

SUPSI

Code Quest - Multiplayer

Studente/i

Matteo Cadoni

Relatore

Masiar Babazadeh

Correlatore

-

Committente

Masiar Babazadeh

Corso di laurea

Ingegneria informatica

Modulo

O1 Architetture software di rete

Anno

2022-2023

Data

1 settembre 2023

STUDENTSUPSI

Indice

Abstract	1
Progetto Assegnato	3
Introduzione	5
1 Stato dell'arte	7
1.1 Gamification	7
1.2 Competitività e cooperazione effetti sulla motivazione	10
1.3 Esempi di applicazioni in commercio	11
2 Il contesto di Code Quest	15
2.1 Nuove modalità di gioco e meccaniche	16
2.1.1 Modalità competitive	16
2.1.2 Modalità cooperative	17
2.2 Conclusioni	17
3 Architettura realizzata	19
3.1 Database	21
3.1.1 User	22
3.1.2 Match	22
3.1.3 Achievement	23
3.1.4 Statistics	23
3.2 Servizio API REST	24
3.3 Frontend web	27
3.4 Autenticazione e sicurezza	33
3.4.1 OAuth 2.0 e OpenID Connect	33
3.5 Keycloak	37
3.5.1 Single-Sign On	37
3.5.2 Identity Brokering e Social Login	38
3.5.3 Admin console	38
3.5.4 Account console	38

3.5.5	Servizi di autorizzazione e API	39
3.5.6	Conclusioni	39
3.6	Configurazione del servizio web e di Keycloak	40
3.6.1	Docker-compose	40
3.6.2	Configurazione di Keycloak	42
3.7	Integrazione di OpenID nei vari servizi	43
3.7.1	Resource Server e Spring	43
3.7.2	OpenID Connect e React	44
3.7.3	OpenID Connect e Unity	46
3.8	Conclusioni	48
4	SmartGrid e algoritmo A*	49
4.1	SmartGrid	50
4.2	Pathfinder e algoritmo A*	52
4.2.1	A* Search	52
4.2.2	A* e SmartGrid	56
5	Gestione del multiplayer online	57
5.1	Architetture per lo sviluppo di videogiochi multiplayer	58
5.2	Photon	59
5.3	Gestione connessione alle partite	59
5.4	Gestione della sincronizzazione	61
5.5	Interazione con il servizio web	65
5.6	Configurazione del client su servizi web e server di gioco differenti	66
5.7	Conclusioni	69
6	Matchmaking	71
7	Sistema di spawn	73
8	Gestione delle partite competitive	77
8.1	Regole del gioco	78
8.2	Elementi di gioco	80
8.3	Assegnamento squadre e avvio di una partita	82
8.4	Conclusioni	87
9	Generazione procedurale dei livelli	89
9.1	PCG (Procedural Content Generation)	89
9.2	BSP (Binary Space Partitioning)	90
9.3	Perlin Noise	95
9.4	Generazione delle mappe e Conclusioni	98

10 Gestione delle partite cooperative	101
10.1 Regole del gioco	101
10.2 Elementi di gioco	103
10.3 Chat e comunicazione	105
10.4 Conclusioni	108
11 Risultati e conclusioni	109
11.1 Risultati	109
11.2 Sviluppi futuri	110
11.3 Conclusioni	111

Elenco delle figure

1.1	Interfaccia di Code Combat	12
1.2	Interfaccia di Codeingame	13
1.3	Esempio di colonia su Screeps	13
1.4	Editor di Screeps	14
3.1	Architettura servizio web	19
3.2	Schema ER Database	21
3.3	Schema wireframe della pagine del profilo utente	28
3.4	Home dell'applicazione	28
3.5	Classifica	29
3.6	Profilo giocatore	29
3.7	Form di registrazione	30
3.8	Schermata di login	31
3.9	Schermata di modifica del profilo	32
3.10	Schermata di gestione degli utenti	32
3.11	Diagramma di funzionamento di OAuth 2.0	34
3.12	Diagramma di funzionamento di OAuth 2.0 con il token di refresh	35
3.13	Diagramma di funzionamento di OAuth 2.0 con il token di refresh	36
3.14	Schema di autenticazione client Unity	48
4.1	SmartGrid che si adatta all'ambiente circostante	51
4.2	Esempio di percorso calcolato dall'algoritmo su una griglia con degli ostacoli	52
4.3	Rappresentazione della distanza di Manhattan tra due punti	53
4.4	Esempio di percorso: In blu viene segnato il tracciato calcolato dall'alieno per raggiungere la torre	56
5.1	Tipi di architettura: (a) client server, (b) multi-server, (c) peer to peer	58
5.2	Diagramma delle classi nel NetworkManager	61
5.3	Diagramma delle classi e funzionamento HttpManager	66
5.4	Schermata del tool	67
5.5	Schermata per la gestione del server di autenticazione	68

5.6 Schermata per la gestione del server di gioco	68
8.1 Interfaccia del gioco: la prima icona rappresenta il bottone "CQ" per richiedere il time-out di programmazione	78
8.2 Rappresentazione di un elemento consumabile nell'interfaccia di gioco	81
9.1 Esempio di alberi generati proceduralmente	89
9.2 Spazio partizionato con il relativo BSPT(BSP Tree)	91
9.3 Esempio di Perlin Noise	95
9.4 Terreno generato con il Perlin Noise	96
9.5 Esempio di livello generato con il plugin	96
10.1 Schermata modalità cooperativa	101
10.2 Rappresentazione del punto di partenza (fumo rosso) e del punto di arrivo (fumo verde)	102
10.3 Chiave posizionata in una mappa di gioco	103
10.4 Rappresentazione di una fotocellula (luce viola) porta (luce verde)	104
10.5 Pannello della chat	107
10.6 Rappresentazione del segnale di notifica	108

Elenco delle tabelle

3.1 Endpoint dell'API	26
3.2 Descrizione dei servizi	41

Abstract

Code Quest è un progetto con l'obiettivo di insegnare i principi fondamentali della programmazione agli studenti delle scuole medie e superiori.

Nella presente tesi è stata sviluppata e introdotta nel progetto esistente la modalità multiplayer che arricchisce l'esperienza di gioco, presentando due modalità: una competitiva e una cooperativa.

Le modalità competitive prevede una competizione uno contro uno all'interno di un'arena. Mentre la modalità cooperativa prevede il raggiungimento di un obiettivo comune.

Rispetto al progetto di partenza sono state apportate diverse modifiche al sistema di movimento, tramite l'introduzione di una griglia che consente l'aggiornamento dinamico. Ciò ha permesso anche l'implementazione di strumenti di generazione procedurale per lo sviluppo dei livelli.

Lo sviluppo del gioco è stato svolto utilizzando Unity. A supporto è stato implementato un servizio web composto da: un database, un server API REST e un'applicazione web, realizzati rispettivamente in Spring e ReactJS.

