            var cube = Instantiate(floor, new Vector3(x, startingY, z),
Quaternion.identity);
            cube.name = "floor[" + x + "; " + z + "]";
            cube.transform.SetParent(parent.transform);
        }
    }
}

void Update()
{
    for (int i = 0; i < bullets.Count; i++)
    {
        if (bullets[i] != null)
        {
            bullets[i].transform.Translate(Vector3.back * fireSpeed *
Time.deltaTime);

```

```

        bullets[i].transform.SetParent(parent.transform);
        if (bullets[i].transform.position.z < -20f)
        {
            Destroy(bullets[i]);
            bullets.RemoveAt(i);
            i--;
        }
    }
}

IEnumerator ShootCannons()
{
    while (true)
    {
        for (int i = 0; i < cannonCount; i += 2)
        {
            FireBullet(cannons[i].transform.position);
        }
        yield return new WaitForSeconds(fireInterval);

        for (int i = 1; i < cannonCount; i += 2)
        {
            FireBullet(cannons[i].transform.position);
        }
        yield return new WaitForSeconds(fireInterval);
    }
}

void FireBullet(Vector3 position)
{
    Vector3 bulletSpawnPosition = position + new Vector3(0, 0, 0f);
    GameObject bullet = Instantiate(bulletPrefab, bulletSpawnPosition,
Quaternion.identity);
    bullets.Add(bullet);
    Destroy(bullet, bulletLifetime);
}
}

```

Questo script si occupa di generare una serie di cannoni su un'area di gioco e di farli sparare proiettili a intervalli regolari.

All'inizio, nel metodo Start(), lo script prende come riferimento un oggetto chiamato generator che contiene lo script TerrainGenerator. Da questo script recupera i limiti dell'area di generazione, ossia le coordinate iniziali e finali sugli assi X, Y e Z.

Successivamente, viene creato un array per i cannoni, e con un ciclo for vengono istanziati i cannoni uno accanto all'altro lungo l'asse X, a una certa altezza (fireHeight) e leggermente spostati sull'asse Z (vicino al limite endingZ). Ogni cannone viene nominato "Cannone" seguito dal suo indice e viene assegnato come figlio di un oggetto parent per mantenere l'organizzazione nella gerarchia.

Dopo aver creato i cannoni, lo script avvia una coroutine chiamata ShootCannons() che gestisce il fuoco alternato dei cannoni.

Sempre nel Start(), viene creato un pavimento fatto di cubi, generando istanze di un prefab floor su tutta l'area delimitata dagli intervalli startingX-endingX e startingZ-endingZ. Ogni cubo viene nominato con la sua posizione e messo come figlio di parent.

Nel metodo Update(), lo script si occupa di aggiornare il movimento dei proiettili sparati. Per ogni proiettile nella lista bullets:

- Viene fatto muovere all'indietro (lungo l'asse Z negativo) alla velocità fireSpeed.
- Il proiettile viene mantenuto come figlio di parent (per mantenere l'ordine nella gerarchia).
- Se il proiettile supera una certa posizione limite sull'asse Z (inferiore a -20), viene distrutto e rimosso dalla lista per evitare sprechi di risorse.

La coroutine ShootCannons() gestisce il tiro alternato dei cannoni in due gruppi:

- Prima fa sparare tutti i cannoni con indice pari (0, 2, 4, ...) simultaneamente.
- Poi attende un intervallo di tempo fireInterval.
- Successivamente fa sparare tutti i cannoni con indice dispari (1, 3, 5, ...).
- Poi attende nuovamente lo stesso intervallo.
- Questo ciclo si ripete all'infinito, creando un ritmo alternato di fuoco.

La funzione FireBullet(Vector3 position) crea un proiettile all'altezza e posizione del cannone passato come parametro, aggiungendolo alla lista di proiettili attivi e impostandone una durata di vita (bulletLifetime) dopo cui viene automaticamente distrutto.

4.7 Classe PortalGenerator

```
using UnityEngine;

public class PortalGenerator : MonoBehaviour
{
    [SerializeField] private GameObject endPortalPrefab;
    [SerializeField] private GameObject goodPortalPrefab;
    [SerializeField] private GameObject badPortalPrefab;
    [SerializeField] private GameObject parent;
    private int numberOfPortals = 4;
    private Vector3[] portalPositions;
    [SerializeField] private Vector3[] portalRotations;
    [SerializeField] private AudioClip goodSFX;
    [SerializeField] private AudioClip badSFX;
    [SerializeField] private GameObject generator;

    private GameObject[] portalsArray;
    private int goodPortal;

    private int endPortalX;
    private float y;
    private int endPortalZ;

    void Start()
    {
        int x =
generator.GetComponent<TerrainGenerator>().getStartX("PortalGenerator")-1;
        int c =
(generator.GetComponent<TerrainGenerator>().getEndZ("PortalGenerator") -
generator.GetComponent<TerrainGenerator>().getStartZ("PortalGenerator")) /
numberOfPortals;
        int z =
generator.GetComponent<TerrainGenerator>().getStartZ("PortalGenerator");
        y =
generator.GetComponent<TerrainGenerator>().getStartY("PortalGenerator")+1.5f;

        endPortalX =
generator.GetComponent<TerrainGenerator>().getEndX("PortalGenerator");
        endPortalZ = z + c * numberOfPortals / 2;
        goodPortal = Random.Range(0, numberOfPortals);

        for (int i = 0; i < numberOfPortals; i++)
        {
            GameObject portalPrefab = (i == goodPortal) ? goodPortalPrefab :
badPortalPrefab;
            if (i == 0)
```

```

        {
            GameObject endPortal = Instantiate(endPortalPrefab, new
Vector3(endPortalX, y, endPortalZ), Quaternion.Euler(portalRotations[i]));
            endPortal.name = "EndPortal";
            endPortal.tag = "Untagged";
            endPortal.transform.SetParent(parent.transform);
        }
        GameObject newPortal = Instantiate(portalPrefab, new
Vector3(x,y,z+c*i), Quaternion.Euler(portalRotations[i]));

        newPortal.name = "Portal" + i;
        newPortal.transform.SetParent(parent.transform);

        AudioSource audioSource = newPortal.AddComponent<AudioSource>();
        audioSource.loop = true;
        audioSource.clip = (i == goodPortal) ? goodSFX : badSFX;
    }
}

public int getEndX()
{
    return endPortalX;
}

public int getEndZ()
{
    return endPortalZ;
}
}

```

Questo script si occupa di generare una serie di portali posizionati lungo un lato del terreno di gioco, includendo un portale finale (end portal), un portale “buono” e più portali “cattivi”.

Nel metodo Start(), lo script utilizza un oggetto generator che contiene lo script TerrainGenerator per ricavare i limiti dell’area di gioco, in particolare le coordinate di inizio e fine sugli assi X, Y e Z, relative al generatore dei portali.

- La coordinata X del portale viene fissata appena prima dell’inizio dell’area (startX - 1).
- La coordinata Y è impostata un po’ più alta rispetto al terreno per posizionare i portali leggermente sopra il suolo.
- L’asse Z viene suddiviso in intervalli uguali per posizionare uniformemente i portali lungo questa direzione.

Lo script determina casualmente quale dei portali sarà quello “buono” (che rappresenta probabilmente la scelta corretta per il giocatore).

Successivamente, nel ciclo for, per ogni portale da generare:

- Se è il primo portale (indice 0), viene creato il portale finale (endPortalPrefab), posizionato nella coordinata calcolata a metà lungo l'asse Z, con una rotazione specifica, e viene rinominato "EndPortal". Questo portale non ha tag specifico.
- Per tutti gli altri portali viene scelto il prefab corretto tra “buono” e “cattivo” in base all'indice casuale scelto.
- Ogni portale viene istanziato nella posizione calcolata lungo X, Y, Z e ruotato secondo una rotazione specifica fornita da un array.
- I portali vengono organizzati come figli di un oggetto parent.
- A ogni portale viene aggiunto un componente AudioSource con il relativo suono di sottofondo, che viene impostato in loop e differenziato per il portale buono (sound positivo) e per quelli cattivi (sound negativo).

Infine, lo script espone due metodi pubblici getEndX() e getEndZ() che ritornano le coordinate X e Z del portale finale, probabilmente per permettere ad altri script di sapere dove si trova il portale di uscita.

4.8 Classe Score

```
public class Score : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI scoreText;
    private float scoreValue;
    private int hundredths;
    private int seconds;
    private int minutes;
    private string time;
    [SerializeField] private GameObject scoreParent;
    [SerializeField] private API_CALL api;
    public TextMeshProUGUI[] scores;
    public TextMeshProUGUI finalScore;

    void Start()
    {
        scoreValue = 0f;
    }

    void Update()
    {
        if (scoreValue < 300) //5 minutes = 300 seconds
        {
            scoreValue += Time.deltaTime;
        }

        hundredths = Mathf.FloorToInt(scoreValue * 100) % 100;
        seconds = Mathf.FloorToInt(scoreValue % 60);
        minutes = Mathf.FloorToInt(scoreValue / 60);
        setScore(minutes, seconds, hundredths);
    }

    public string getScore()
    {
        return time;
    }

    public void setScore(int m, int s, int h)
    {
        scoreText.text = string.Format("{0:00}:{1:00}.{2:00}", m, s, h);
        time = scoreText.text;
    }

    public void setGUIScore()
    {
        TimeSpan timeSum = new TimeSpan(0, 0, 0, 0, 0);
    }
}
```

```

scoreParent.SetActive(true);
for (int i = 0; i < PlayerPrefs.GetInt("CompletedLevels"); i++)
{
    string name = "Time" + i;
    int j = i + 1;
    TimeSpan timeForSum = TimeSpan.Parse("00:00:" +
PlayerPrefs.GetString(name));
    timeSum += timeForSum;
    scores[i].text = string.Format("Score {0}: " +
PlayerPrefs.GetString(name), j);
}
for(int i = PlayerPrefs.GetInt("CompletedLevels");i<5; i++)
{
    int j = i + 1;
    scores[i].text = string.Format("");
}
string finalTime = timeSum.ToString();
finalTime = finalTime.Remove(0, 3);
print(finalTime.Length);
finalScore.text = string.Format("Final score: " + finalTime);
PlayerPrefs.SetString("score", finalTime);
if (PlayerPrefs.GetInt("CompletedLevels") > 4)
{
    StartCoroutine(api.addScore());
}
}
}

```

Inizializzo un TextMeshProGui scoreText, che fungerà da GUI per mostrare il tempo passato. scoreValue terrà il valore del tempo, hundredths sono i centesimi, seconds i secondi e minutes i minuti. La variabile time contiene la stringa che contiene il tempo formattato.

API_CALL è uno script che si interfaccia con il database.

L'array scores contiene tutte le GUI dei testi degli score che verranno mostrati a fine partita.

Nello Start impostiamo scoreValue a 0, così ogni livello avrà il proprio tempo.

Nell'Update verifichiamo che il player non superi il tempo limite di 5 minuti, fino a quel punto aumentiamo il tempo.

Eseguiamo i calcoli per convertire il valore numerico del tempo trascorso dall'inizio dello script, per convertirlo e inserirlo in una stringa.

La funzione setScore() formatta il testo della scoreText (la gui) al tempo calcolato prima. Salviamo questo valore nella variabile time.

setGUIScore() viene richiamato quando si vince il gioco, inserisce nelle GUI i tempi impiegati e nasconde eventuali GUI di livelli non completati.

In questa funzione calcoliamo anche il tempo totale impiegato a completare i livelli.

Se si ha completato 5 livelli, questo valore viene salvato nel Database.

4.9 Classe API_CALL

```
public class API_CALL : MonoBehaviour
{
    private string URL = "localhost:8080/API/public";
    private void Start()
    {
        StartCoroutine(addUser("SamTrevano"));
    }

    private UnityWebRequest CreateRequest(string path, RequestType type =
RequestType.GET, object data = null)
    {
        var request = new UnityWebRequest(path, type.ToString());

        if (data != null)
        {
            var bodyRaw = Encoding.UTF8.GetBytes(JsonUtility.ToJson(data));
            request.uploadHandler = new UploadHandlerRaw(bodyRaw);
        }

        request.downloadHandler = new DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type", "application/json");

        return request;
    }

    private IEnumerator getUser(String username)
    {
        String url = URL + "/user/" + username;
        var getRequest = CreateRequest(url);
        yield return getRequest.SendWebRequest();

        string text = getRequest.downloadHandler.text;
        text = text.Replace("[", "");
        text = text.Replace("]", "");

        if (text == null)
        {
            User user = JsonUtility.FromJson<User>(text);
            PlayerPrefs.SetInt("userId", user.id);
            PlayerPrefs.SetString("userName", user.username);
            PlayerPrefs.Save();

            yield return user;
        }
        else
    }
```

```

        {
            yield return null;
        }

    }

    private IEnumerator addUser(string _username)
    {
        if (getUser(_username) != null)
        {
            UserToAdd user = new UserToAdd() { username = _username, password =
"PasswordSicura", email = "www.samtrevano@sam.ch" };
            var postRequest = CreateRequest(URL + "/user", RequestType.POST,
user);
            yield return postRequest.SendWebRequest();

            if (postRequest.result != UnityWebRequest.Result.Success)
            {
                yield break;
            }
            StartCoroutine(getUser(user.username));
        }
    }

    public IEnumerator addScore()
    {
        ScoreAPI scoreAPI = new ScoreAPI() { score =
PlayerPrefs.GetString("score"), user_id = PlayerPrefs.GetInt("userId") };
        var postRequest = CreateRequest(URL + "/user", RequestType.POST,
scoreAPI);
        yield return postRequest.SendWebRequest();

        if (postRequest.result != UnityWebRequest.Result.Success)
        {
            yield break;
        }
    }

    private IEnumerator updateScore(ScoreAPI _scoreApi)
    {
        int id = _scoreApi.user_id;
        String _url = URL + "/user/" + id;
        var putRequest = CreateRequest(_url, RequestType.PUT, _scoreApi);
        yield return putRequest.SendWebRequest();

        if (putRequest.result != UnityWebRequest.Result.Success)
        {

```

```

        yield break;
    }
}

public enum RequestType
{
    GET = 0,
    POST = 1,
    PUT = 2
}

public class User
{
    public int id;
    public string username;
    public string password;
    public string email;
    public string type;
}


public class UserToAdd
{
    public string username;
    public string password;
    public string email;
}