La sicurezza e l'autenticazione sono stati gestiti tramite Keycloak che ha consentito di usare il protocollo OpenID Connect.

Il risultato ottenuto è un gioco che fornisce una funzionalità multigiocatore che comprende le due modalità operative citate e lo rende pronto alla fase di test nelle scuole.

Parole chiave: *Competitività, Cooperazione, Gamification, Motivazione, Apprendimento, Sicurezza, Web*

English Version

Code Quest is a project with the goal of teaching the fundamentals of programming to middle and high school students.

In this thesis, a multiplayer mode was developed and introduced into the existing project that enriches the game experience by presenting two modes: a competitive mode and a cooperative mode.

Competitive modes involve one-on-one competition within an arena. While the cooperative mode involves achieving a common goal.

Compared to the initial design, several changes have been made to the movement system through the introduction of a grid that allows dynamic upgrading. This also allowed the implementation of procedural generation tools for level development.

Game development was carried out using Unity. To support it, a web service was implemented consisting of: a database, a REST API server, and a web application, built in Spring and ReactJS, respectively.

Security and authentication were handled using Keycloak, which allowed the OpenID Connect protocol to be used.

The result obtained is a game that provides multiplayer functionality that includes the two modes of operation mentioned above and makes it ready for the testing phase in schools.

Keywords: *Competitiveness, Cooperation, Gamification, Motivation, Learning, Security, Web*

Progetto assegnato

Descrizione

Code Quest è un gioco mirato all'apprendimento dell'informatica. Il gioco conta tre livelli di base dove il giocatore deve programmare un robot per farlo muovere in un'area con ostacoli e nemici. La programmazione del robot rispecchia i concetti base della programmazione e stimola il giocatore a migliorare le proprie abilità per sviluppare del codice sempre più raffinato ed efficace per superare i livelli.

In questo progetto di diploma si mira ad aggiungere delle funzionalità multiplayer dove due ,o più, giocatori possono programmare i propri robot per combattere tra di loro, oppure cooperare per superare dei livelli.

Compiti

Sviluppare l'interfaccia multiplayer dove i giocatori possono:

- Definire le regole del multiplayer tenendo in considerazione gli aspetti didattici del gioco
- Connettersi alla stessa instance di gioco
- Giocare uno contro l'altro
- Giocare in cooperativa
- Implementare un sistema a torneo

Obiettivi

Definire innovative regole di gioco per un'esperienza multiplayer su Code Quest (game design skills) Sviluppare le competenze di base legate allo sviluppo del multiplayer su Unity usando Photon (programming skills) Implementare il mutiplayer e testarne la robustezza (programming + testing skills)

Tecnologie

Per questo progetto sono state utilizzate le seguenti tecnologie:

- Unity
- Photon
- Spring
- React
- Docker
- OIDC
- Keycloak
- JavaFx

Introduzione

Code Quest è un progetto che pone l'obiettivo di realizzare un videogioco che insegni la programmazione agli studenti delle scuole medie e superiori. La gamification è uno strumento che nella didattica si sta diffondendo sempre di più per via dei benefici che si sono ottenuti negli anni.

L'insegnamento della programmazione è un processo complesso e molto spesso mantenere un buon livello di motivazione non è sempre semplice, sia per gli studenti che per gli insegnanti. Grazie alla gamification si cerca di insegnare tramite attività ludiche, sfruttando diversi elementi di game design presenti nei giochi che hanno il fine di intrattenimento.

In questo documento verrà analizzato lo sviluppo delle modalità multiplayer e degli aggiornamenti introdotti. Il cuore di Code Quest è il linguaggio CQ, un linguaggio di programmazione realizzato ad hoc per il progetto che semplifica certi tipi di istruzioni, mantenendo molti elementi simili ai linguaggi moderni.

Il gioco è stato sviluppato con Unity, uno dei software di sviluppo di videogiochi più usati al mondo. Per quanto riguarda la parte multiplayer si è utilizzato il framework Photon una popolare tecnologia per lo sviluppo di videogiochi multiplayer e non solo. Inoltre, Photon si occupa anche dell'hosting delle partite tramite il sistema cloud messo a disposizione, compreso di piano gratuito.

Nella parte iniziale verrà analizzata la letteratura relativa alla competitività e alla cooperazione nell'ambito della gamification, questo studio ha consentito di prendere le decisioni su quali meccaniche introdurre all'interno di Code Quest. Successivamente verrà esaminato il servizio web realizzato per la registrazione dei risultati e per gestire gli account dei giocatori.

Il servizio web si compone di: un database MySQL, un servizio REST realizzato in Java tramite il framework Spring e di un frontend web realizzato in React, uno dei framework javascript più usati. Per quanto riguarda la parte di sicurezza è stato usato Keycloak, un server open-source che consente di avere un sistema di autenticazione SSO centralizzato. Questa scelta ha consentito l'uso del

protocollo Open ID Connect per connettere i vari punti di accesso al sistema, mantendo il tutto sicuro.

Dopo il servizio web verrà presentato il nuovo sistema di movimento basato su una griglia che è in grado di adattarsi all'ambiente circostante. Verrà anche presentato l'algoritmo A* utilizzato per calcolare il percorso sulla griglia per raggiungere determinati punti evitando gli ostacoli. Questo algoritmo è stato usato per fare muovere in autonomia alcuni elementi.

Nei capitoli successivi verrà descritto come è stato gestito il multiplayer tramite Photon, con la gestione delle partite, il matchmaking, la gestione delle partite competitive e cooperative. Inoltre, verrà presentato il sistema di generazione procedurale, realizzato per velocizzare lo sviluppo dei livelli cooperativi e che getta le basi per un eventuale sviluppo di un sistema automatico di generazione delle partite di questa tipologia.

Nella parte finale verranno presentati i risultati, eventuali sviluppi futuri e le conclusioni.

Capitolo 1

Stato dell'arte

In questo capitolo, sarà presentato il concetto di gamification attraverso un'analisi approfondita della letteratura esistente e dei prodotti attualmente disponibili sul mercato. Si esamineranno in particolare gli aspetti chiave da considerare quando si applica la gamification in un contesto multigiocatore.

1.1 Gamification

Oggi giorno il mondo dell'insegnamento è sempre alla ricerca di nuovi strumenti per rendere la didattica sempre più stimolante e coinvolgente. La motivazione in fase di apprendimento è un elemento fondamentale e ha un grosso impatto sul processo di apprendimento e sui risultati [1].

Tra le varie strategie per aumentare la motivazione, una che ha guadagnato notevole attenzione negli ultimi anni è la "gamification". Il termine nasce all'inizio degli anni 2000, ma ha acquisito un considerevole incremento d'interesse dal 2010. In questo capitolo, verrà presentata la definizione di gamification, le sue componenti chiave e come può influenzare sia la motivazione estrinseca che intrinseca.

Secondo l'articolo "*How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction*" [2], l'idea alla base della gamification è quella di applicare gli elementi motivazionali dei videogiochi, attraverso specifici elementi di design, in contesti non ludici.

Tuttavia, nonostante il numero crescente di applicazioni applicazioni che usano la gamification, non esiste ancora una definizione scientifica universalmente accettata del termine. La definizione più attuale e diffusa è quella descritta da Gustafsson [3] il quale la definisce come: "l'uso di elementi di game design all'interno di contesti non di gioco".

Questa definizione non impone un obiettivo specifico, per evitare di limitare le possibili applicazioni, ma si concentra su 4 componenti principali [2]:

- **Game** definito come un sistema in cui i giocatori si impegnano in un conflitto artificiale che ha delle regole da seguire in cui il risultato è quantificabile. Questa caratteristica la si ottiene grazie all'utilizzo di meccaniche come punteggi, livelli e competizioni.
- **Elements** consentono di distinguere la gamification da videogiochi seri, ossia giochi che sono sviluppati con uno scopo non di intrattenimento. Elementi che vengono utilizzati sono ad esempio i badge, le classifiche e i punteggi.
- **Design** contrappone “elementi di design del gioco” e “tecniche basate sul gioco” nella gamification. Sottolinea che la gamification consiste nell'applicare intenzionalmente elementi di design del gioco (come livelli, punteggi, sfide) in contesti non di gioco, non nell'utilizzare tecniche di gioco (come motori di gioco o controller) in contesti estranei.
- **Non-game contexts** non specifica le possibili aree di applicazione della gamification, lasciando libertà di utilizzo in molti scenari. La definizione esclude esplicitamente l'uso di elementi di game design all'interno dei giochi stessi o durante il processo di game design.

Inizialmente si ipotizzava che la gamification influenzasse principalmente la motivazione estrinseca.

La **motivazione estrinseca** si riferisce all'esecuzione di un comportamento che dipende fondamentalmente dal raggiungimento di un risultato che è separato dall'azione stessa. In altre parole, la motivazione estrinseca è strumentale per natura. Viene eseguita al fine di ottenere un altro risultato [4]. Ad esempio, un utente potrebbe essere motivato a completare più lezioni o quiz in una app di apprendimento delle lingue per guadagnare punti e salire nella classifica degli utenti.

Elementi come badge, classifiche e punteggi possono essere utilizzati come ricompense e di conseguenza condizionare il giocatore a comportarsi in un determinato modo per ricevere il premio [2].

Tuttavia, la motivazione estrinseca può essere internalizzata all'interno della **motivazione intrinseca** [2]. La motivazione intrinseca si riferisce al desiderio di svolgere un'attività perché è in sé gratificante o piacevole, non perché porta a un risultato esterno o a una ricompensa. In altre parole, l'azione e il risultato sono la stessa cosa. Un esempio è un bambino che gioca all'aperto, correndo e

saltando, semplicemente perché trova queste attività divertenti e soddisfacenti, non perché sta cercando di ottenere una ricompensa esterna [4].

Questo processo però avviene quando i giocatori sono più competenti e auto-determinati. Attraverso questo meccanismo, ci si aspetta che la gamification possa favorire la motivazione intrinseca attraverso le caratteristiche specifiche degli elementi di gioco e il loro feedback diretto [2].

In conclusione, la gamification è uno strumento potente che può essere utilizzato per aumentare sia la motivazione estrinseca che intrinseca. Sebbene la gamification sia stata inizialmente vista come un modo per aumentare la motivazione estrinseca attraverso l'uso di ricompense esterne come badge, punti e classifiche, la ricerca ha dimostrato che può anche favorire la motivazione intrinseca. Questo avviene quando i giocatori si sentono competenti e autodeterminati, e quando gli elementi di gioco e il feedback che forniscono sono in linea con i loro interessi e obiettivi personali.

Tuttavia, è importante notare che la gamification non è una soluzione universale per tutti i problemi di motivazione. Il suo successo dipende da una serie di fattori, tra cui il design del gioco, il contesto in cui viene utilizzato, e le caratteristiche individuali dei giocatori.

Nei prossimi capitoli, verrà approfondito ulteriormente la relazione tra gamification e motivazione, esaminando come specifici elementi di design dei giochi possono influenzare la motivazione in diversi contesti. Inoltre, verrà analizzate alcune applicazioni esistenti per comprendere meglio come la gamification viene applicata nella pratica e come le varie meccaniche sono state introdotte all'interno di Code Quest.

1.2 Competitività e cooperazione effetti sulla motivazione

L'apprendimento della programmazione è una sfida che richiede impegno, dedizione e una grande dose di pratica. Tuttavia, mantenere alti livelli di motivazione può essere difficile, soprattutto quando gli studenti si confrontano con concetti complessi. Questo capitolo analizza l'intersezione tra gamificazione, competizione e cooperazione nell'apprendimento della programmazione.

L'uso di applicazioni multigiocatore comporta la distinzione di due macro insiemi: Il primo quello delle **applicazioni competitive** e il secondo quello delle **applicazioni cooperative**.

Nel contesto delle applicazioni che promuovono la competizione, l'implementazione di sistemi di punteggio e classifiche può essere un metodo efficace. Questa competizione stimolata tra gli utenti può rafforzare la loro motivazione intrinseca, alimentando un desiderio di miglioramento e successo [1].

Nel caso delle applicazioni che favoriscono la cooperazione, l'ambiente si concentra maggiormente sulla coordinazione tra i vari partecipanti, con l'obiettivo di raggiungere un traguardo comune. Questo può includere il superamento di un ostacolo specifico o il completamento di un livello con un determinato risultato. Queste applicazioni mettono in risalto l'importanza della coordinazione e della condivisione di processi costruttivi, dove la discussione dei problemi e la ricerca di soluzioni avvengono in maniera collettiva [1].

Dai risultati dello studio del 2021 di Bovermann [1] è emerso che le applicazioni con un alto livello di competizione ha stimolato la motivazione intrinseca nella maggior parte dei candidati. Con un alto livello di competizione si intende un'attività che utilizza classifiche e punti, ma che alla fine si ha un gruppo/utente vincitore. Un altro aspetto rilevante emerso dalla ricerca riguarda l'approccio cooperativo. È stato osservato che gli studenti mostrano una maggiore motivazione quando i punti vengono raccolti a livello di gruppo o profilo individuale, con la prospettiva di emergere come il gruppo vincente e di visualizzare i propri successi su una classifica, o quando ricevono badge per il proprio profilo.

Questo sembra essere più stimolante rispetto all'idea di lavorare verso un obiettivo collettivo. Questi risultati suggeriscono che, nel contesto dell'apprendimento gamificato, la competizione, sia a livello di gruppo che individuale, può essere un potente catalizzatore per la motivazione degli studenti [1].