public class ScoreAPI
{
    public int user_id;
    public String score;
}

```

Nello start avvio la coroutine addUser, che aggiunge un utente chiamato "SamTrevano" nel database. La funzione CreateRequest richiede tre parametri, l'URL a cui mandare la richiesta, il tipo di richiesta (GET, POST o PUT) e eventualmente dei dati da mandare, in caso di POST o PUT. Creiamo una variabile request che contiene una new UnityWebRequest. Controllo se si ha passato un data, se si faccio un encoding in JSON per mandarlo. Ritorno la richiesta.

Questa funzione verrà utilizzata da tutte le altre per mandare le richieste. Nella funzione getUser(String USername), creo una request tramite la funzione CreateRequest, e ritorno la risposta della richiesta. in una variabile text salvo il risultato ritornato dalla richiesta (downloadHandler.text), e rimuovo da esso le parentesi quadre che lo circondano, inutili in questo caso. Se non si rimuovono, non si può fare l'encoding del testo.

	SAMT – Sezione Informatica	Pagina 47 di 101
	Colorful Songs	

Se il text è diverso da null, deserealizzo il JSON e salvo nelle playerPref l'id e l'username dell'user, altrimenti ritorno null.

la funzione addUser() usa getUser per verificare se l'username che voglio usare non è già utilizzato. Creo un UserToAdd, ed inserisco l'username, password e email. Creo una richiesta in post, verifico che il risultato sia positivo, se sì, chiamo getUser con l'username dell'utente per salvare il suo id e username nelle playerPref.

Le funzioni addScore e updateScore sono identiche, cambia solo il metodo utilizzato. Creo un scoreApi, che contiene lo score finale e l'id dell'user, creo una richiesta in POST (addScore) o PUT (updateScore).

public enum RequestType viene usato per definire il tipo della richiesta. UserToAdd è diverso da User perchè non ha l'ID, essendo che viene usato per fare il post e l'ID non viene impostato manualmente. ScoreAPI contiene l'id dell'user e lo score finali, entrambi presi dalle playerPref.

4.10 Telecamera

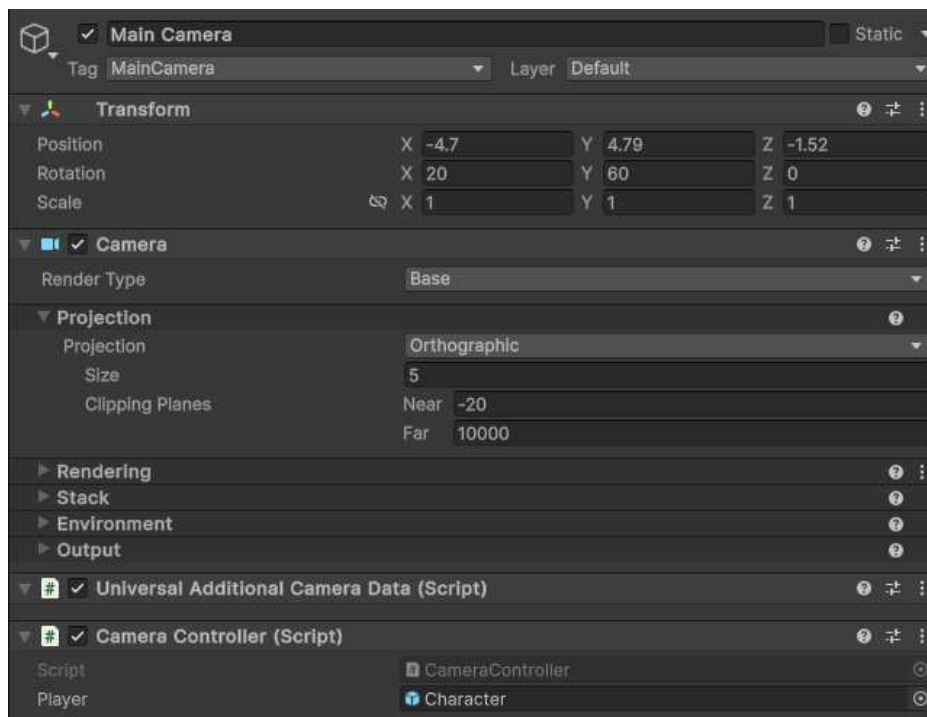


Figura 20: Componenti Telecamera

Questa è la struttura dei componenti della Main Camera.

La telecamera è di tipo “Orthographic”, per avere un miglior effetto di isometrico il valore di Near è -20 in modo che venga renderizzato anche ciò che è più indietro ma comunque presente nello schermo.

“Universal Additional Camera Data” è uno script di default di unity che serve per far funzionare gli oggetti camera.

Invece, “Camera Controller” è uno script al quale si passa il game object del personaggio e serve per far sì che la telecamera sia “attaccata” al giocatore e si sposti con lui.

4.10.1 Camera Controller

```
public class CameraController : MonoBehaviour
{
    [SerializeField] public GameObject player;
    private Vector3 offset;
    private float y;
    void Start()
    {
        offset = transform.position - player.transform.position;
        y = transform.position.y;
    }

    void Update()
    {
        transform.position = player.transform.position + offset;
        transform.position = new Vector3(transform.position.x, y,
transform.position.z);
    }
}
```


4.11 Audio

Per prima cosa bisogna avere un mixer audio con il quale si possono gestire i diversi tipi di audio, nel nostro caso gestiamo solo due categorie, music e SFX ed inoltre il master che gestisce entrambi.

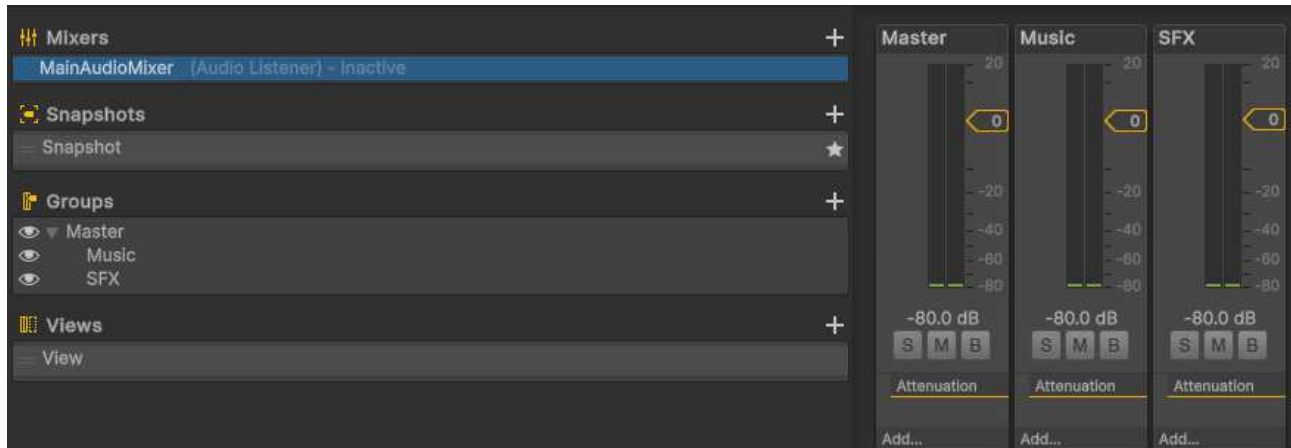


Figura 22: Audio Mixer

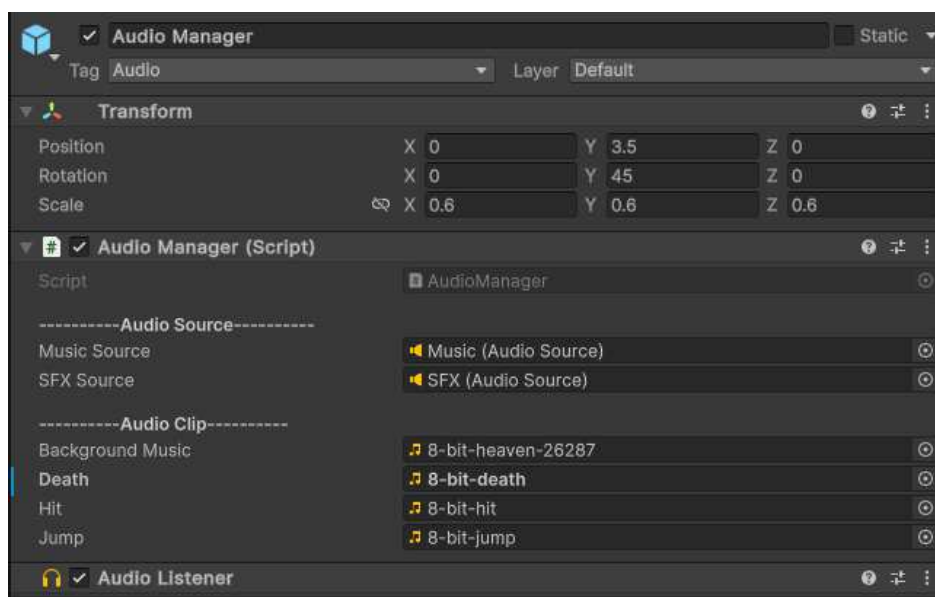


Figura 21: Componenti Audio Manager

Il game object dell'audio manager è all'interno del personaggio e contiene sia l'Audio listener che è quello che permette di gestire come si sente l'audio e fino a quanto distante. Inoltre, contiene anche lo script per la gestione degli audio, allo script si passano, le due sorgenti audio, quella per la musica e quella per gli SFX, e tutte le varie clip audio necessarie nel gioco: la musica di background, il sound quando si muore, quando si prende danno e quando si salta.

4.11.1 Audio Manager

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    [Header("-----Audio Source-----")]
    [SerializeField] private AudioSource musicSource;
    [SerializeField] private AudioSource SFXSource;

    [Header("-----Audio Clip-----")]
    [SerializeField] private AudioClip backgroundMusic;
    [SerializeField] private AudioClip death;
    [SerializeField] private AudioClip hit;
    [SerializeField] private AudioClip jump;

    void Start()
    {
        musicSource.clip = backgroundMusic;
        musicSource.Play();
    }

    public void PlaySFX(AudioClip clip)
    {
        SFXSource.PlayOneShot(clip);
    }
}
```

Questo script è associato al gestore dell'audio chiamato AudioManager. Serve a gestire l'audio all'interno del gioco, ovvero la musica di sottofondo e gli effetti sonori. Il suo scopo è dividere i vari tipi di suoni in due categorie principali: la musica e gli effetti sonori, per garantire che vengano riprodotti separatamente e permettere la gestione dei volumi.

Dichiarazione delle variabili:

- musicSource: un AudioSource utilizzato per riprodurre la musica di sottofondo.
- SFXSource: un altro AudioSource che si occupa di riprodurre gli effetti sonori (come morte, colpo, salto, ecc.).
- backgroundMusic, death, hit, jump: clip audio che rappresentano rispettivamente la musica di sottofondo, il suono della morte, il suono del colpo e il suono del salto.

Start():

- All'inizio, nel metodo Start(), viene impostato il clip di musicSource con la musica di sottofondo (backgroundMusic), che poi viene riprodotto utilizzando il metodo Play().
- PlaySFX(AudioClip clip):
- Questo metodo consente di riprodurre un effetto sonoro, dato come parametro, usando il PlayOneShot() sull'oggetto SFXSource. PlayOneShot() è usato per riprodurre clip audio brevi come gli effetti sonori, senza interrompere altre eventuali tracce audio.

Getter per gli effetti sonori:

- I metodi getDeath(), GetHit() e GetJump() restituiscono rispettivamente le clip audio associate agli effetti di morte, colpo e salto. Questi metodi possono essere chiamati da altre parti del codice per ottenere questi clip audio e riprodurli tramite il metodo PlaySFX().

Funzionalità e uso, lo script gestisce due categorie principali di audio nel gioco:

- La musica di sottofondo viene riprodotta continuamente.
- Gli effetti sonori in gioco, come il salto.

Per esempio, quando il personaggio muore, il gioco può chiamare il metodo PlaySFX(death) per riprodurre l'effetto sonoro della morte. Analogamente, per un salto, si chiamerebbe PlaySFX(jump).

4.12 Personaggio

Questa è la struttura dei game object del personaggio.

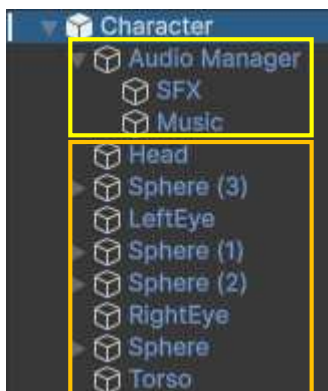


Figura 23: Struttura Player

4.12.1 Componenti

Questa è la struttura dei componenti del personaggio, il quale è un prefab.

È presente un Box Collider per poter gestire le collisioni ed interazioni con gli altri game object fisici presenti nel gioco, quali colpi dei cannoni, portali, nemici, etc.

Grazie al Character controller si può comandare il nostro personaggio.

Lo script "Player Movement" presenta il codice effettivo per poter far muovere il personaggio e gli si passano i valori per: la velocità normale con cui cammina, quella per la corsa, l'accelerazione gravitazionale e l'altezza massima a cui può saltare.

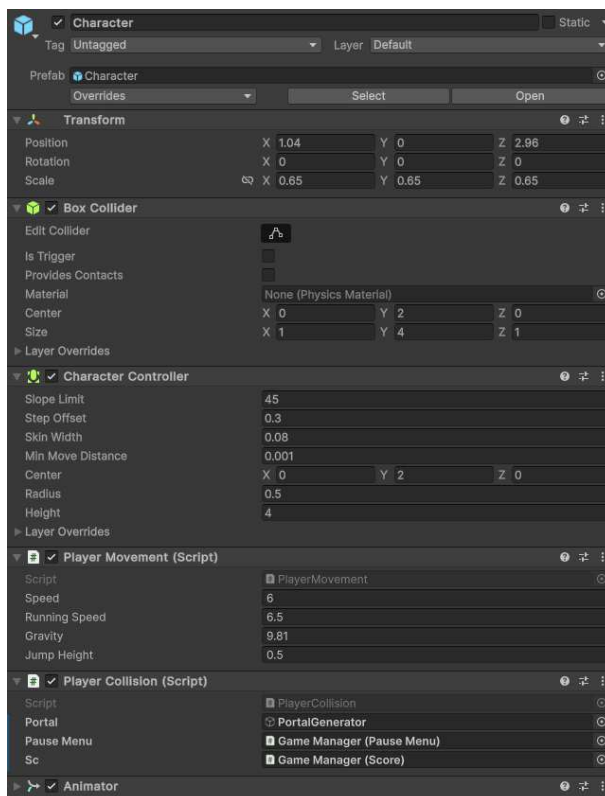


Figura 24: Componenti Player

4.12.2 Player Movement

```
public class PlayerMovement : MonoBehaviour
{
    [SerializeField] private float speed = 6.0f;
    [SerializeField] private float runningSpeed = 6.5f;
    [SerializeField] private float gravity = 9.81f;
    [SerializeField] private float jumpHeight = 0.5f;
    private AudioManager audioManager;
    private CharacterController characterController;
    private float verticalVelocity;
    private float x;
    private float y;
    private float z;
    private Vector3 position;
    private float voidHeight = -15;
    private int defaultMovement;
    private float initialRotationY;
    private HealthManager healthManager;

    private void Awake()
    {
        audioManager =
            GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
    }

    void Start()
    {
        healthManager = new HealthManager();
        characterController = GetComponent<CharacterController>();
        if (!PlayerPrefs.HasKey("DefaultMovement"))
        {
            PlayerPrefs.SetInt("DefaultMovement", 1);
            PlayerPrefs.Save();
        }
        defaultMovement = PlayerPrefs.GetInt("DefaultMovement");

        if (defaultMovement == 0)
        {
            initialRotationY = 0f;
        }
        else if (defaultMovement == 1)
        {
            initialRotationY = -45f;
        }

        transform.rotation = Quaternion.Euler(0, initialRotationY, 0);
    }

    void Update()
    {
        Movement();
        Turn();
    }

    private void Movement()
    {
        x = Input.GetAxis("Horizontal");
        z = Input.GetAxis("Vertical");
    }
}
```

```

    if (x > 0f || z > 0f || x < 0f || z < 0f)
    {
        GetComponent<Animator>().SetBool("isWalking", true);
    }
    else
    {
        GetComponent<Animator>().SetBool("isWalking", false);
    }
    y = VerticalForceCalculation();
    if(defaultMovement == 0)
    {
        position = new Vector3(x + z, y, z - x);
    }
    else if (defaultMovement == 1)
    {
        position = new Vector3(x, y, z);
    }
    float currentSpeed = speed;
    if (Input.GetButton("Fire3") && (Mathf.Abs(x) > 0.1f || Mathf.Abs(z) > 0.1f))
    {
        currentSpeed = runningSpeed;
    }
    characterController.Move(position * currentSpeed * Time.deltaTime);
}

private void Turn()
{
    if (Mathf.Abs(position.x) > 0.1 || Mathf.Abs(position.z) > 0.1)
    {
        var targetAngle = Mathf.Atan2(x, z) * Mathf.Rad2Deg;
        transform.rotation=Quaternion.Euler(0, initialRotationY + targetAngle, 0);
    }
}

private float VerticalForceCalculation()
{
    if (characterController.isGrounded)
    {
        veritcalVelocity = -1f;
        if (Input.GetButtonDown("Jump"))
        {
            veritcalVelocity = Mathf.Sqrt(jumpHeight * gravity * 2);
            audioManager.PlaySFX(audioManager.GetJump());
        }
    }
    else
    {
        veritcalVelocity -= gravity * Time.deltaTime;
        if (IsVoid())
        {
            if (!healthManager.IsDead())
            {
                GetComponent<PlayerCollision>().Teleport(0, 2f, 0, true);
            }
        }
    }
    return veritcalVelocity;
}

```

```
private bool IsVoid()
{
    if (transform.position.y < voidHeight)
    {
        return true;
    }
    return false;
}
}
```

Questo script è associato al personaggio del giocatore. Lo scopo è gestire il movimento, la rotazione, il salto, la corsa e il controllo della caduta nel vuoto del personaggio. Utilizza un CharacterController per il movimento fisico e interagisce con altri componenti come AudioManager, HealthManager e PlayerCollision.