Entrando nello specifico dei coding games, lo studio condotto da Fisher nel 2021 [5] ha rilevato che mettersi in competizione con altri utenti fa aumentare "l'autovalutazione" di ciò che si sta facendo, ma fa diminuire l'aspetto del divertimento, e quindi rischiando di demotivare l'utente. Lo studio conclude che l'uso della

competizione in questo contesto non ha sempre vantaggi, ma può avere anche degli svantaggi.

In conclusione, la competizione e la cooperazione svolgono ruoli significativi nell'apprendimento della programmazione. La gamification, attraverso l'uso di elementi di design dei videogiochi, può migliorare l'esperienza di apprendimento e aumentare la motivazione degli studenti. La competizione e la cooperazione, possono stimolare la motivazione intrinseca e promuovere l'apprendimento collaborativo. Tuttavia, la ricerca sulla gamification e sugli effetti della competizione e della cooperazione è ancora in corso e non fornisce ancora conclusioni definitive su alcuni aspetti. Nonostante ciò, ci sono elementi comuni emersi da vari studi che forniscono indicazioni interessanti. Questi elementi sono stati presi in considerazione durante la progettazione del sistema che verrà dettagliatamente presentato nei capitoli successivi.

1.3 Esempi di applicazioni in commercio

Attualmente, esistono diversi titoli sul mercato che combinano la gamification con la programmazione in ambienti multiplayer. La maggior parte di questi giochi si basa su meccaniche di tipo puzzle, che richiedono ai giocatori di risolvere enigmi o problemi attraverso il coding. In questa sezione, verranno presentati alcuni di questi titoli, mettendo in luce le loro caratteristiche distintive. Come fonte di ispirazione per lo sviluppo di Code Quest sono stati presi come riferimento tre titoli specifici: Code Combat, Codingame e Screeps.

Code Combat è una piattaforma di apprendimento interattiva online che trasforma l'apprendimento della programmazione in un gioco avvincente. Gli studenti imparano a programmare utilizzando linguaggi come Python, JavaScript e altri linguaggi, risolvendo sfide in un gioco di ruolo basato su turni. Oltre all'aspetto educativo, Code Combat ha anche una componente competitiva. La piattaforma ospita campionati di e-sports, dove i partecipanti si sfidano programmando le mosse dei loro personaggi per raggiungere gli obiettivi del gioco. Questi obiettivi variano a seconda dell'arena in cui si svolge la competizione, rendendo ogni sfida diversa dalle altre. In sostanza, Code Combat non solo insegna ai giovani a programmare, ma lo fa in un modo che è sia coinvolgente che competitivo, offrendo un'esperienza di apprendimento unica [6].



Figura 1.1: Interfaccia di Code Combat

Codingame è un'altra piattaforma di apprendimento online simile a Code Combat, ma offre una gamma più ampia di linguaggi di programmazione tra cui scegliere. Tuttavia, la modalità di interazione con le istanze di gioco è meno sofisticata. Infatti, Codingame utilizza esclusivamente i canali di standard input e standard output per la comunicazione, il che rende le operazioni di lettura e scrittura molto generali, ma limita la possibilità di interazioni dirette con la scena di gioco, a meno che non si utilizzino specifici valori o schemi di stringhe. In Codingame, ai giocatori vengono assegnati problemi da risolvere e viene stilata una classifica in base al tempo impiegato per completare i problemi. Inoltre, la piattaforma promuove un ambiente di apprendimento collaborativo: i giocatori che completano i problemi più velocemente possono scegliere di aiutare i loro compagni pubblicando il codice che hanno utilizzato. Questo consente di stimolare l'apprendimento peer-to-peer e la collaborazione. Inoltre, il gioco fornisce diversi badge che si ottengono completando le missioni proposte [7].

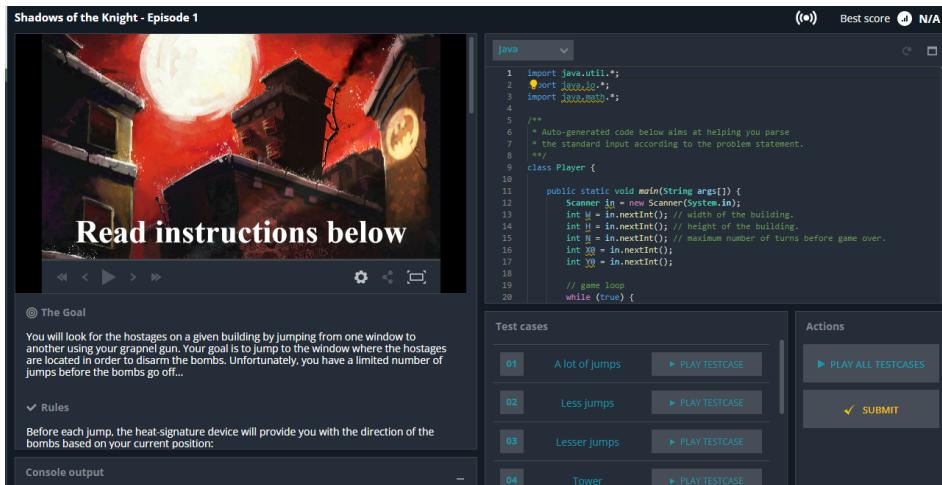


Figura 1.2: Interfaccia di Codeingame

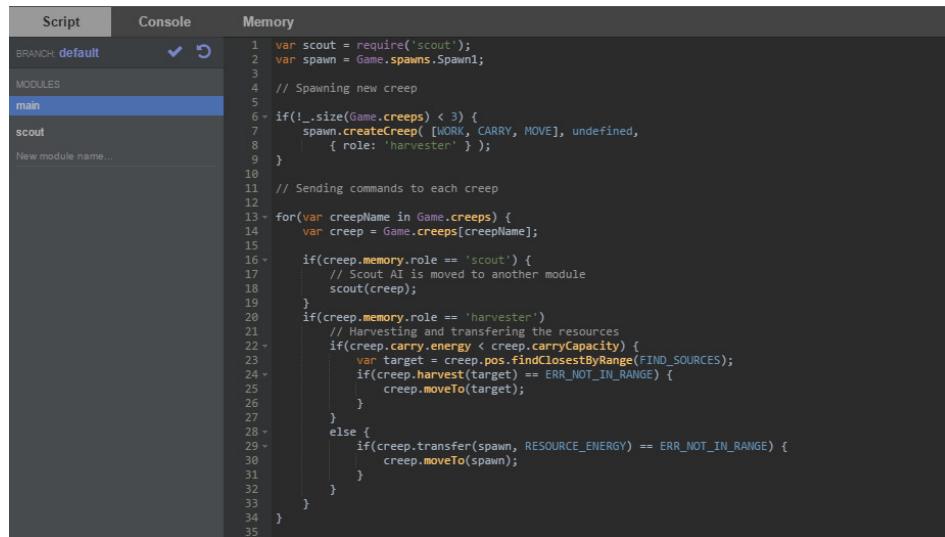
Screeps è un gioco di strategia in tempo reale massivo multiplayer online (MMO RTS) in cui ogni giocatore può creare e gestire la propria colonia all'interno di un unico mondo persistente condiviso da tutti i giocatori. Le colonie possono estrarre risorse, costruire unità e conquistare territori. Man mano che si conquista territorio, si farà crescere la propria influenza sul mondo di gioco, così come le proprie capacità di espansione. Tuttavia, la competizione è feroce, poiché più giocatori possono puntare allo stesso territorio.



Figura 1.3: Esempio di colonia su Screeps

La caratteristica distintiva di Screeps è che tutte le azioni nel gioco sono controllate tramite script JavaScript. Questo significa che le operazioni continuano ad essere eseguite anche quando il giocatore è offline, a patto che siano state

programmate correttamente. In sostanza, l'obiettivo di Screeps è scrivere un'intelligenza artificiale (IA) che sia in grado di costruire e mantenere una colonia in modo efficiente. Il codice del gioco può essere scritto utilizzando l'editor integrato nel gioco o un qualsiasi ambiente di sviluppo (IDE) esistente. Inoltre, oltre a JavaScript, Screeps supporta l'uso di altri linguaggi di programmazione tramite WebAssembly, ampliando così la sua accessibilità a una vasta gamma di sviluppatori.



```

 1 var scout = require('scout');
2 var spawn = Game.spawns.Spawn1;
3
4 // Spawning new creep
5
6 if(!_.size(Game.creeps) < 3) {
7     spawn.createCreep([WORK, CARRY, MOVE], undefined,
8         { role: 'harvester' } );
9 }
10
11 // Sending commands to each creep
12
13 for(var creepName in Game.creeps) {
14     var creep = Game.creeps[creepName];
15
16     if(creep.memory.role == 'scout') {
17         // Scout AI is moved to another module
18         scout(creep);
19     }
20     if(creep.memory.role == 'harvester') {
21         // Harvesting and transferring the resources
22         if(creep.carry.energy < creep.carryCapacity) {
23             var target = creep.pos.findClosestByRange(FIND_SOURCES);
24             if(creep.harvest(target) == ERR_NOT_IN_RANGE) {
25                 creep.moveTo(target);
26             }
27         }
28         else {
29             if(creep.transfer(spawn, RESOURCE_ENERGY) == ERR_NOT_IN_RANGE) {
30                 creep.moveTo(spawn);
31             }
32         }
33     }
34 }
35

```

Figura 1.4: Editor di Screeps

In sintesi, Screeps è un gioco unico che combina elementi di strategia in tempo reale con la programmazione, offrendo un'esperienza di gioco profonda e coinvolgente per gli appassionati di entrambi i campi [8].

In conclusione, il mercato offre una serie di strumenti sofisticati e completi nel campo della gamification. Sebbene i puzzle game siano spesso la modalità di gioco prevalente, si è potuto constatare come la programmazione possa essere integrata anche in giochi di strategia, come dimostra l'esempio di Screeps. Questo amplia notevolmente le possibilità di apprendimento e coinvolgimento, offrendo un'esperienza di gioco che va oltre il semplice intrattenimento per incorporare elementi di sviluppo delle competenze tecniche.

Capitolo 2

Il contesto di Code Quest

Code Quest è un progetto con l'obiettivo di insegnare i principi fondamentali della programmazione agli studenti delle scuole medie e superiori. A differenza di altri prodotti menzionati nei capitoli precedenti, Code Quest utilizza un linguaggio di programmazione appositamente progettato, chiamato CQ.

Questo linguaggio è stato sviluppato con l'obiettivo di semplificare l'apprendimento dei costrutti di base della programmazione, eliminando elementi potenzialmente complicati come i tipi di dati e la punteggiatura specifica del codice.

L'approccio di Code Quest è quello di rendere la programmazione accessibile e intuitiva, permettendo agli studenti di concentrarsi sui concetti chiave senza essere sopraffatti da dettagli tecnici complessi. Questo rende Code Quest un ottimo strumento per introdurre gli studenti alla programmazione.

Attualmente, Code Quest offre solo una modalità single-player, ma include già diversi elementi che faciliteranno l'integrazione di una modalità multiplayer.

2.1 Nuove modalità di gioco e meccaniche

L'evoluzione del progetto Code Quest, che è l'oggetto di questa tesi, si concentra sulla componente multiplayer, introducendo meccaniche sia competitive che cooperative.

Questa nuova versione del gioco introduce anche un significativo cambiamento nelle meccaniche di movimento. Nella versione precedente, il sistema di controllo del robot era basato sulla fisica, ma i test di gameplay hanno rivelato che questo sistema era troppo macchinoso, lento e complicato. Per risolvere questo problema, è stato introdotto un sistema di movimento basato su una griglia.

Inoltre, essa è in grado di analizzare l'ambiente in tempo reale e occupare le celle in cui sono presenti ostacoli. Questa funzionalità ha permesso di sviluppare un sistema di generazione procedurale per dare agli sviluppatori la libertà di creare ambienti di gioco complessi, delegando la gestione dello spazio di gioco alla griglia stessa. Questi cambiamenti saranno riflessi anche nella modalità singleplayer per mantenere la coerenza in tutto l'universo di gioco.

Nella prossime sezioni verranno presentate velocemente le modalità competitive e cooperative.

2.1.1 Modalità competitive

Per quanto riguarda l'aspetto competitivo, il gioco attualmente propone due modalità: *Time Attack* e *Stop and Go*. Entrambe le modalità condividono lo stesso obiettivo: distruggere le torri nemiche o il robot avversario in un'arena di gioco. Durante le partite, che al momento sono esclusivamente uno contro uno, i giocatori possono generare elementi utili per l'attacco e la difesa.

Le due modalità di gioco differiscono principalmente per quanto riguarda il tempo di programmazione. Nella modalità "*Time Attack*", i due giocatori potranno programmare liberamente il robot ed eseguire il codice, mentre, nella modalità "*Stop and Go*", è previsto un tempo prestabilito per analizzare la situazione e scrivere il codice, una modalità molto più simile a una partita a scacchi in modo sincrono. Una volta scaduto il tempo, il codice viene eseguito e può essere modificato al timeout successivo.

2.1.2 Modalità cooperative

La modalità cooperativa, d'altro canto, consente a 2 o più giocatori di collaborare per risolvere problemi che richiedono la cooperazione di tutti i partecipanti. Attualmente, ci sono alcuni livelli che cambiano dinamicamente ad ogni partita, offrendo così una varietà di situazioni che i giocatori devono affrontare, richiedendo l'uso di un algoritmo diverso ogni volta.

Esistono numerose tipologie di scenari che possono essere sviluppati in nella modalità cooperativa di Code Quest. Le missioni sono progettate in modo da avere una grossa componente cooperativa, costringendo i giocatori a cooperare e organizzare le mosse da fare. Ad esempio, potrebbe essere necessario che un giocatore mantenga premuto un interruttore per permettere all'altro di passare attraverso una porta. Queste sono solo alcune delle molteplici situazioni che possono emergere durante il gioco.