Nella funzione Awake()

- Viene recuperato il riferimento all'AudioManager, cercandolo nella scena tramite il tag "Audio". Questo permetterà allo script di riprodurre effetti sonori, come il suono del salto.

Nella funzione Start()

- Viene inizializzato il HealthManager e recuperato il componente CharacterController.
- Viene verificata l'esistenza della chiave "DefaultMovement" nei PlayerPrefs per determinare la modalità di movimento (visuale isometrica o classica). Se non è presente, viene impostata a 1 come valore predefinito.
- In base al valore di defaultMovement, viene settata una rotazione iniziale del personaggio (0° o -45° sull'asse Y), per adattare il movimento a una visuale isometrica.

Nella funzione Update()

Ogni frame il personaggio esegue due operazioni principali:

- Movement() – Gestisce il movimento del personaggio in base all'input da tastiera, alla gravità, e alla corsa.
- Turn() – Ruota il personaggio nella direzione di movimento, adattandosi all'input ricevuto.

Nella funzione Movement()

- Vengono letti gli input orizzontali e verticali.
- Viene attivata o disattivata l'animazione "isWalking" a seconda della presenza di movimento.
- Viene calcolata la forza verticale tramite la funzione VerticalForceCalculation().
- In base al tipo di movimento (defaultMovement), viene determinata la direzione di spostamento:
- defaultMovement == 0: movimento isometrico.
- defaultMovement == 1: movimento classico (in avanti = Z).
- Se il tasto "Fire3" (Shift sinistro di default) è premuto, la velocità viene aumentata (runningSpeed).


Infine, viene applicato lo spostamento con characterController.Move().

Nella funzione Turn()

- Se il personaggio si sta muovendo, viene calcolato l'angolo di rotazione in base all'input (x, z) e applicato alla rotazione del personaggio. Questo permette al giocatore di guardare nella direzione del movimento.

Nella funzione VerticalForceCalculation()

- Se il personaggio è a terra, viene impostata una leggera forza verso il basso. Se viene premuto il tasto "Jump", viene calcolata e applicata una forza verso l'alto per il salto, e viene riprodotto l'audio del salto.
- Se il personaggio è in aria, la gravità viene applicata continuamente.

	SAMT – Sezione Informatica	Pagina 56 di 101
	Colorful Songs	

- Se il personaggio cade sotto un certo livello (voidHeight), viene controllato se è ancora vivo tramite HealthManager. In tal caso, viene teletrasportato in una posizione sicura usando la funzione Teleport() dello script PlayerCollision.

Nella funzione IsVoid()

- Controlla se la posizione Y del personaggio è inferiore al valore definito da voidHeight. In tal caso, restituisce true, indicando che il personaggio è "caduto nel vuoto".

4.12.3 Player Collision

```
using System.Collections;
using System.Runtime.InteropServices;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.UIElements;

public class PlayerCollision : MonoBehaviour
{
    [SerializeField] private GameObject portal;
    private HealthManager healthManager;
    [SerializeField] private PauseMenu pauseMenu;
    [SerializeField] private Score sc;

    void Start()
    {
        healthManager = HealthManager.Instance;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Enemy"))
        {
            healthManager.LooseOneHeart();
        }
    }

    private void OnTriggerEnterStay(Collider other)
    {
        if (other.gameObject.CompareTag("Portal"))
        {
            int x = portal.GetComponent<PortalGenerator>().getEndX();
            int z = portal.GetComponent<PortalGenerator>().getEndZ();
            Teleport(x+1, 0.75f, z, false);
        }

        if (other.gameObject.CompareTag("BadPortal"))
        {
            Teleport(1, 0.75f, 1, true);
        }

        if (other.gameObject.CompareTag("FinishPortal"))
        {
            int CompletedLevels = PlayerPrefs.GetInt("CompletedLevels");
            string timeToComplete = sc.getScore();
            string timeForLevel = "Time" + CompletedLevels;
            CompletedLevels++;
            PlayerPrefs.SetInt("CompletedLevels", CompletedLevels);
            PlayerPrefs.SetString(timeForLevel, timeToComplete);
            PlayerPrefs.Save();
            pauseMenu.Restart();
        }
    }
}
```

```
public async void Teleport(float x, float y, float z, bool damage)
{
    GetComponent<PlayerMovement>().enabled = false;
    await Task.Delay(300);
    transform.position = new Vector3(x, y, z);
    if (damage)
    {
        healthManager.LooseOneHeart();
    }
    await Task.Delay(300);
    GetComponent<PlayerMovement>().enabled = true;
}

IEnumerator Wait(float time)
{
    yield return new WaitForSeconds(time);
}
}
```

Questo script è associato al personaggio del giocatore. Il suo scopo è gestire le collisioni con vari oggetti nella scena, come nemici e portali. Inoltre, contiene una funzione per teletrasportare il giocatore in un'altra posizione, con eventuali effetti collaterali (come la perdita di salute).

Nella funzione Start()

- Viene inizializzato il riferimento al HealthManager, accedendo alla sua istanza singleton (HealthManager.Instance). Questo permette allo script di gestire la salute del giocatore in risposta agli eventi di collisione.

Nella funzione OnTriggerEnter(Collider other)

- Quando il giocatore entra in contatto con un oggetto dotato di collider trigger:
- Se l'oggetto ha il tag "Enemy", il giocatore perde una vita tramite la funzione LooseOneHeart() del HealthManager.

Nella funzione OnTriggerStay(Collider other)

- Questa funzione viene chiamata ogni frame in cui il giocatore rimane dentro un trigger:

Se entra in un oggetto con tag "Portal":

- Viene recuperata la posizione finale del portale (getEndX(), getEndZ()).
- Il giocatore viene teletrasportato a una nuova posizione, senza subire danni.

Se entra in un oggetto con tag "BadPortal":

- Il giocatore viene teletrasportato in una posizione predefinita (1, 0.75, 1) subendo danno (perdita di una vita).

Se entra in un oggetto con tag "FinishPortal":

- Viene incrementato il numero di livelli completati (CompletedLevels).
- Viene salvato il tempo impiegato per completare il livello corrente, recuperato tramite Score.getScore().
- I dati vengono salvati con PlayerPrefs.
- Infine, viene chiamata pauseMenu.Restart() per ricominciare la scena o ripartire dal menu.

Nella funzione Teleport(float x, float y, float z, bool damage)

Questa funzione asincrona consente di spostare il giocatore in una nuova posizione dopo un breve ritardo: Il componente PlayerMovement viene temporaneamente disabilitato per evitare interferenze durante il teletrasporto.

Dopo 300 ms, il personaggio viene spostato nella nuova posizione, se il parametro damage è true, viene tolta una vita al giocatore, dopo un ulteriore ritardo di 300 ms, il movimento del giocatore viene riattivato.

Nella funzione Wait(float time), una coroutine standard che può essere usata per introdurre un ritardo temporale. In questo script, non viene utilizzata direttamente ma potrebbe servire per future espansioni.

4.12.4 HealthManager

```
using System.Collections;
using TMPro;
using UnityEngine;
public class HealthManager : MonoBehaviour
{
    public static HealthManager Instance { get; private set; } // Singleton
    [SerializeField] private GameObject healthBar;
    private static int health; // 0-5
    private static bool isInvincible;
    private float invincibilityDurationSeconds = 2;
    private Transform[] heartImages;
    [SerializeField] private AudioSource musicSource;
    [SerializeField] private GameObject deathGUI;
    [SerializeField] private TextMeshProUGUI winLostGUI;
    [SerializeField] private GameObject hitGUI;
    private AudioManager audioManager;
    [SerializeField] private Score sc;
    void Awake()
    {
        audioManager =
GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject); // Evita duplicati nel caso di più scene
        }
    }

    void Start()
    {
        heartImages = healthBar.GetComponentsInChildren<Transform>();
        health = heartImages.Length - 1;
    }
}
```

```

public void LooseOneHeart()
{
    if (!isInvincible)
    {
        string currentLastHeart = "Heart " + health;
        GameObject.Find(currentLastHeart).SetActive(false);
        audioManager.PlaySFX(audioManager.GetHit());
        health -= 1;

        if (IsDead())
        {
            hitGUI.SetActive(false);
        }
        else
        {
            hitGUI.SetActive(true);
            StartCoroutine(SetInvincible());
        }
    }
}

private IEnumerator SetInvincible()
{
    isInvincible = true;
    yield return new WaitForSeconds(invincibilityDurationSeconds);

    isInvincible = false;
    hitGUI.SetActive(false);
}

public void resetInvincible()
{
    isInvincible = false;
}

public static void EndScreen(string text)
{
    Time.timeScale = 0f;
    Instance.musicSource?.Pause();
    Instance.deathGUI.gameObject?.SetActive(true);
    Instance.winLostGUI.text = string.Format("You {0}", text);
}

public bool IsDead()
{
    return health <= 0;
}
}

```

Questo script gestisce la salute del giocatore durante il gioco, controllando la barra della salute visiva, lo stato di invincibilità temporanea dopo che il giocatore viene colpito, e la schermata di fine partita (vittoria o sconfitta).

All'avvio, nel metodo `Awake()`, lo script imposta un pattern Singleton per assicurare che esista una sola istanza attiva di questo gestore. Se viene rilevata più di una istanza (ad esempio cambiando scena), la seconda viene distrutta per evitare conflitti.

Nel metodo `Start()`, lo script recupera tutti i figli dell'oggetto `healthBar` che rappresentano le immagini dei cuori, e imposta la salute iniziale a un valore pari al numero di cuori meno uno (presumibilmente perché il primo Transform è il contenitore stesso).

Il metodo principale di questo script è `LooseOneHeart()`, che viene chiamato quando il giocatore perde un punto salute:

- Prima verifica che il giocatore non sia invincibile.
- Disattiva il cuore corrispondente alla salute corrente, disattivandolo tramite `GameObject.Find()` con il nome "Heart X".
- Riproduce un effetto sonoro di colpo tramite l'audio manager.
- Decrementa il valore della salute.
- Se il giocatore è morto (salute ≤ 0), nasconde l'interfaccia `hitGUI`.
- Se invece è ancora vivo, attiva l'interfaccia di colpo (`hitGUI`) e attiva una coroutine `SetInvincible()` che rende il giocatore temporaneamente invincibile per 2 secondi, evitando ulteriori perdite immediate di salute.
-

La coroutine `SetInvincible()` imposta `isInvincible` a `true`, attende 2 secondi, quindi lo riporta a `false` e nasconde l'interfaccia `hitGUI`.

Il metodo `resetInvincible()` permette di resettare manualmente lo stato di invincibilità, impostandolo a `false`.

Il metodo statico `EndScreen(string text)` viene chiamato per mostrare la schermata di fine partita, fermando il tempo di gioco (`Time.timeScale = 0`), mettendo in pausa la musica, attivando il pannello `deathGUI` e aggiornando il testo con un messaggio del tipo "You Win" o "You Lost" a seconda del parametro passato. Infine, il metodo `IsDead()` verifica se la salute è scesa a zero o meno, indicando che il giocatore è morto.

4.13 Nemici

4.13.1 Tamburo

```
public class DrumEnemy : MonoBehaviour
{
    private float enemyX;
    private float y;
    private float enemyZ;
    private float startX;
    private float endX;
    private float startZ;
    private float endZ;

    private int p1;
    private int p2 = 0;

    private float p1X;
    private float p1Z;
    private float p2X;
    private float p2Z;

    private int enemyPosition = 0;

    private Vector3 position;
    [SerializeField] private GameObject generator;
    private void Start()
    {
        startX = generator.GetComponent<TerrainGenerator>().getEndX("1")+0.5f;
        endX = generator.GetComponent<TerrainGenerator>().getStartX("2")-2f;
        startZ = 0.5f;
        endZ = generator.GetComponent<TerrainGenerator>().getEndZ("1")-2;
        enemyX = startX-1f + (endX - startX+1.5f) / 2;
        enemyZ = (endZ+2f) / 2;
        y = 0.5f;

        position = new Vector3(enemyX, y, enemyZ);
        StartCoroutine(MovementDrum());
    }
}
```

```
IEnumerator MovementDrum()
{
    float[,] positions = {
        {enemyX,enemyZ},
        {endX,enemyZ},
        {endX,startZ},
        {enemyX,startZ},
        {startX,startZ},
        {startX,enemyZ},
        {startX,endZ},
        {enemyX,endZ},
        {endX,endZ}
    };
    while (true)
    {
        if(enemyPosition == 0)
        {
            do
            {
                p1 = p2;
                p2 = Random.Range(0, 9);
            } while (p1 == p2);

            p1X = positions[p1,0];
            p1Z = positions[p1,1];

            p2X = positions[p2,0];
            p2Z = positions[p2,1];
            enemyPosition++;
        }
        if (enemyPosition == 1)
        {
            position = new Vector3(p1X, y, p1Z);
            enemyPosition++;
        }
        else if (enemyPosition == 2 || enemyPosition == 4)
        {
            float diffZ = p2Z - p1Z;
            float diffX = p2X - p1X;
            position = new Vector3(p1X + diffX/2, y + 2f, p1Z + diffZ / 2);
            if (enemyPosition == 1)
            {
                enemyPosition++;
            }
        }
    }
}
```

```

        else
        {
            enemyPosition = 0;
        }
    }
    else if (enemyPosition == 3)
    {
        position = new Vector3(enemyX, y, startZ);
        enemyPosition++;
    }
    GetComponent<Rigidbody>().transform.position = position;
    yield return new WaitForSeconds(0.5f);
}
}
}

```

Questo script è associato al nemico chiamato DrumEnemy.

Lo script serve per far muovere il nemico su un percorso predefinito all'interno della mappa, generata da un oggetto chiamato generator, che ha uno script chiamato TerrainGenerator.

All'inizio, nello Start(), il nemico calcola la sua posizione iniziale prendendo dei riferimenti dallo script TerrainGenerator.