2.2 Conclusioni

In conclusione, Code Quest, con l'introduzione della modalità online, amplia le sue possibilità di interazione, sfruttando ulteriori elementi di gamification. Con le nuove meccaniche e modifiche, dovrebbe essere un strumento facilmente accessibile e funzionale per l'apprendimento della programmazione. Nei prossimi capitoli, verrà presentato il servizio web che si integra con la sezione multiplayer e le nuove funzionalità introdotte.

Capitolo 3

Architettura realizzata

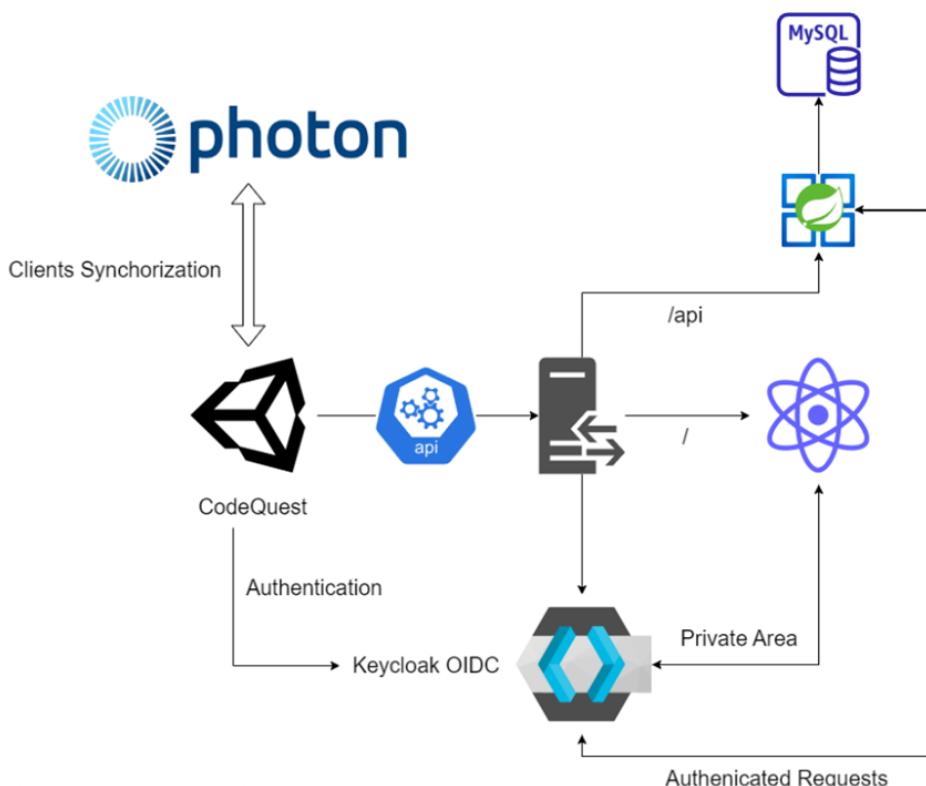


Figura 3.1: Architettura servizio web

I giochi multigiocatore moderni richiedono tipicamente un account utente, il quale funge da identificativo del giocatore e consente l'associazione di statistiche di gioco, progressi e premi accumulati a seguito del raggiungimento di specifici obiettivi.

Nel contesto di Code Quest, è stata progettata un'architettura, illustrata nella Figura 3.1.

In aggiunta, è stata presa la decisione di adottare una piattaforma centralizzata per la gestione e la memorizzazione delle password e degli account utente. Questo approccio consente non solo di garantire un'alta scalabilità del sistema, ma anche di offrire un livello superiore di sicurezza per i dati degli utenti. L'architettura su cui si basa Code Quest è costituita da quattro componenti principali:

- Un database relazionale
- Un servizio di API REST
- Un frontend web
- Un sistema per la gestione degli account.

Nel corso di questo capitolo, verrà eseguita un'analisi dettagliata dell'architettura e del suo funzionamento.

3.1 Database

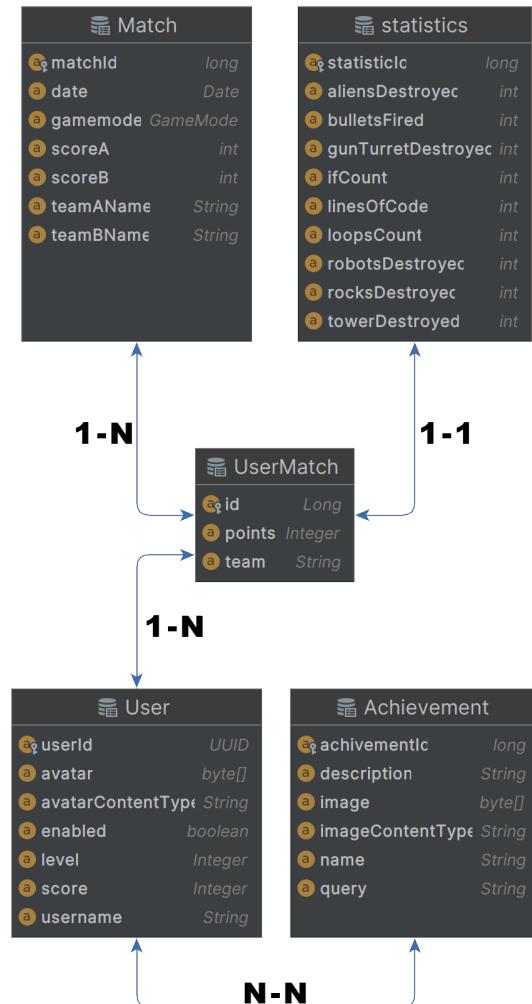


Figura 3.2: Schema ER Database

Il database è l'elemento centrale del servizio web. Infatti, esso permette la memorizzazione dei dati delle partite, dei profili degli utenti e dei vari progressi di quest'ultimi. Per la gestione della base dati è stato scelto il DBMS MySQL.

La Figura 3.2 fornisce una rappresentazione visiva del diagramma ER del database. Esso è composto da quattro entità fondamentali: **User**, **Match**, **Statistics** e **Achievement**. Queste entità sono le colonne portatati del sistema e rappresentano rispettivamente gli utenti, le partite giocate e i traguardi raggiunti all'interno del servizio di Code Quest. Inoltre, è presente un ulteriore entità **UserMatch** che rappresenta la relazione molto a molti tra User e Match che è stata decomposta in due relazioni uno-molti.

3.1.1 User

L'entità "**User**" rappresenta ciascun utente che si è registrato al servizio.

Notiamo che questa tabella contiene esclusivamente i dati necessari per l'applicazione, senza includere alcun campo relativo all'autenticazione, come password ed e-mail. Questa scelta è dovuta al fatto che la registrazione dell'utente avviene su un servizio esterno che gestisce l'autenticazione, argomento che verrà trattato dettagliatamente nella sezione di **Keycloak**.

Ogni utente è registrato con un codice identificativo, condiviso con il sistema di autenticazione, per garantire una corrispondenza univoca tra i due database. Allo stesso modo, lo username è condiviso.

Altri dati memorizzati includono il tipo dell'immagine e lo stato dell'account. Quando un profilo viene eliminato, viene rimosso dal sistema di autenticazione, ma non dal database dell'applicazione. Invece, viene anonimizzato e disabilitato per preservare l'integrità del database. Infine, il database tiene traccia del livello attuale del giocatore e dei punti esperienza accumulati. Quando un giocatore raggiunge mille punti, questi vengono azzerati e il livello aumenta di un'unità.

3.1.2 Match

L'entità "**Match**" rappresenta ciascuna partita disputata. Questa entità è collegata alla tabella degli utenti attraverso una relazione molti a molti, che include dettagli specifici di ogni giocatore relativi alla singola partita, come i punti guadagnati e la squadra di appartenenza.

La tabella delle partite è strutturata con un codice identificativo numerico, la data e l'ora di conclusione della partita, la modalità di gioco (che può essere "**Time attack**" o "**Stop and go**"), i punteggi delle squadre A e B e i rispettivi nomi. Di default, i nomi delle squadre sono impostati come A e B. Tuttavia, poiché le partite attuali sono uno contro uno, vengono inseriti i nomi dei singoli giocatori. Questi campi offrono la flessibilità di inserire anche nomi di squadre composte da più giocatori, una caratteristica che potrebbe essere utilizzata in future modalità di gioco a squadre.

3.1.3 Achievement

Gli "**Achievement**" rivestono un ruolo fondamentale nella gamification. Nel database, sono modellati come entità che possono essere associate a più giocatori attraverso una relazione molti a molti.

Ogni Achievement è identificato da un id numerico , il nome, la descrizione del premio, la query che stabilisce la condizione che il giocatore deve soddisfare per ottenere il badge e dall'immagine che lo raffigura.

La query che si va a scrivere deve seguire una determinata sintassi. Ogni elemento deve essere separato da uno spazio e deve contenere campi e operatori validi. I parametri che si possono interrogare sono tutti quelli presenti nella tabella "Statistics" in aggiunta al numero di partite giocate e vinte ottenibili con le rispettive keyword *matchPlayed* e *matchWon*.

La verifica della query avviene alla creazione, alla modifica e ogni volta che viene aggiunta una partita.

Infine, la query consente di concatenare tramite operatori logici AND e OR per consentire la creazione di badge più articolati.

3.1.4 Statistics

L'entità "**Statistics**" rappresenta la performance di ogni giocatore in ciascuna partita. La tabella include specifiche metriche di gioco, come ad esempio il numero di torri distrutte, il numero di alieni eliminati, il numero di cicli di gioco utilizzati, il numero di righe di codice scritte, e così via.

Queste statistiche sono strettamente collegate alla relazione molti-a-molti "**UserMatch**", che mette in correlazione gli utenti con le partite a cui partecipano. Infatti, all'interno della tabella "UserMatch", è presente un riferimento specifico alla riga delle "Statistics" associata a ciascun giocatore. Questa è una relazione uno-a-uno, in quanto ogni partita giocata da un utente specifico corrisponde a un singolo set di statistiche.

3.2 Servizio API REST

Per interagire efficacemente con il database è necessaria un'interfaccia facilmente utilizzabile sia da utenti umani che da programmi esterni. Questo è possibili grazie alle API REST, un sistema che tramite chiamate HTTP permette di interagire con il database e altri servizi. Questo sistema è stato sviluppato utilizzando il framework Spring usando Java come linguaggio.

Il server agisce come un ponte tra l'utente, o il programma, e il database, permettendo operazioni di recupero, salvataggio, modifica ed eliminazione dei dati. La particolarità di questo servizio è che le operazioni effettuate sul database sono determinate dalla combinazione del percorso specificato nel URL e dal tipo di richiesta HTTP effettuata (che può essere GET, POST, PUT, DELETE, ecc.).

Il servizio web è stato sviluppato seguendo l'architettura a strati *controller*, *service*, *repository* per consentire una migliore suddivisione delle responsabilità e rendendo il codice più organizzato, riutilizzabile e facile da mantenere.

Il *Controller* è il primo punto di contatto per le richieste HTTP in entrata. Funziona come un intermediario tra il client, che invia la richiesta, e il server, che elabora quanto richiesto dal client e restituisce una risposta. Il Controller è responsabile dell'instradamento delle richieste ai metodi di servizio appropriati, basandosi sul metodo HTTP e sull'URL specificato nella richiesta [9].

Il livello *Service*, d'altro canto, rappresenta il cuore dell'applicazione, poiché è qui che risiede la logica di business. Questo strato si occupa dell'elaborazione dei dati, dell'esecuzione dei calcoli e dell'applicazione delle regole aziendali. Il Service interagisce con il livello Repository per recuperare e memorizzare i dati [10].

Infine, il *Repository* è il componente che gestisce l'archiviazione e il recupero dei dati. Interagisce direttamente con il database o con l'origine dei dati, eseguendo operazioni **CRUD** (Create, Read, Update, Delete). Il Repository fornisce un'astrazione sull'origine dei dati sottostante, permettendo al livello Service di interagire con i dati senza dover conoscere le specifiche del database o dell'origine dei dati [9].

Il servizio si articola in una serie di **endpoint**, o percorsi, che facilitano l'esecuzione di varie operazioni. Questi endpoint si suddividono in due categorie: **liberi** e **autenticati**.

Quelli **liberi** sono accessibili a chiunque invii una richiesta e sono tipicamente associati a operazioni di tipo GET, lettura. Questi endpoint non richiedono autenticazione e permettono di recuperare dati senza alterarli.

D'altro canto, quelli **autenticati** sono progettati per operazioni che implicano la modifica, l'aggiunta o la cancellazione dei dati. Questi endpoint richiedono un processo di autenticazione, al termine del quale l'utente riceve un *token* di accesso. Solo tramite esso, l'utente può accedere a queste funzionalità e interagire attivamente con i dati. Esso andrà passato all'interno del header richiesta http nel campo *Authentication* seguendo la seguente formattazione “*Authentication: Bearer [token]*” .

Gli endpoint del servizio sono organizzati in diverse categorie, ognuna delle quali corrisponde a un particolare tipo di dati o funzionalità. Queste categorie sono: **Match**, **User**, **Achievement** e **NoAuth**.

Gli endpoint sotto la categoria **Match** operano sui dati relativi alle partite. Questi possono includere operazioni come il salvataggio di una nuova partita o la ricerca di achievement basata su specifici criteri.

Gli endpoint nella categoria **User** gestiscono le operazioni relative agli utenti. Questi possono includere operazioni come l'aggiornamento delle informazioni di un utente esistente o la cancellazione di un account e recuperare le informazioni di un profilo.