In particolare, calcola un intervallo sull'asse X e sull'asse Z all'interno del quale potrà muoversi. Vengono usati i metodi getEndX1(), getStartX2() e getEndZ() per determinare questi limiti.

Poi, viene settata la posizione iniziale del nemico, che è circa al centro di quest'area, e viene avviata una coroutine chiamata MovementDrum() che serve per gestire il movimento nel tempo.

Dentro la coroutine MovementDrum():

- Viene definito un array bidimensionale chiamato positions che contiene 9 possibili punti di movimento per il nemico, tutti calcolati in base a startX, endX, startZ, endZ e enemyX.
- All'interno del ciclo while(true) (quindi che gira per sempre), il nemico esegue un movimento ogni 0.5 secondi.
- Se enemyPosition è 0, allora vengono scelti due punti casuali tra i 9 della lista (p1 e p2), assicurandosi che siano diversi. Questi due punti rappresentano il punto di partenza e quello di arrivo.
- Quando enemyPosition è 1, il nemico si sposta sul primo punto (p1).
- Quando è 2 o 4, il nemico si sposta in un punto intermedio tra p1 e p2, ma alzandosi anche di 2 sull'asse Y, come se facesse un piccolo salto o un movimento verticale.
- Quando enemyPosition è 3, si muove su una posizione fissa lungo l'asse Z.
- Dopo ogni spostamento, la posizione viene aggiornata usando il componente Rigidbody, e viene aspettato mezzo secondo prima di continuare.

4.13.2 Trombetta

```
public class TrumpetEnemy : MonoBehaviour
{
    private float enemyX;
    private float y;
    private float enemyZ;

    [Header("Prefabs")]
    [SerializeField] private GameObject enemyPrefab;
    [SerializeField] private GameObject bulletPrefab;

    [Header("Spawn settings")]
    [SerializeField] private Transform parent;

    [Header("FirePoint Offset")]
    [SerializeField] private Vector3 firePointOffset = new Vector3(0, 0, 1);

    [Header("Shooting Settings")]
    [SerializeField] private float rotationSpeed = 90f;
    [SerializeField] private float timeBetweenShots = 0.1f;
    [SerializeField] private float bulletSpeed = 5f;
    [SerializeField] private float bulletLifetime = 3f;

    private GameObject enemyInstance;
    private float shootTimer = 0f;
    private List<GameObject> bullets = new List<GameObject>();
    private Vector3 enemySpawnPosition = Vector3.zero;
    [SerializeField] private GameObject generator;

    void Start()
    {
        float startX = generator.GetComponent<TerrainGenerator>().getEndX("1");
        float endX = generator.GetComponent<TerrainGenerator>().getStartX("2");
        float endZ = generator.GetComponent<TerrainGenerator>().getEndZ("1");
        enemyX = startX + (endX - startX) / 2;
        enemyZ = endZ / 2;
        y = 1.2f;
        enemySpawnPosition = new Vector3(enemyX, y, enemyZ);
        enemyInstance = Instantiate(enemyPrefab, enemySpawnPosition,
Quaternion.identity);
        if (parent != null) enemyInstance.transform.SetParent(parent);
    }

    void Update()
    {
        if (enemyInstance == null) return;
    }
}
```

```

        enemyInstance.transform.Rotate(Vector3.up, rotationSpeed *
Time.deltaTime);

        shootTimer -= Time.deltaTime;
        if (shootTimer <= 0f)
        {
            ShootBullet();
            shootTimer = timeBetweenShots;
        }

        foreach (var bullet in bullets)
        {
            if (bullet != null)
            {
                bullet.transform.Translate(bullet.transform.forward * bulletSpeed
* Time.deltaTime, Space.World);
            }
        }
    }

    void ShootBullet()
    {
        Vector3 firePointWorldPos = enemyInstance.transform.position +
enemyInstance.transform.TransformDirection(firePointOffset);

        GameObject bullet = Instantiate(bulletPrefab, firePointWorldPos,
enemyInstance.transform.rotation);
        if (parent != null) bullet.transform.SetParent(parent);

        bullets.Add(bullet);
        Destroy(bullet, bulletLifetime);
    }
}

```

Questo script è associato al nemico chiamato TrumpetEnemy. Lo scopo di questo script è creare un nemico che ruota su sé stesso e spara proiettili a intervalli regolari. Anche in questo caso viene utilizzato lo script TerrainGenerator per calcolare dove posizionare il nemico sulla mappa.

Nella funzione Start(), all'inizio dello script, il nemico calcola la sua posizione centrale sulla mappa, utilizzando le coordinate ottenute da TerrainGenerator, in particolare getEndX1(), getStartX2() e getEndZ(). Dopo aver calcolato la posizione (enemySpawnPosition), viene istanziato il nemico tramite un prefab (enemyPrefab) e posizionato nella scena. Se è stato assegnato un oggetto padre (parent), il nemico viene attaccato ad esso nella gerarchia.

Nella funzione Update()

Ogni frame il nemico esegue due azioni principali:

1. Ruota su sé stesso usando transform.Rotate() sull'asse Y. Questo crea un effetto di rotazione continua, come una torretta che si muove su 360°.

2. Spara proiettili a intervalli regolari.

Il timer shootTimer viene decrementato ogni frame.

Quando arriva a 0, viene chiamata la funzione ShootBullet() e il timer viene resettato al valore di timeBetweenShots.

Oltre a questo, viene anche gestito il movimento dei proiettili: per ogni proiettile nella lista bullets, se non è stato distrutto, viene spostato in avanti rispetto alla sua direzione, con una velocità impostata da bulletSpeed.

Nella funzione ShootBullet()

Viene calcolata la posizione da cui sparare il proiettile (chiamata firePointWorldPos). Questa posizione è leggermente davanti al nemico, grazie all'offset definito in firePointOffset.

- Viene creato un nuovo proiettile (bulletPrefab) in quella posizione, con la stessa rotazione del nemico.
- Se è stato assegnato un parent, anche il proiettile viene inserito sotto di esso nella gerarchia.
- Il proiettile viene aggiunto alla lista bullets per poter essere gestito nel tempo.
- Infine, viene distrutto automaticamente dopo un tot di secondi, in base a bulletLifetime.

4.14 Global Volume (Effetti Visivi)

Il Global Volume in Unity serve per aggiungere effetti visivi a tutta la scena, migliorando l'aspetto grafico indipendentemente dalla posizione della telecamera. Fa parte dei sistemi grafici URP o HDRP e contiene le impostazioni che definiscono quali effetti sono attivi.

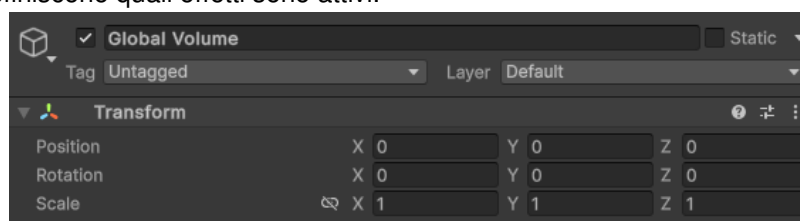


Figura 25: Global Volume

Tra questi effetti, il Tonemapping regola luci e colori per adattare immagini HDR agli schermi normali, evitando zone troppo chiare o scure e rendendo la scena più realistica.

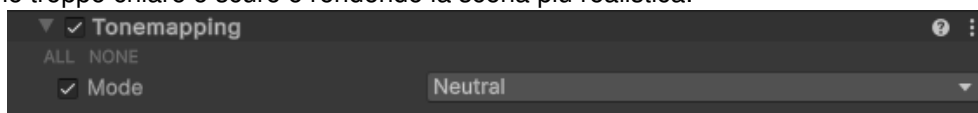


Figura 26: Global Volume (Tonemapping)

Il Bloom crea un bagliore morbido attorno alle parti più luminose, con Threshold a 0.9 per far brillare solo le luci intense e Intensity a 1 per un effetto equilibrato e naturale.

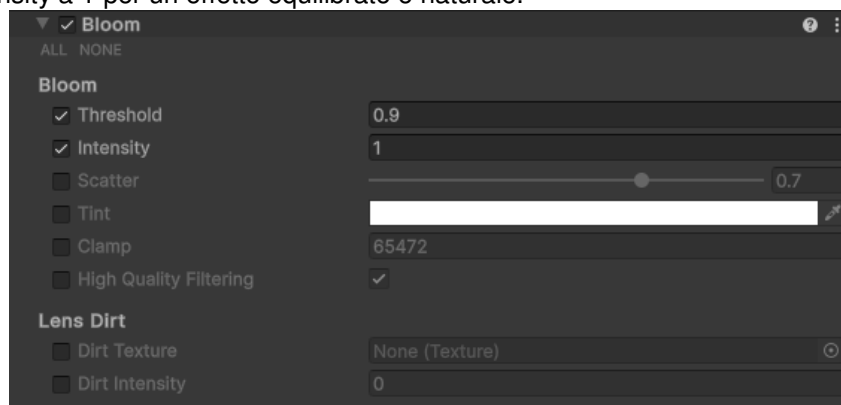


Figura 27: Global Volume (Bloom)

La Panini Projection corregge la distorsione prospettica, soprattutto con campi visivi ampi, mantenendo dritte le linee verticali ai bordi e riducendo l'effetto "occhio di pesce"; con Distance a 0.29 la correzione è leggera e mantiene la profondità.

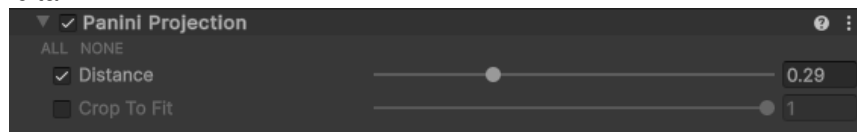


Figura 28: Global Volume (Panini Projection)

La Vignette scurisce i bordi dell'immagine con un colore nero, creando un'ombra morbida che focalizza l'attenzione al centro; con intensità e morbidezza a 0.43 l'effetto è moderato e dona un tono più cinematografico o drammatico.

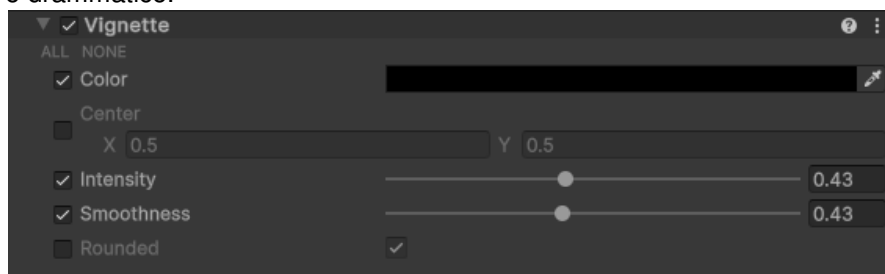


Figura 29: Global Volume (Vignette)

La Chromatic Aberration simula un difetto ottico separando leggermente i colori ai bordi, con intensità 0.15 per un tocco lieve di realismo o stile senza essere fastidioso.



Figura 30: Global Volume (Chromatic Aberration)

Infine, il Film Grain aggiunge una grana simile a quella delle pellicole fotografiche per un aspetto vintage o realistico. Utilizzando una texture personalizzata (4x-scanlines-1920x960-ver2), con intensità 0.08 la grana è molto leggera, e la Response 1 fa sì che reagisca uniformemente alla luce, mantenendo un effetto naturale.

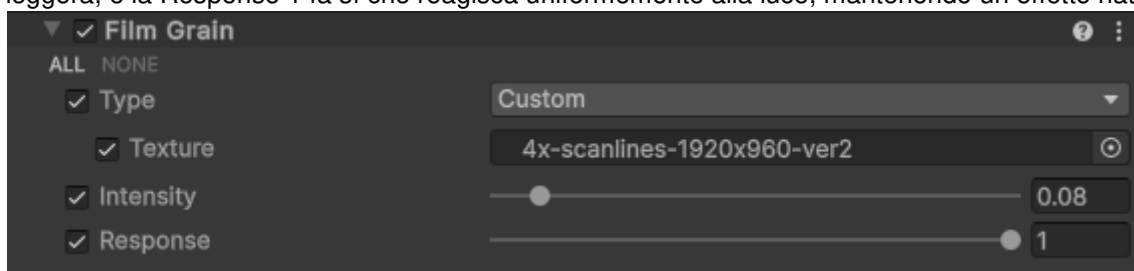


Figura 31: Global Volume (Film Grain)

4.15 Leaderboard

Il progetto utilizza MVC e eloquent come ORM per fare le query al database.

Questa pagina mostra tutti gli utenti con il loro rispettivo punteggio. Da questa pagina è possibile aggiungere nuovi amici, filtrare tra il punteggio degli amici e quello di tutti gli altri utenti e cercare una persona specifica.

4.15.1 Funzione index

Questa funzione mostra i dati nella home page.

```
public function index()
{
    if (isset($_SESSION['userType'])) {
        require_once 'application/models/Leaderboard.php';
        $leaderboard_data = Leaderboard::getData();
        $friends = Friend::showFriend($_SESSION['UserId']);
        $friendIds = [];
        foreach ($friends as $friend) {
            if ($friend->userId1 == $_SESSION['UserId']) {
                $friendIds[] = $friend->userId2;
            } else {
                $friendIds[] = $friend->userId1;
            }
        }

        if (isset($_SESSION['type'])) {
            $_SESSION['type'] = $_POST['type'];
        } else {
            $_SESSION['type'] = "global";
        }
        require_once 'application/views/_templates/header.php';
        if ($this->isAdmin()) {
            require_once 'application/views/admin/index.php';
        } else {
            require_once 'application/views/leaderboard/index.php';
        }
    }
    else {
        header("Location:" . URL);
    }
}
```

In questa funzione vengono presi tutti i dati dalla tabella leaderboard e vengono passati alla variabile leaderboard_data che poi li mostrerà nella view.

La variabile friends contiene tutti gli amici dell'utente che ha effettuato l'accesso alla pagina. Con il foreach viene passato ogni utente nella tabella friend e inserito nell'array friendIds, che viene riempito solo se viene trovato l'id dell'utente che ha effettuato l'accesso alla pagina. Questo array serve poi nella view per la gestione del pulsante per inviare una richiesta di amicizia.

nel resto della funzione vengono richiamate poi le view da mostrare all'utente.

4.15.2 Funzione radioFilter

Questa funzione prende il valore dei radio button nella view. In base al valore passato (global o friend) cambia pagina e esegue una query diversa

```
public function radioFilter()
{
    if (isset($_SESSION['userType'])) {
        require_once 'application/models/Leaderboard.php';
        $leaderboardMapper = Leaderboard::getData();
        if (isset($_POST['type'])) {
            $friends = Friend::showFriend($_SESSION['UserId']);
            $friendIds = [];
            foreach ($friends as $friend) {
                if ($friend->userId == $_SESSION['UserId']) {
                    $friendIds[] = $friend->userId;
                } else {
                    $friendIds[] = $friend->userId;
                }
            }

            if ($_POST['type'] == 'global') {
                $leaderboard_data = Leaderboard::getData();

                $checked = "global";
                $_SESSION['type'] = $checked;
                if ($this->isAdmin()) {
                    require_once 'application/views/_templates/header.php';
                    require_once 'application/views/admin/index.php';
                } else {
                    require_once 'application/views/_templates/header.php';
                    require_once 'application/views/leaderboard/index.php';
                }
            } else if ($_POST['type'] == 'friend') {
                $leaderboard_data =
                User::getDataByFriendId($_SESSION["UserId"]);

                $checked = "friend";
                $_SESSION['type'] = $checked;
                if ($this->isAdmin()) {
                    require_once 'application/views/_templates/header.php';
                    require_once 'application/views/admin/index.php';
                } else {
                    require_once 'application/views/_templates/header.php';
                    require_once 'application/views/leaderboard/index.php';
                }
            }
        } else {
            header("Location:" . URL);
        }
    }
}
```

Inizialmente viene eseguito lo stesso controllo che viene eseguito anche nella funzione index, cioè vengono controllati tutti gli amici dell'utente.

Dopo di che viene controllato se il valore passato dal radio button è global o friend. Nel caso in cui il valore sia global viene richiamata la funzione getData(), vedi funzionamento al capitolo [4.8.1.5](#). Nel caso in cui

invece il valore passato sia friend allora viene richiamata la funzione `getDataByFriendId()`, vedi funzionamento al capitolo [4.8.1.6](#). la variabile `checked` serve per passare il valore del radio button di nuovo alla view così da mostrare all'utente quale leaderboard sta visualizzando.

4.15.3 Funzione `searchFilter`

Questa funzione permette di eseguire delle ricerche per nome tra gli utenti.

```
public function searchFilter()
{
    if (isset($_SESSION['userType'])) {
        require_once "application/models/Leaderboard.php";
        if (isset($_POST['search'])) {
            $username = $this->validator->sanitizeInput($_POST['usernameSearch']);
            $username = $this->validator->checkTextArea($username);
            if (is_null($username)) {
                require_once 'application/views/_templates/header.php';
                if ($this->isAdmin()) {
                    require_once 'application/views/admin/index.php';
                } else {
                    require_once 'application/views/leaderboard/index.php';
                }
            }
            $leaderboard_data = Leaderboard::getDataByUsername($username);
            $friends = Friend::showFriend($_SESSION['UserId']);
            $friendIds = [];
            foreach ($friends as $friend) {
                if ($friend->userId == $_SESSION['UserId']) {
                    $friendIds[] = $friend->userId2;
                } else {
                    $friendIds[] = $friend->userId1;
                }
            }
            require_once 'application/views/_templates/header.php';
            if ($this->isAdmin()) {
                require_once 'application/views/admin/index.php';
            } else {
                require_once 'application/views/admin/index.php';
            }
        }
        if (isset($_POST['deleteFilter'])) {
            require_once "application/models/Leaderboard.php";
            require_once 'application/views/_templates/header.php';
            $friends = Friend::showFriend($_SESSION['UserId']);
            $friendIds = [];
            foreach ($friends as $friend) {
                if ($friend->userId == $_SESSION['UserId']) {
                    $friendIds[] = $friend->userId2;
                } else {
                    $friendIds[] = $friend->userId1;
                }
            }
            if ($this->isAdmin()) {
                if ($_SESSION['type'] == 'friend') {
                    $leaderboard_data =
```

```
User::getDataByFriendId($_SESSION["UserId"]);
    } elseif ($_SESSION['type'] == 'global') {
        $leaderboard_data = Leaderboard::getData();
    }
    require_once 'application/views/admin/index.php';
} else {
    if ($_SESSION['type'] == 'friend') {
        $leaderboard_data =
User::getDataByFriendId($_SESSION["UserId"]);
    } elseif ($_SESSION['type'] == 'global') {
        $leaderboard_data = Leaderboard::getData();
    }
    require_once 'application/views/leaderboard/index.php';
}
}
} else {
    header("Location:" . URL);
}
}
```

Come prima cosa viene controllato l'input tramite la funzione sanitizelInput, questa funzione elimina gli spazi all'inizio e alla fine della stringa passata, poi elimina i backslash e infine rimuove tutti i caratteri speciali dalla stringa.

Ora richiamo la funzione getDataByUsername(), vedi funzionamento al capitolo [4.8.1.7](#).

Per eliminare la ricerca controllo che il pulsante 'deleteFilter' sia stato premuto e abbia mandato il valore. Se il pulsante è stato premuto allora vado a controllare che radio button è premuto, e in base al valore di esso richiamo le funzioni getData() o getDataByFriendId().

4.15.4 Funzione showFriend

Questa è la funzione che esegue la query per trovare tutti gli amici di un determinato utente.

```
public static function showFriend($userId)
{
    return self::select('friend.userId1', 'friend.userId2', 'friend.pending')
        ->distinct()
        ->where('friend.pending', '=', 0)
        ->where('friend.userId1', '=', $userId)
        ->orWhere('friend.userId2', '=', $userId)
        ->get();
}
```

Questa è una query effettuata con eloquent ORM. In questo caso si vogliono avere lo userId1, userId2 e lo stato della richiesta di amicizia, in base all'id della persona loggata nella pagina. Quindi va a controllare che lo stato della richiesta (pending) sia 0, questo significa che la richiesta è stata accettata, e che l'id della persona sia o nella colonna userId1 o in userId2.

4.15.5 Funzione getData

Questa funzione è una query che va a prendere i valori da due tabelle e li mette a disposizione per poi mostrarli all'utente.

```
public static function getData()
{
    return self::select('user.id','user.username as username',
        'leaderboard.score')
        ->distinct()
        ->join('user', 'leaderboard.user_id', '=', 'user.id')
        ->orderBy('leaderboard.score', 'ASC')
        ->get();
}
```

In questo caso si vuole avere l'id dell'utente dalla tabella user, lo username dell'utente (valore che è nella tabella user) e il punteggio dell'utente. Per questo motivo viene eseguita una join tra le due tabelle, la join viene eseguita tra i campi user_id della tabella leaderboard e il campo id della tabella user, dopo di che i dati vengono ordinati in ordine ascendente in base al punteggio.

4.15.6 Funzione getDataByFriendId

Questa query serve per ritornare tutti gli amici della persona che ha fatto l'accesso al sito.

```
public static function getDataByFriendId($userId)
{
    return self::select('u.username as username', 'l.score', 'u.id as id')
        ->distinct()
        ->join('friend as f', 'f.userId1', '=', 'user.id')
        ->join('user as u', 'f.userId2', '=', 'u.id')
        ->join('leaderboard as l', 'f.userId2', '=', 'l.user_id')
        ->where('f.userId1', '=', $userId)
        ->where('f.pending', '=', 0)
        ->orderBy('l.score', 'ASC')
        ->get();
}
```

Qui si vuole avere lo username dell'utente, il punteggio e l'id dell'utente. Però al contrario della funzione getData() in questo caso i dati devono essere degli amici dell'utente. Quindi per far sì di avere solo i dati degli amici bisogna fare una join anche alla tabella friend. Quindi la join alla tabella friend viene fatto tramite lo userId1 della tabella friend e l'id della tabella user, dopo di che c'è un'altra join tra user e friend però viene fatta tra la colonna userId2 di friend e la colonna id di user. Dopo di che c'è la join tra la tabella leaderboard e la tabella friend, qui la join è fatta tramite le colonne userId2 e user_id. Ora viene controllato che lo userId1 della tabella friend corrisponda all'id che viene passato come variabile, poi viene controllato che pending sia 0 e infine viene ordinato in maniera ascendente per il punteggio.

4.15.7 Funzione getDataByUsername

Questa query ritorna un utente, nel caso in cui venga scritto il nome per intero, o tutti gli utenti che contengono una determinata sequenza di caratteri (es: 'eli', ritorna tutti gli utenti che contengono 'eli' nel loro nome)

```
public static function getDataByUsername($username)
{
    return self::select('user.username as username', 'leaderboard.score')
        ->distinct()
        ->join('user', 'leaderboard.user_id', '=', 'user.id')
        ->where('user.username', 'like', "%$username%")
        ->orderBy('leaderboard.score', 'ASC')
        ->get();
}
```

In questa query si vuole avere lo username e il punteggio, quindi viene fatta una join tra le tabelle user e leaderboard. Dopo di che viene controllato tramite l'operatore 'LIKE' e i simboli '%' che lo username contenga la stringa passata, e infine viene ordinato in ordine ascendente in base al punteggio.

4.15.8 Funzione newFriend

Questa funzione manda una richiesta di amicizia a un altro utente

```
public function newFriend()
{
    if (isset($_SESSION['userType'])) {
        $friends = Friend::friendRequestWithFriendId($_SESSION['UserId']);
        require_once 'application/views/_templates/header.php';
        require_once 'application/views/notifications/index.php';
    } else {
        header("Location: " . URL);
    }
}
```

Viene richiamata la funzione friendRequestWithFriendId che è una query per mandare la richiesta all'altro utente.

4.15.9 Funzione friendRequestWithFriendId

Questa query controlla se ci sono delle richieste di amicizia in sospeso.

```
public static function friendRequestWithFriendId($userId)
{
    return self::select('user.username as username', 'friend.userId2',
        'friend.userId1')
        ->distinct()
        ->join('user', 'friend.userId1', '=', 'user.id')
        ->where('friend.userId2', '=', $userId)
        ->where('friend.pending', '=', 1)
        ->get();
}
```

In questo caso si vuole avere lo username, lo userId1 e lo userId2, poi viene fatta una join tra le tabelle user e friend tramite le colonne userId1 e id. Viene comparato lo userId2 con lo userId che viene passato come parametro e infine viene controllato che il parametro pending sia a 1, così da avere solo le richieste in sospeso che riguardano l'utente specifico.

4.15.10 Struttura della tabella

Questa è la struttura di come viene generata la tabella

```
<div class="container-md bg-dark bg-opacity-25 rounded-3 pt-4 pb-1">
  <table class="table table-bordered">
    <thead>
      <tr>
        <th>Username</th>
        <th>High Score</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <?php foreach ($leaderboard_data as $leaderboardValue): ?>
        <?php
        $isFriend = in_array($leaderboardValue->id, $friendIds);
        $alreadyFriend = $isFriend ? "Remove Friend" : "Send Friend
Request";
        ?>
        <tr>
          <td><?php echo $leaderboardValue->username ?></td>
          <td><?php echo $leaderboardValue->score ?></td>
          <td>
            <?php if ($isFriend): ?>
              <a href="<?php echo URL .
'friendController/removeFriend/' . $leaderboardValue->id; ?>" class="btn btn-
outline-warning" ><?php echo $alreadyFriend?></a>
              <?php else: ?>
              <a href="<?php echo URL .
'friendController/friendRequest/' . $leaderboardValue->id; ?>" class="btn btn-
outline-info"><?php echo $alreadyFriend?></a></td>
            <?php endif; ?>
          </td>
        </tr>
      <?php endforeach; ?>
    </tbody>
  </table>
</div>
```

Per la generazione della tabella si parte con un foreach della variabile `leaderboard_data`, che è la variabile che contiene tutti i dati ricevuti dalle query. Dopo di che si va a vedere se nella nuova variabile `leaderboardValue`, che ora contiene i valori di `leaderboard_data`, il valore dell'id è presente nell'array di id. Se l'id è presente viene inserito nella variabile `alreadyFriend` "Remove Friend", sennò viene inserito "Send Friend Request". Questi valori verranno poi stampati nel pulsante, e in base a che stringa è presente il pulsante manderà una richiesta o rimuoverà l'amicizia. Ora viene estrapolato il nome utente dalla variabile `leaderboardValue` e nella colonna dopo della tabella verrà inserito il punteggio dell'utente.

4.16 API

Questa API serve per permettere di gestire i dati nel database da parte del gioco, quindi tramite questa API è possibile creare un nuovo utente, aggiornare lo score, inserire un nuovo score, eliminare l'utente o avere i dati di un utente specifico.

4.16.1 UserController

Questa classe serve per gestire le richieste, in base all'header della richiesta viene eseguito un metodo diverso.

```
public function processRequest()
{
    switch($this->requestMethod) {
        case 'GET':
            if($this->userId) {
                $response = $this->getUser($this->userId);
            }elseif($this->username) {
                $response = $this->getUserFromUsername($this->username);
            }
            break;
        case 'POST':
            $response = $this->createFromRequest();
            break;
        case 'PUT':
            $response = $this->updateScoreFromRequest($this->userId);
            break;
        case 'DELETE':
            $response = $this->deleteUser($this->userId);
            break;
        default:
            $response = $this->notFoundResponse();
            break;
    }
    header($response['status_code_header']);
    if ($response['body']) {
        echo $response['body'];
    }
}
```

Questo è lo switch che gestisce le richieste, nel caso in cui l'header della richiesta sia GET va poi a vedere se viene passato come parametro l'id o lo username. Se la richiesta è in POST allora va a creare l'utente o va a creare lo score. La gestione viene poi fatta nella funzione insert (vedi capitolo [4.9.2.2](#)). Nel caso in cui l'header sia PUT allora viene modificato il punteggio dell'utente. E per ultimo nel caso l'header sia DELETE viene eliminato l'utente con l'id specifico.

4.16.2 UserGateway

Questa classe contiene tutti i metodi per eseguire le query al database. Infatti tutti i metodi che vengono richiamati dallo UserController poi richiamano i metodi di questa classe per eseguire le query.

```
public function insert(Array $input)
{
    if (isset($input['username']) && isset($input['password']) &&
    isset($input['email'])) {
        $statement = "
            insert into user
                (username, password, email)
            values
                (:username, :password, :email)";

        try{
            $statement = $this->db->prepare($statement);
            $statement->execute(array(
                'username' => $input['username'],
                'password' => password_hash($input['password'],
                PASSWORD_DEFAULT),
                'email' => $input['email']
            ));
            return $statement->rowCount();
        }catch(PDOException $e){
            $this->logger->error("insert user: " . $e->getMessage());
        }
    }elseif(isset($input['score']) && isset($input['user_id'])){
        $statement = "
            insert into leaderboard
                (score, user_id)
            values
                (:score, :user_id)";

        try{
            $statement = $this->db->prepare($statement);
            $statement->execute(array(
                'score' => $input['score'],
                'user_id' => $input['user_id'],
            ));
            return $statement->rowCount();
        }catch(PDOException $e){
            $this->logger->error("insert score: " . $e->getMessage());
        }
    }
}
```

Questo è il metodo di insert, e è anche il metodo più complesso, questo perché questo metodo permette sia di creare un utente, sia un punteggio. Quindi viene controllato se vengono passati i parametri username e password, se vengono passati allora viene creato l'utente, se non vengono passati allora si controlla che vengano passati il punteggio e lo user_id, anche qui se vengono passati allora viene creato il punteggio.

4.16.3 Index

Questa è la classe dove viene controllato l'url che viene passato all'API.

```
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");

$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$uri = explode('/', $uri);

if ($uri[3] !== 'user') {
    header('HTTP/1.1 404 Not Found');
    exit();
}

$userId = null;
$username = '';
if (isset($uri[4])) {
    try {
        $userId = (int) $uri[4];
    } catch (\Exception $e) {
        $logger->info($e->getMessage());
    }
    $username = $uri[4];
}

$requestMethod = $_SERVER['REQUEST_METHOD'];

$controller = new
UserController($dbConnection, $requestMethod, $userId, $username);
$controller->processRequest();
```

Il primo header() permette l'accesso all'API da qualsiasi origine. Il secondo header() imposta il tipo di codifica dei caratteri, in questo caso UTF-8, e permette di inviare e ricevere file JSON. Il terzo header() dichiara quali metodi sono permessi, in questo caso sono permessi i metodi: GET, POST, PUT, DELETE, OPTIONS. Il quarto header() imposta la durata di vita della richiesta, se dopo 3600 secondi non c'è una risposta la richiesta viene abortita. L'ultimo header() serve per autorizzare tutti gli header precedenti. Dopo aver impostato gli header viene suddiviso l'url, e visto che l'url di base è "API/public/user/" viene controllato che alla terza posizione dell'url ci sia l'endpoint /user, se c'è allora la richiesta va avanti, se non c'è viene restituita una pagina di 404 Not Found. Poi visto che dopo /user ci può essere sia un id sia un nome, viene controllato che sia un numero o una stringa. Dopo di che viene creato un nuovo UserController e viene richiamato il metodo processRequest() ([4.9.2.1](#)).

5 Test

5.1 Protocollo di test

Test Case:	TC-001	Nome:	Gestione utenti
Riferimento:	REQ-001		
Descrizione:	Avere una gestione utenti accessibile dalla pagina Admin, questo comprende aggiunta/modifica/eliminazione dei dati.		
Prerequisiti:	Il database ColorfulSongs accessibile. Pagina Admin accessibile.		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio browser. 2. Andare all'indirizzo della pagina Admin. 3. Fare l'accesso come account Admin. 4. Aggiungere/modificare/eliminare dei dati. 		
Risultati attesi:	La pagina di Admin è raggiungibile, il login viene effettuato con successo e si possono eseguire tutte le azioni.		

Test Case:	TC-002	Nome:	Effettuare il login in game corretto
Riferimento:	REQ-002		
Descrizione:	Poter effettuare il login all'interno del gioco stesso.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Il database ColorfulSongs accessibile. 2. Il gioco deve comunicare con il servizio API. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio gioco. 2. Sul menù principale cliccare il tasto di Login. 3. Eseguire il login con le credenziali valide. 4. Reindirizzamento al menu principale, con un feedback che si è loggati all'interno del proprio account. 		
Risultati attesi:	Quando si apre il gioco, si deve poter cliccare sul pulsante che permette il Login, le credenziali devono essere verificate tramite l'API, effettuato il login si deve essere reindirizzati al menu principale e avere un feedback che si è loggato all'interno dell'account.		

Test Case:	TC-003	Nome:	Effettuare il login in game sbagliato
Riferimento:	REQ-002		
Descrizione:	Controllare la gestione degli errori del login all'interno del gioco stesso.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Il database ColorfulSongs accessibile. 2. Il gioco deve comunicare con il servizio API. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio gioco. 2. Sul menu principale cliccare il tasto di Login. 3. Eseguire il login con le credenziali errate. 4. Feedback di errore delle credenziali 		
Risultati attesi:	Il gioco deve ritornare un feedback il quale riporti un errore nelle credenziali		

Colorful Songs

Test Case:	TC-004	Nome:	Effettuare la registrazione corretta
Riferimento:	REQ-003		
Descrizione:	Poter effettuare la registrazione all'interno del gioco stesso.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Il database ColorfulSongs accessibile. 2. Il gioco deve comunicare con il servizio API. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio gioco. 2. Sul menu principale cliccare il tasto di Register. 3. Eseguire la registrazione con le credenziali valide. 4. Reindirizzamento al menu principale, con un feedback che l'account è stato creato correttamente 		
Risultati attesi:	Quando si apre il gioco, si deve poter cliccare sul pulsante che permette la registrazione, le credenziali devono essere verificate tramite l'API, effettuata la registrazione si deve essere reindirizzati al menu principale e avere un feedback che l'account è stato creato.		

Test Case:	TC-005	Nome:	Visualizzazione corretta Leaderboard Web
Riferimento:	REQ-004		
Descrizione:	Visualizzare la classifica tramite il proprio browser.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Pagina Leaderboard accessibile. 2. Aver effettuato il Login sulla pagina Leaderboard. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio browser. 2. Andare all'indirizzo della pagina Leaderboard. 3. Eseguire un login con credenziali corrette 4. Consultare la Leaderboard. 		
Risultati attesi:	La pagina della Leaderboard è raggiungibile, il login viene effettuato con successo e si può visualizzare la Leaderboard.		

Test Case:	TC-006	Nome:	Visualizzazione negata Leaderboard Web
Riferimento:	REQ-004		
Descrizione:	tentare di visualizzare la classifica tramite il proprio browser senza essere loggati.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Pagina Leaderboard accessibile. 2. Aver effettuato il Login sulla pagina Leaderboard. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio browser. 2. Andare all'indirizzo della pagina Leaderboard. 3. Eseguire un login con credenziali errate 4. Feedback errore 		
Risultati attesi:	Viene ritornato all'utente che le credenziali sono errate e deve ritentare il login		

Colorful Songs

Test Case:	TC-007	Nome:	Login nella Leaderboard Web corretto
Riferimento:	REQ-005		
Descrizione:	Eseguire un login con delle credenziali corrette		
Prerequisiti:	<ol style="list-style-type: none"> 1. Pagina Login Leaderboard accessibile. 2. Il database ColorfulSongs 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio browser 2. Andare all'indirizzo della pagina di Login Leaderboard 3. Effettuare un login con credenziali valide 4. Reindirizzamento alla pagina Leaderboard. 		
Risultati attesi:	Si viene reindirizzati correttamente alla pagina della leaderboard e le proprie credenziali vengono salvate nella sessione.		

Test Case:	TC-008	Nome:	Login nella Leaderboard Web errato
Riferimento:	REQ-005		
Descrizione:	Eseguire un login con delle credenziali sbagliate		
Prerequisiti:	<ol style="list-style-type: none"> 1. Pagina Login Leaderboard accessibile. 2. Il database ColorfulSongs 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio browser 2. Andare all'indirizzo della pagina di Login Leaderboard 3. Effettuare un login con credenziali errate 4. Feedback di errore. 		
Risultati attesi:	All'utente viene ritornato l'errore che le credenziali sono errate.		

Test Case:	TC-009	Nome:	Creare ostacoli
Riferimento:	REQ-006		
Descrizione:	I diversi script di gioco creano degli ostacoli che saranno presenti nella mappa di gioco.		
Prerequisiti:	Creare gli script degli ostacoli.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco 2. Premere il pulsante "Start" per avviare una nuova partita. 3. Nella stanza si generano casualmente gli ostacoli 		
Risultati attesi:	Quando si gioca in ogni stanza saranno presenti degli ostacoli generati casualmente.		

Test Case:	TC-0010	Nome:	Creazione del dungeon casuale
Riferimento:	REQ-007		
Descrizione:	Il dungeon deve essere creato casualmente ad ogni nuova partita per permettere un'esperienza diversa dalla precedente.		
Prerequisiti:	Creare uno script che permetta la creazione casuale di stanze con integrazioni al REQ-006 ovvero gli ostacoli.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere il pulsante "Start" per avviare una nuova partita. 		
Risultati attesi:	La stanza di gioco in cui si gioca è casuale e sarà diversa della prossima.		

Colorful Songs

Test Case:	TC-011	Nome:	Salvataggio dell'highscore
Riferimento:	REQ-008		
Descrizione:	Completata la propria partita, se è il punteggio stabilito nel dungeon è più alto del precedente viene salvato come highscore.		
Prerequisiti:	1. Il database ColofulSongs.		
Procedura:	1. Avviare il gioco. 2. Completare la partita		
Risultati attesi:	Se il giocatore ha completato il dungeon in un tempo inferiore rispetto ai precedenti tentativi viene salvato il suo nuovo Highscore, visualizzabile nella leaderboard.		

Test Case:	TC-012	Nome:	Tutorial Testuale
Riferimento:	REQ-009		
Descrizione:	Nel menu principale, sotto le impostazioni, avere una pagina che presenta i comandi al giocatore e una breve descrizione del gioco.		
Prerequisiti:	1. Possedere il gioco.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, andare sotto le opzioni, comandi. 3. Visualizzare i comandi con le loro spiegazioni.		
Risultati attesi:	All'interno delle impostazioni di gioco è presente un tutorial per come giocare.		

Test Case:	TC-013	Nome:	Aggiungere amici
Riferimento:	REQ-010		
Descrizione:	Dal menu principale del gioco, sarà possibile inviare richieste di amicizia ad altri giocatori tramite il loro ID utente.		
Prerequisiti:	Possedere il gioco.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, inviare la richiesta di amicizia ad un amico tramite il suo ID utente. 3. Se l'amico vuole, può accettare o rifiutare la richiesta di amicizia, avere un feedback in entrambi i casi.		
Risultati attesi:	Il giocatore tramite l'ID del proprio amico potrà mandare una richiesta di amicizia, che dovrà poi essere accettata o rifiutata dal proprio amico, in entrambi i casi si avrà un feedback.		

Test Case:	TC-014	Nome:	Giocare con tastiera e/o controller
Riferimento:	REQ-011		
Descrizione:	Il gioco deve essere giocabile sia da una tastiera sia da un controller.		
Prerequisiti:	Avere una tastiera e/o controller.		
Procedura:	1. Avviare il gioco. 2. Giocare tramite una tastiera e/o controller.		
Risultati attesi:	Che si giochi con tastiera o controller il gioco deve funzionare		

Colorful Songs

Test Case:	TC-015	Nome:	Visualizzazione menu principale.
Descrizione:	All'avvio dell'applicazione, visualizzare il menu principale.		
Prerequisiti:	Possedere il gioco.		
Procedura:	1. Avviare il gioco.		
Risultati attesi:	Quando si avvia il gioco, si visualizza il menu principale.		

Test Case:	TC-016	Nome:	Pulsante Start
Descrizione:	Dal menu principale del gioco, sarà possibile iniziare a giocare tramite il pulsante "Start".		
Prerequisiti:	TC-015 funzionante.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Start" e iniziare la partita.		
Risultati attesi:	La partita inizia.		

Test Case:	TC-017	Nome:	Pulsante Options
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare il menu opzioni tramite il pulsante "Options".		
Prerequisiti:	TC-015 funzionante.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni.		
Risultati attesi:	Il menu opzioni viene mostrato.		

Test Case:	TC-018	Nome:	Pulsante Options->Video
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare il menu video tramite il pulsante "Options", per poi selezionare "Video".		
Prerequisiti:	TC-017funzionante.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni. 3. Premere il pulsante "Video".		
Risultati attesi:	Il menu Options->Video viene mostrato.		

Test Case:	TC-019	Nome:	Pulsante Options->Audio
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare il menu Audio tramite il pulsante "Options", per poi selezionare "Audio".		
Prerequisiti:	TC-017funzionante.		
Procedura:	1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni. 3. Premere il pulsante "Audio".		
Risultati attesi:	Il menu Options->Audio viene mostrato.		

Colorful Songs

Test Case:	TC-020	Nome:	Pulsante Options->Audio->Slider
Descrizione:	Tramite gli slider impostare l'audio dell'applicativo.		
Prerequisiti:	TC-019funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni. 3. Premere il pulsante "Audio". 4. Usare gli slider. 		
Risultati attesi:	L'audio viene impostato correttamente in base agli slider.		

Test Case:	TC-021	Nome:	Pulsante Options->Controls
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare il menu Controls tramite il pulsante "Options", per poi selezionare "Controls".		
Prerequisiti:	TC-017 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni. 3. Premere il pulsante "Controls". 		
Risultati attesi:	Il menu Options->Controls viene mostrato.		

Test Case:	TC-022	Nome:	Pulsante Options->Controls->Movement Preference
Descrizione:	Tramite il menu a tendina modificare le movement preference.		
Prerequisiti:	TC-021 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Options" e visualizzare il menu delle opzioni. 3. Premere il pulsante "Controls". 4. Utilizzare il menu a tendina per modificare le movement preference 		
Risultati attesi:	Le movement preference vengono impostate correttamente in base al menu a tendina.		

Test Case:	TC-023	Nome:	Pulsante Credits.
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare i crediti tramite il pulsante "Credits".		
Prerequisiti:	TC-015 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante "Credits" e visualizzare i crediti. 		
Risultati attesi:	I crediti vengono mostrati.		

Colorful Songs

Test Case:	TC-024	Nome:	Pulsante Options->Back.
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare il menu opzioni e utilizzare il pulsante back per tornare al menu principale.		
Prerequisiti:	TC-017 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante “Options”. 3. Dal menu opzioni premere il pulsante “Back”. 		
Risultati attesi:	Entrando nel menu opzioni è possibile premere il pulsante back per tornare al menu principale.		

Test Case:	TC-025	Nome:	Pulsante Credits->Back.
Descrizione:	Dal menu principale del gioco, sarà possibile visualizzare i crediti e utilizzare il pulsante back per tornare al menu principale.		
Prerequisiti:	TC-023 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante “Credits”. 3. Dai crediti premere il pulsante “Back”. 		
Risultati attesi:	Entrando nei crediti è possibile premere il pulsante back per tornare al menu principale.		

Test Case:	TC-026	Nome:	Pulsante Exit.
Descrizione:	Dal menu principale del gioco, sarà possibile chiudere dal gioco premendo il pulsante “Exit”.		
Prerequisiti:	TC-015 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Dal menu principale, premere il pulsante “Exit”. 		
Risultati attesi:	Premendo il pulsante exit si chiude il gioco.		

Test Case:	TC-027	Nome:	Menu in gioco.
Descrizione:	Quando si è in gioco, si può accedere al menu.		
Prerequisiti:	TC-016 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere “Start” 3. In gioco, premere il keybind per il menu. 		
Risultati attesi:	Quando si è in gioco, quando si preme il keybind specifico si apre il menu in gioco.		

Colorful Songs

Test Case:	TC-028	Nome:	Menu in gioco->Resume
Descrizione:	Quando si è in gioco, si può accedere al menu e ritornare a giocare.		
Prerequisiti:	TC-027 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start" 3. In gioco, premere il keybind per il menu. 		
Risultati attesi:	Quando si preme "Resume", il menu viene chiuso e si torna a giocare.		

Test Case:	TC-029	Nome:	Menu in gioco->Options
Descrizione:	Quando si è in gioco, si può accedere al menu e modificare le opzioni.		
Prerequisiti:	TC-027 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 		
Risultati attesi:	Quando si preme il pulsante "Options" si visualizzano le opzioni.		

Test Case:	TC-030	Nome:	Menu in gioco->Options->Video
Descrizione:	Quando si è in gioco, si può accedere al menu e visualizzare le opzioni video.		
Prerequisiti:	TC-029 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Video". 		
Risultati attesi:	Visualizzare le opzioni Video		

Test Case:	TC-031	Nome:	Menu in gioco->Options->Audio
Descrizione:	Quando si è in gioco, si può accedere al menu e visualizzare le opzioni audio.		
Prerequisiti:	TC-030 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Audio". 		
Risultati attesi:	Visualizzare le opzioni di Audio.		

Colorful Songs

Test Case:	TC-032	Nome:	Menu in gioco->Options->Audio->Slider
Descrizione:	Impostare l'audio tramite gli slider.		
Prerequisiti:	TC-032 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Audio". 6. Usare gli slider. 		
Risultati attesi:	L'audio viene impostato correttamente in base agli slider.		

Test Case:	TC-033	Nome:	Menu in gioco->Options->Controls
Descrizione:	Quando si è in gioco, si può accedere al menu e visualizzare le opzioni dei controlli.		
Prerequisiti:	TC-029 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Controls". 		
Risultati attesi:	Visualizzare le opzioni di Controls.		

Test Case:	TC-034	Nome:	Menu in gioco->Options->Controls-> Movement Preference
Descrizione:	Modificare il Movement Preferences con il menu a tendina		
Prerequisiti:	TC-033 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Controls". 6. Utilizzare il menu a tendina per modificare le movement preference. 		
Risultati attesi:	Le movement preference vengono impostate correttamente in base al menu a tendina.		

Colorful Songs

Test Case:	TC-035	Nome:	Menu in gioco->Options->Back
Descrizione:	Quando si è in gioco, si può accedere al menu opzioni e tornare indietro.		
Prerequisiti:	TC-029 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Options". 5. Selezionare il pulsante "Back". 		
Risultati attesi:	Dal menu opzioni si può tornare indietro al menu generale.		

Test Case:	TC-036	Nome:	Menu in gioco->Main Menu
Descrizione:	Quando si è in gioco, si può accedere al menu opzioni e tornare indietro.		
Prerequisiti:	TC-027 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Main Menu". 		
Risultati attesi:	Dal menu in gioco premendo "Main Menu" si torna al menu principale.		

Test Case:	TC-037	Nome:	Menu in gioco->Quit
Descrizione:	Quando si è in gioco, si può chiudere il gioco		
Prerequisiti:	TC-027 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il gioco. 2. Premere "Start". 3. In gioco, premere il keybind per il menu. 4. Selezionare il pulsante "Quit". 		
Risultati attesi:	Dal menu in gioco premendo "Quit" si chiude il gioco.		

Test Case:	TC-038	Nome:	Invio richiesta d'amicizia
Descrizione:	Dalla leaderboard mandare una richiesta di amicizia		
Prerequisiti:	TC-005 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Visualizzare la leaderboard 2. Cliccare su "Send Friend Request" 3. Controllare che sia arrivata all'utente 		
Risultati attesi:	All'utente a cui è stata mandata la richiesta arriva la notifica.		

Colorful Songs

Test Case:	TC-039	Nome:	Accettazione richiesta d'amicizia
Descrizione:	Dal proprio centro notifiche accettare una richiesta di amicizia		
Prerequisiti:	TC-038 funzionante.		
Procedura:	<ol style="list-style-type: none"> 1. Visualizzare la leaderboard 2. Andare nel centro notifiche 3. Accettare la richiesta di amicizia 		
Risultati attesi:	Entrambi i giocatori hanno l'altro nella proprio leaderboard degli amici se hanno già giocato.		

Test Case:	TC-040	Nome:	Creare un nuovo utente tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo POST creare un nuovo utente		
Prerequisiti:	API funzionante		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo POST 3. Scrivere l'url (http://localhost:8080/API/public/user) 4. Inserire un JSON: <pre> { "username":"testcase", "password":"testcase", "email":"test@case.cn" } </pre> 5. Mandare la richiesta 		
Risultati attesi:	L'utente è stato creato nel database		

Test Case:	TC-041	Nome:	Creare un nuovo punteggio tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo POST creare un nuovo punteggio		
Prerequisiti:	API funzionante		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo POST 3. Scrivere l'url (http://localhost:8080/API/public/user) 4. Inserire un JSON: <pre> { "score":"00:35:29", "user_id":"26" } </pre> 5. Mandare la richiesta 		
Risultati attesi:	Il punteggio è stato creato nel database		

Colorful Songs

Test Case:	TC-042	Nome:	Aggiornare il punteggio di un utente tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo PUT aggiornare il punteggio di un utente		
Prerequisiti:	API funzionante e punteggio già presente		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo PUT 3. Scrivere l'url (http://localhost:8080/API/public/user/26) 4. Inserire un JSON: <pre>{ "score": "00:15:29", "user_id": "26" }</pre> 5. Mandare la richiesta 		
Risultati attesi:	Il punteggio dell'utente è cambiato nel database		

Test Case:	TC-043	Nome:	Ricevere i dettagli di un utente utilizzando il suo id tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo GET visualizzare i dettagli di un utente specifico		
Prerequisiti:	API funzionante		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo GET 3. Scrivere l'url (http://localhost:8080/API/public/user/26) 4. Mandare la richiesta 		
Risultati attesi:	Tutti i dettagli dell'utente		

Test Case:	TC-044	Nome:	Ricevere i dettagli di un utente utilizzando il suo username tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo GET visualizzare le informazioni utili per il login di un utente		
Prerequisiti:	API funzionante		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo GET 3. Scrivere l'url (http://localhost:8080/API/public/user/testcase) 4. Mandare la richiesta 		
Risultati attesi:	Tutti i dettagli utili per fare il login dell'utente		

Colorful Songs

Test Case:	TC-045	Nome:	Eliminare un utente tramite API
Riferimento:	REQ-12		
Descrizione:	Tramite il metodo DELETE eliminare un utente		
Prerequisiti:	API funzionante		
Procedura:	<ol style="list-style-type: none"> 1. Aprire Postman 2. Selezionare il metodo DELETE 3. Scrivere l'url (http://localhost:8080/API/public/user/26) 4. Mandare la richiesta 		
Risultati attesi:	L'utente viene eliminato		

Test Case:	TC-046	Nome:	Effettuare la registrazione sbagliata
Riferimento:	REQ-003		
Descrizione:	Tentare di effettuare una registrazione all'interno del gioco stesso con un utente già esistente.		
Prerequisiti:	<ol style="list-style-type: none"> 1. Il database ColorfulSongs accessibile. 2. Il gioco deve comunicare con il servizio API. 		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il proprio gioco. 2. Sul menu principale cliccare il tasto di Register. 3. Eseguire la registrazione con le credenziali di un utente esistente. 4. Viene ritornato l'errore delle credenziali 		
Risultati attesi:	Viene ritornato all'utente l'errore che esiste già un utente con lo stesso username.		

5.2 Risultati test

Test Case	Risultato	Data
TC-001	Passato	21.05.2025
TC-002	Non passato	21.05.2025
TC-003	Non passato	21.05.2025
TC-004	Non passato	21.05.2025
TC-005	Passato	21.05.2025
TC-006	Passato	21.05.2025
TC-007	Passato	21.05.2025
TC-008	Passato	21.05.2025
TC-015	Passato	14.05.2025
TC-016	Passato	14.05.2025
TC-017	Passato	14.05.2025
TC-018	Passato	14.05.2025
TC-019	Passato	14.05.2025
TC-020	Passato	14.05.2025
TC-021	Passato	14.05.2025
TC-022	Passato	14.05.2025
TC-023	Passato	14.05.2025
TC-024	Passato	14.05.2025
TC-025	Passato	14.05.2025
TC-026	Passato	14.05.2025
TC-027	Passato	14.05.2025
TC-028	Passato	14.05.2025
TC-029	Passato	14.05.2025
TC-030	Passato	14.05.2025
TC-031	Passato	14.05.2025
TC-032	Passato	14.05.2025
TC-033	Passato	14.05.2025
TC-034	Passato	14.05.2025
TC-035	Passato	14.05.2025
TC-036	Passato	14.05.2025
TC-037	Passato	14.05.2025
TC-038	Passato	21.05.2025
TC-039	Passato	21.05.2025
TC-040	Passato	21.05.2025
TC-041	Passato	21.05.2025
TC-042	Passato	21.05.2025
TC-043	Passato	21.05.2025
TC-044	Passato	21.05.2025
TC-045	Passato	21.05.2025
TC-046	Non passato	21.05.2025

5.3 Mancanze/limitazioni conosciute

A progetto terminato, mancano sicuramente una grafica più dettagliata, eseguita meglio, questo perché ci siamo concentrati sul gameplay del gioco per renderlo il più completo possibile sotto l'aspetto di giocabilità, lasciando in disparte il comparto grafico. La grafica non era comunque un requisito di questo progetto, essendo che la parte importante era la programmazione, quindi il gameplay.

Altre mancanze sono che il Login e la Registrazione non sono state implementate nel gioco per mancanza di tempo.

6 Consuntivo

6.1 Primo Sprint

ColorfulSongs	118.58 h?	mer 29.01.25	mer 28.05.25	
Sprint 1	21 h	mer 29.01.25	mer 12.02.25	
Analisi dei requisiti	2 h	mer 29.01.25	mer 29.01.25	Full Team
Backlog	1 h	mer 29.01.25	mer 29.01.25	Full Team
Meeting di Kickoff	1 h	mer 29.01.25	mer 29.01.25	Full Team
Schema ER	1 h	mer 29.01.25	mer 29.01.25	Sebastiano De Bertoldi
Use case	1 h	mer 29.01.25	mer 29.01.25	Lorenzo Berther
Creazione Database ColorfulSongs	1 h	mer 05.02.25	mer 05.02.25	Sebastiano De Bertoldi
Kanban	1 h	mer 29.01.25	mer 29.01.25	Kamil Siddiqui
Gantt Preventivo	2 h	mer 29.01.25	mer 29.01.25	Simone Demarchi
Inizializzare il progetto Unity	1 h	mer 29.01.25	mer 29.01.25	Lorenzo Berther, Simone Demarchi
Implementare Controller Game	1 h	mer 29.01.25	mer 29.01.25	Lorenzo Berther, Simone Demarchi
Pianificare la struttura degli enigmi	1 h	mer 12.02.25	mer 12.02.25	Lorenzo Berther, Simone Demarchi, Kamil Siddiqui
Configurazione ambiente di sviluppo PHP	2 h	mer 12.02.25	mer 12.02.25	Sebastiano De Bertoldi
Activity Diagram	1 h	mer 12.02.25	mer 12.02.25	Kamil Siddiqui
Mock up Interfacce	2 h	mer 12.02.25	mer 12.02.25	Sebastiano De Bertoldi, Simone Demarchi
Movimento base del player	1 h	mer 12.02.25	mer 12.02.25	Lorenzo Berther, Simone Demarchi
Animazione del personaggio	1 h	mer 12.02.25	mer 12.02.25	Lorenzo Berther, Simone Demarchi
Creazione pagine per il login Web	1 h	mer 12.02.25	mer 12.02.25	Sebastiano De Bertoldi

Figura 32: Primo Sprint

Nel primo sprint abbiamo eseguito l'analisi dei requisiti e il meeting di kickoff, dando ufficialmente inizio al progetto. Abbiamo organizzato il Kanban, creato il Gantt preventivo e realizzato lo schema ER, l'Use Case Diagram e l'Activity Diagram.

Per Unity, abbiamo inizializzato il progetto, implementato il controller base del gioco e pianificato la struttura degli enigmi e il movimento base del player, compresa l'animazione del personaggio.

Sul fronte web, abbiamo configurato l'ambiente PHP e creato mockup per le interfacce, inclusa la pagina di login web.

6.2 Secondo Sprint

4 Sprint 2	21 h	mer 19.02.25	mer 05.03.25	
Creazione funzione per eliminare gli utenti	1 h	mer 19.02.25	mer 19.02.25	Sebastiano De Bertoldi, Kamil Siddiqui
Pagina admin web	1 h	mer 19.02.25	mer 19.02.25	Sebastiano De Bertoldi, Kamil Siddiqui
Menu GUI Gioco	2 h	mer 19.02.25	mer 19.02.25	Lorenzo Berther, Simone Demarchi
Menu Opzioni Gioco	2 h	mer 19.02.25	mer 19.02.25	Lorenzo Berther, Simone Demarchi
Generazione stanza dinamica	2 h	mer 26.02.25	mer 26.02.25	Kamil Siddiqui
Implementazione audio	2 h	mer 26.02.25	mer 26.02.25	Lorenzo Berther, Simone Demarchi
Generazione nemici statici	2 h	mer 26.02.25	mer 26.02.25	Lorenzo Berther
Demo tecnica	1 h	mer 26.02.25	mer 26.02.25	Full Team
Implementazione menu controller	2 h	mer 05.03.25	mer 05.03.25	Lorenzo Berther, Simone Demarchi
API PHP	2 h	mer 05.03.25	mer 05.03.25	Sebastiano De Bertoldi
Filtri leaderboard	1 h	mer 05.03.25	mer 05.03.25	Sebastiano De Bertoldi, Kamil Siddiqui
Feedback Web	1 h	mer 05.03.25	mer 05.03.25	Sebastiano De Bertoldi, Kamil Siddiqui
Modificato il database	2 h	mer 05.03.25	mer 05.03.25	Sebastiano De Bertoldi, Kamil Siddiqui

Figura 33: Secondo Sprint

Lo sviluppo si è concentrato sull'implementazione del menu principale di gioco e delle opzioni, oltre all'integrazione dell'audio (musica e effetti sonori).

Abbiamo creato funzioni per la gestione degli utenti, la generazione dinamica delle stanze, nemici statici e una demo tecnica.

Lato backend, abbiamo sviluppato API PHP, filtri per la leaderboard, e apportato modifiche al database e alla pagina admin.

6.3 Terzo Sprint

4 Sprint 3	14 h	mer 12.03.25	mer 19.03.25	
Creazione nemico, tamburo	2 h	mer 12.03.25	mer 12.03.25	Lorenzo Berther
Generazione degli ostacoli nelle stanze	3 h	mer 12.03.25	mer 12.03.25	Simone Demarchi, Kamil Siddiqui
Aggiornato il database	2 h	mer 12.03.25	mer 12.03.25	Sebastiano De Bertoldi
WEB, permettere all'utente di vedere i propri amici	2 h	mer 19.03.25	mer 19.03.25	Sebastiano De Bertoldi
Generare casualmente il nemico tamburo	3 h	mer 19.03.25	mer 19.03.25	Lorenzo Berther, Simone Demarchi, Kamil Siddiqui
WEB, sezione amici	2 h	mer 19.03.25	mer 19.03.25	Sebastiano De Bertoldi

Figura 34: Terzo Sprint

Questo sprint è stato dedicato all'arricchimento del gameplay con la creazione di nuovi nemici (tamburo) e ostacoli dinamici generati casualmente.

Abbiamo inoltre implementato funzionalità social web per la gestione degli amici e aggiornato il database a supporto di queste novità.

6.4 Quarto Sprint

▲ Sprint 4	21 h	mer 09.04.25	mer 23.04.25	
Creazione ostacolo, percorso invisibile	4 h	mer 09.04.25	mer 09.04.25	Kamil Siddiqui
Creazione ostacolo, pedane finte	4 h	mer 09.04.25	mer 09.04.25	Kamil Siddiqui
Creazione ostacolo, cannoni	4 h	mer 16.04.25	mer 16.04.25	Lorenzo Berther
WEB, creazione script per l'highscore	3 h	mer 23.04.25	mer 23.04.25	Sebastiano De Bertoldi
Sezione impostazioni nel gioco	3 h	mer 23.04.25	mer 23.04.25	Simone Demarchi

Figura 35: Quarto Sprint

Abbiamo sviluppato nuovi ostacoli (percorsi invisibili, pedane finte, cannoni) e, sul lato web, uno script per l'highscore e la sezione impostazioni nel gioco.

6.5 Quinto Sprint

▲ Sprint 5	14 h	mer 30.04.25	mer 07.05.25	
Creazione ostacolo, portale finto	3 h	mer 30.04.25	mer 30.04.25	Lorenzo Berther
Rimuovere il cursore quando si gioca da controller	3 h	mer 30.04.25	mer 30.04.25	Simone Demarchi
CRT Shader	3 h	mer 07.05.25	mer 07.05.25	Lorenzo Berther, Sebastiano De Bertoldi
Creazione ostacolo, parkour	3 h	mer 07.05.25	mer 07.05.25	Sebastiano De Bertoldi, Kamil Siddiqui
Script caduta del player	2 h	mer 07.05.25	mer 07.05.25	Simone Demarchi

Figura 36: Quinto Sprint

Questo sprint ha visto la creazione di ulteriori ostacoli (portale finto, parkour), la rimozione del cursore per l'uso con controller, l'implementazione di effetti grafici (CRT Shader) e uno script per la caduta del player.

6.6 Sesto Sprint

▲ Sprint 6	21 h?	mer 14.05.25	mer 28.05.25	
Miglioramento delle GUI	2 h	mer 14.05.25	mer 14.05.25	Lorenzo Berther, Simone Demarchi
Preferenza del movimento	2 h	mer 14.05.25	mer 14.05.25	Lorenzo Berther
Feedback sonoro danni in gioco	2 h	mer 14.05.25	mer 14.05.25	Simone Demarchi
Creazione nemico, trombetta	2 h	mer 21.05.25	mer 21.05.25	Lorenzo Berther
Frame di invincibilità	2 h	mer 21.05.25	mer 21.05.25	Lorenzo Berther
Generazione nemici casuale	2 h	mer 21.05.25	mer 21.05.25	Kamil Siddiqui
Gestione punteggio della partita in gioco	2 h	mer 21.05.25	mer 21.05.25	Kamil Siddiqui
Protocollo di test	7 h	mer 21.05.25	mer 28.05.25	Full Team
Documentazione	118.58 h	mer 29.01.25	mer 28.05.25	Full Team

Figura 37: Sesto Sprint

Infine, abbiamo migliorato le GUI e le preferenze di movimento, aggiunto feedback sonoro per i danni, introdotto nuovi nemici (trombetta) e meccanismi di invincibilità. Abbiamo completato il protocollo di test e la documentazione finale.

6.7 Confronto

Aspetto	Gantt Consuntivo	Gantt Preventivo
Durata complessiva	118.58 ore	113.5 ore
Numero di sprint	6 sprint, più suddivisi e dettagliati	5 sprint, più generici e concentrati
Granularità delle attività	Molto elevata, molte micro-attività da 1-3 ore	Attività più ampie e meno suddivise
Pianificazione Unity	Dettagliata: controller, movimenti, animazioni, nemici, ostacoli	Meno dettagliata, con attività di gioco raggruppate
Pianificazione Web/PHP	Include mockup, configurazioni, API, filtri leaderboard, funzioni utente	Attività simili ma meno granulari, alcune concentrate in un'unica giornata
Gestione documentazione	Inclusa ma sovrapposta agli sprint di sviluppo	Prevista a parte e costante lungo tutto il progetto
Test e rifiniture	Protocollo di test dettagliato nell'ultimo sprint	Test backend e game testing concentrati nell'ultimo sprint
Flessibilità con il tempo	Attività ben distribuite e bilanciate nel tempo	Alcuni sprint molto brevi o molto lunghi (es. sprint 4 molto corto)

7 Conclusioni

Il progetto è stato completato con successo, non avrà molto impatto essendo che al mondo ci sono migliaia di centinaia di videogiochi che vengono continuamente pubblicati. È stato comunque un successo abbastanza importante per il nostro team, essendo il nostro primo progetto ci sentiamo soddisfatti di quello che abbiamo creato. Abbiamo scoperto che lavorare in team è più difficile di quanto sembra, ma c'è l'abbiamo comunque fatta e abbiamo completato il progetto, non riteniamo che sia stata una perdita di tempo. I risultati ottenuti riteniamo che siano buoni, considerando che maggior parte dello sviluppo su Unity è stata una nuova scoperta per maggior parte del team, sono risultati che ci serviranno in progetti futuri compresi sia nell'ambito videoludico ma anche di team work e di programmazione.

7.1 Sviluppi futuri

In futuro è in programma di migliorare i controlli e di aggiungere la possibilità di cambiare i tasti specifici così da rendere l'esperienza di gioco più personalizzabile. Oltre a questo sarà possibile giocare in multiplayer online, vedendo a stile fantasma chi sta giocando il livello contro di te, questo comprende anche la rigiocabilità di un seed (seme che crea il dungeon, si dovrà implementare anche questo), da esso si potrà ricreare lo stesso livello da giocare e perfezionare per tempi record.

Sicuramente una grafica migliorata e più unica è in programma, l'ispirazione low-poly resterà ma con modelli più dettagliati e migliorati anche in ambito di animazioni. Restano sempre sulla grafica, sarà possibile modificare il proprio personaggio per renderlo più "personale", con cosmetici e modifica del personaggio base. (Volto, altezza, colore, capelli, ecc...) Restano sempre nei limiti imposti dal gioco per non permettere ai giocatori di utilizzare questa aggiunta come vantaggio competitivo (hitbox più piccolo con personaggio più basso per esempio).

È in programma anche un possibile porting sulla console Nintendo Switch e Xbox, così da rendere il gioco disponibile su più piattaforme, concesso che il Centro Professionale Tecnico di Trevano ci lasci il permesso per farlo, anche se questo richiederà molto lavoro e dedizioni.

7.2 Considerazioni personali

7.2.1 Lorenzo

Durante questo progetto ho compreso le basi di Unity e le sue potenzialità, permettono di avere una base solida per un videogioco in 3D. Anche se siamo partiti da 0, Unity offre dei template base completi, con collisioni e personaggio, partendo da 0 abbiamo dovuto programmare il movimento base e la rotazione del personaggio, mentre altre funzioni come la gravità ci pensa Unity a gestirle.

Ho appreso le basi e qualche funzione avanzata di Blender, essendomi occupato di ogni modello del gioco ho potuto capire quanto sia importante anche questo ruolo, non riguarda la programmazione ma come l'utente alla fine vedrà il gioco e secondo me ha molto impatto questo aspetto.

Quando ho proposto di fare un videogioco mi aspettavo qualcosa di più semplice a dire la verità, anche perché inizialmente era un videogioco 2D, mi è piaciuto imparare partire da 0 fino ad arrivare ad un videogioco effettivamente giocabile.

In conclusione, ho capito come lavorare in un team, anche se piccolo, è importante suddividersi i ruoli e lavorare per riuscire a restare nelle tempistiche e portare avanti il progetto senza problemi. Ognuno è essenziale perché senza la parte di un membro non si può continuare.

7.2.2 Sebastiano

Durante questo progetto sono riuscito a affinare meglio le mie capacità di PHP, infatti sono riuscito a implementare quasi tutte le conoscenze che ho appreso durante l'anno in ambito di PHP. In più ho dovuto creare una API per la prima volta. Questo ha rappresentato una sfida interessante, perché essendo la prima volta non sapevo come si facesse, e anche se ho seguito una guida per la creazione ho comunque dovuto riadattarla per far funzionare il tutto.

Purtroppo non sono riuscito a utilizzare anche Unity, quindi non sono riuscito ad apprendere nuove competenze.

In conclusione, come progetto è stato stimolante, ho capito come lavorare in team, ho capito che la suddividendo i ruoli si riesce a lavorare meglio, e anche con un po' più di tranquillità, rispetto al lavorare da soli.

7.2.3 Simone

Questo progetto mi è stato molto utile per affinare le mie capacità di Team-work, è stata una sfida interessante il fatto di non dover gestire solo il mio tempo ma fare in modo che gli orari di tutti si incastrassero correttamente. Mi è piaciuto molto che ci si dava una mano a vicenda nei punti in cui si era più deboli e la suddivisione del lavoro spesso alleggeriva il carico di lavoro rendendolo più piacevole, produttivo e efficiente. Anche se in alcuni casi è stato difficile lavorare sul lavoro di qualcun altro, dato dal fatto che ognuno ragionava con una logica differente.

In questo progetto ho affinato le mie competenze di Unity con tutte le sue potenzialità in quanto Game Engine, quali ad esempio la gestione della gravità e il rendering video che non sono da scrivere a mano come codice. Anche se le mie conoscenze di Unity erano abbastanza scarse è facile trovare documentazione e la community è attiva, per tanto è abbastanza facile apprendere le conoscenze base necessarie per un progetto come il nostro.

Ho appreso anche delle conoscenze di base sulla struttura di un videogioco, quali la pipeline del rendering, il game loop e simili.

Ho dovuto anche approfondire le mie conoscenze di PHP e l'MVC dato che Sebastiano a volte mi chiedeva una mano.

La parte del progetto che mi è piaciuta di più del progetto è stato il Ti-Nerd, anche se fuori orario. Mi è piaciuta molto perché potevo interagire direttamente con il pubblico che dava dei feedback in tempo reale che ci hanno aiutato molto per capire che linea di tiro adottare, ad esempio che tipo di movimento usare, la difficoltà degli ostacoli e la giocabilità.

Personalmente mi sono anche divertito a creare un videogioco partendo da zero e renderlo funzionante, anche se spesso ci sono stati dei problemi con il team per vari motivi, principalmente per la logica diversa tra ognuno di noi, sia nello scrivere il codice, che nella struttura e simili.

7.2.4 Kamil

Questo progetto mi è stato molto utile, essendo che mi ha permesso di rivedere e affinare la mia capacità di Team-Work e imparare una buona base di Unity.

È stata una divertente e interessante sfida, essendo che non ero da solo, e abbiamo dovuto imparare a suddividerci il lavoro trovando i punti forti di ognuno.

Non è stato per niente facile lavorare in un team su un engine che non avevamo mai visto a scuola, essendo che c'era chi era più informato e chi meno. Ed essendo anche che tutti pensiamo in modo diverso, quindi collaborare e aiutarci per problemi nel codice è stato difficile, ma mi ha aiutato ad aprire la mente e vedere i problemi da più prospettive.

In questo progetto ho imparato a usare Unity, fino ad ora non lo avevo mai usato, quindi ero più indietro in confronto ai miei compagni che lo avevano già usato. Questo mi ha portato a dovermi documentare molto, cercando online e leggendo documentazioni o forum.

La parte più bella del progetto erano le nostre milestone agli eventi pubblici, come il Tinerd. Quindi il dover avere un prodotto giocabile da mostrare a un pubblico, saperlo presentare a una persona non esperta o molto esperta è stato molto d'aiuto per il progetto.

Personalmente mi sono divertito, nonostante qualche litigio o disaccordo su metodi di soluzione, abbiamo lavorato bene come team.

8 Glossario

Termine	Descrizione
API	Application Programming Interface , è un insieme di definizioni e protocolli per la creazione e l'integrazione di applicazioni software
Box collider	Componente fisico in Unity che definisce un'area cubica invisibile per rilevare collisioni tra oggetti.
Character controller	Componente di Unity progettato per gestire il movimento dei personaggi, come camminare e saltare, senza usare fisica rigida.
Coroutine	Funzione speciale in Unity (scritta in C#) che permette di eseguire operazioni nel tempo, come ritardi o animazioni.
Dungeon	Ambiente di gioco (spesso sotterraneo) usato in giochi d'avventura o roguelike, pieno di nemici, ostacoli o ricompense.
Eloquent	È un ORM
GameObject	Oggetto base in Unity che rappresenta qualsiasi elemento della scena, come personaggi, luci o suoni.
Game Loop	Ciclo continuo che gestisce l'aggiornamento del gioco, il rendering e la logica, fondamentale in ogni motore di gioco.
MVC	Model View Controller: è un framework un modello di architettura software che organizza un'applicazione in tre parti interconnesse: Model, View, Controller
ORM	Object-Relational Mapping: È una tecnica di programmazione che integra la programmazione ad oggetti con i sistemi RDBMS • Si tratta di un'astrazione dei dati presenti in un DB e della sua implementazione I dati presenti nel database relazionale vengono rappresentati sotto forma di oggetti nel linguaggio scelto
Parkour	Meccanica di gioco ispirata ai movimenti acrobatici realistici, come correre sui muri, arrampicarsi e saltare tra ostacoli.
Pattern	Schema ricorrente nel design del gioco, usato per progettare comportamenti, livelli, o nemici (es. pattern di attacco).
Platform	Genere di videogioco in cui il personaggio salta tra piattaforme ed evita ostacoli (es. Super Mario).
Prefab	Oggetto preconfigurato in Unity che può essere riutilizzato più volte nella scena o nel progetto.
Rendering pipeline	Insieme di processi che convertono i dati della scena 3D in immagini visibili sullo schermo.
Roguelike	Genere di gioco con livelli generati casualmente, morte permanente e forte enfasi sull'esplorazione e la difficoltà.
SFX (Sound Effects)	Effetti sonori in un gioco, come esplosioni, passi o colpi, che arricchiscono l'esperienza audio.
Texture	Immagine applicata su un modello 3D per dargli dettagli visivi come colori, superfici e materiali.
Unity	Motore di sviluppo multiplatforma per videogiochi e applicazioni interattive 2D/3D.

9 Indice delle figure

Figura 1: Use Case	9
Figura 2: Gantt preventivo	10
Figura 3: Schema E-R	13
Figura 4: Schema logico	13
Figura 5: Design interfaccia di Login	14
Figura 6: Design interfaccia di admin	15
Figura 7: Design interfaccia leaderboard.....	15
Figura 8: Design di una stanza.....	16
Figura 9: Design del percorso con pedane finte.....	16
Figura 10: Design della pedana (Come apparirebbe all'utente dopo qualche tentativo)	16
Figura 11: Design dell'ostacolo "Percorso con pedana invisibile e soluzione sul muro".....	17
Figura 12: Design dell'ostacolo "Percorso con pedana invisibile e soluzione sul muro" visto dall'utente.....	17
Figura 13: Design portali finti (dopo errore)	17
Figura 14: Design portali finti.....	17
Figura 15: cannoni dispari	18
Figura 16: Cannoni pari	18
Figura 17: Parkour.....	18
Figura 18: Pattern Tamburo.....	19
Figura 19: Trombetta	19
Figura 20: Componenti Telecamera.....	48
Figura 21: Componenti Audio Manager.....	49
Figura 22: Audio Mixer.....	49
Figura 23: Struttura Player.....	52
Figura 24: Componenti Player	52
Figura 25: Global Volume	67
Figura 26: Global Volume (Tonemapping)	67
Figura 27: Global Volume (Bloom)	67
Figura 28: Global Volume (Panini Projection)	68
Figura 29: Global Volume (Vignette)	68
Figura 30: Global Volume (Chromatic Abberation)	68
Figura 31: Global Volume (Film Grain).....	68
Figura 32: Primo Sprint.....	93
Figura 33: Secondo Sprint.....	94
Figura 34: Terzo Sprint	94
Figura 35: Quarto Sprint	95
Figura 36: Quinto Sprint.....	95
Figura 37: Sesto Sprint.....	95

10 Bibliografia

10.1 Sitografia

- [Unity CharacterController.collisionFlags](#), 05-02-2025
- <https://www.youtube.com/watch?v=-FhvQDqmgmU>, *Animate Characters in Unity 3D*, 12-02-2025
- <https://www.youtube.com/watch?v=gMd0xDEFE20>, *Import a Custom Font into Unity*, 26-02-2025
- <https://www.youtube.com/watch?v=N8whM1GiH4w>, *MUSIC SOUND EFFECTS Unity*, 26-02-2025
- <https://www.youtube.com/watch?v=49KVkXXpFB4>, *Unity CRT TV VFX*, 12-03-2025
- [developer.okta](#), *Simple REST API in PHP source code*, 12-03-2025
- [Gitlab Simple Rest API](#), *Simple REST API in PHP*, 26-03-2025
- <https://dev-bay.com/php-simple-rest-api/>, *PHP – simple REST API*, 26-03-2025
- <https://www.youtube.com/watch?v=cU4UaqrPb84>, *Unity3D API Request*, 02-04-2025
- [Unity Scripting](#), *Reference Scripting API*, 02-04-2025
- <https://docs.unity.com/>, *Documentazione Unity*, dal 05-02-2025 al 21-05-2025
- <https://chatgpt.com/>, *Spiegazione errori*, dal 05-02-2025 al 21-05-2025

11 Allegati

- Diario di lavoro
- Codice sorgente
- Prodotto
- Gantt preventivo
- Gantt consuntivo
- QdC
- Activity Diagram
- Database
- Use case
- Abstarct