Gli endpoint sotto la categoria **Achievement** operano sui dati relativi agli achievement. Questi possono includere operazioni come l'aggiunta di un nuovo achievement, l'aggiornamento di un achievement esistente, o la ricerca di achievement basata su specifici criteri.

In aggiunta, è possibile effettuare una richiesta POST al percorso "/validate" per inviare la query da verificare. Questa funzionalità permette di centralizzare la logica delle query sul server, facendo sì che sia possibile verificare la validità sintattica della stringa di query prima del suo caricamento. In questo modo, si riducono gli errori e si garantisce che vengano caricate sul sistema solo query correttamente formattate e funzionanti.

Infine, gli endpoint nella categoria **NoAuth** sono quelli per i quali non è richiesta l'autenticazione, ma che comunque permettono richieste diverse da GET. Attualmente, sotto questa categoria, è possibile registrare un nuovo utente attraverso il percorso /noauth/user/new.

Nelle seguenti tabelle sono riassunti i percorsi delle varie richieste, per maggiori informazioni è possibile consultare la pagina di documentazione del servizio realizzata tramite swagger.

Tabella 3.1: Endpoint dell'API

Sezione	Metodo	Endpoint	Auth
Match			
	POST	/match/add	Sì
	GET	/match/{id}	No
	GET	/match/leaderboard/{gameMode}	No
	GET	/match/gameModes	No
	GET	/match/count	No
	GET	/match/all	No
NoAuth			
	POST	/noauth/user/new	No
Achievement			
	PUT	/achievement/update	No
	POST	/achievement/validate	Sì
	POST	/achievement/remove/{userId}/{achievementId}	No
	GET	/achievement/image/{achievementId}	Sì
	POST	/achievement/image/{achievementId}	Sì
	POST	/achievement/create	Sì
	POST	/achievement/add/{userId}/{achievementId}	Sì
	GET	/achievement/{achievementId}	No
	GET	/achievement/check/{userId}/{achievementId}	No
	GET	/achievement/all	No
	DELETE	/achievement/delete/{achievementId}	Sì
Users			
	GET	/user/{id}	No
	PUT	/user/{id}	Sì
	DELETE	/user/{id}	Sì
	GET	/user/{id}/avatar	No
	POST	/user/{id}/avatar	Sì
	GET	/user/count	No
	GET	/user/all	No
	POST	/user/register	Sì

In conclusione, l'API svolge un ruolo cruciale, in quanto viene utilizzata sia dall'applicazione web che gestisce l'interfaccia utente, sia dal gioco stesso.

Alcune operazioni come la gestione del livello e dei badge del giocatore sono affidate direttamente alle API, poiché coinvolgono dati che sono memorizzati nel database. Il sistema inoltre assegna automaticamente un avatar di default a ogni giocatore quando si registra, che potrà modificare successivamente caricando un'immagine a suo piacimento.

Nelle sezioni successive, ci si concentrerà sull'analisi dell'applicazione web che gestisce l'interfaccia utente e, infine, verrà esaminato il sistema di autenticazione gestito da Keycloak.

3.3 Frontend web

Come precedentemente menzionato, la maggior parte dei giochi multigiocatore si avvale di un servizio web, con le API REST come elemento cardine. Tuttavia, la gestione dei dati esclusivamente attraverso chiamate REST può risultare poco intuitiva e scomoda.

Per semplificare l'utilizzo del servizio web, è comune pratica sviluppare un'applicazione web che permetta di interfacciarsi con le API attraverso pagine internet. Esse sono dotate di form e tabelle rendendo più semplice l'inserimento, la visualizzazione dei dati e l'interazione con il servizio.

Per Code Quest, è stata sviluppata un'applicazione web con ReactJS e TypeScript. L'adozione di questo framework, in combinazione con la libreria Chakra UI, ha permesso di creare un'applicazione funzionale con un'interfaccia grafica pulita e responsive in tempi relativamente brevi.

L'applicazione è stata sviluppata seguendo il pattern **Model-View-ViewModel** (MVVM).

Il **Model** è rappresentato dalle classi TypeScript che definiscono gli elementi che devono essere recuperati dalle API. Queste classi forniscono la struttura dei dati che l'applicazione gestisce.

Le **View** sono costituite da tutti i moduli che devono essere visualizzati nell'applicazione. Questi moduli sono mostrati all'interno del componente App, che include una barra superiore fissa e varia solo il contenuto sottostante. Inoltre, lo stato della barra cambia a seconda che l'utente sia autenticato o meno.

Nella Figura 3.3 viene presentato lo schema wireframe che illustra l'interfaccia del profilo utente.

Il **ViewModel** contiene i file JavaScript che permettono di recuperare i dati dalle API e di trasmetterli alla View. Questo strato agisce come un intermediario tra il Model e la View, gestendo la logica di visualizzazione e l'interazione dell'utente.

L'approccio MVVM permette di separare la logica dell'applicazione, la gestione dei dati e l'interfaccia utente, rendendo l'applicazione più organizzata, manutenibile e scalabile.

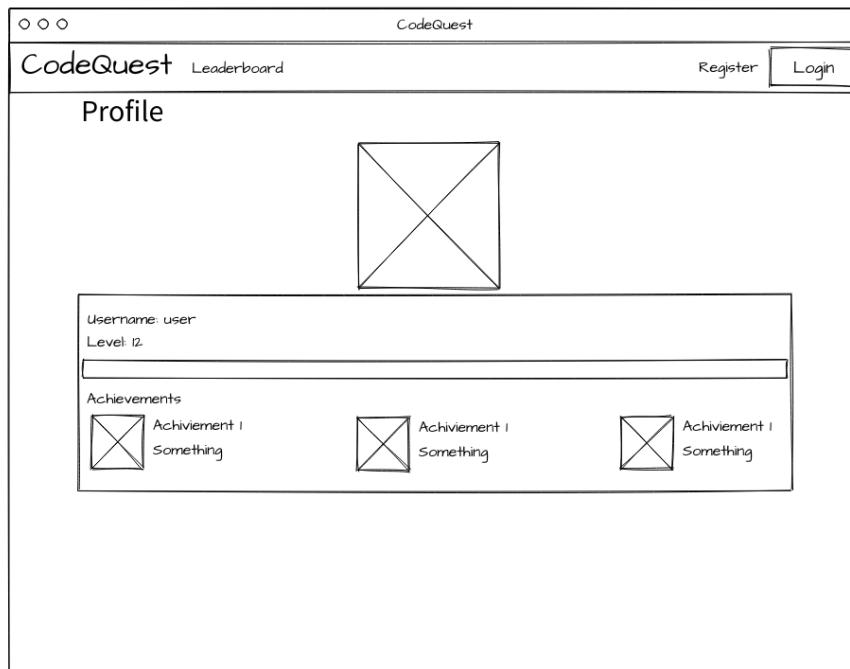


Figura 3.3: Schema wireframe della pagine del profilo utente

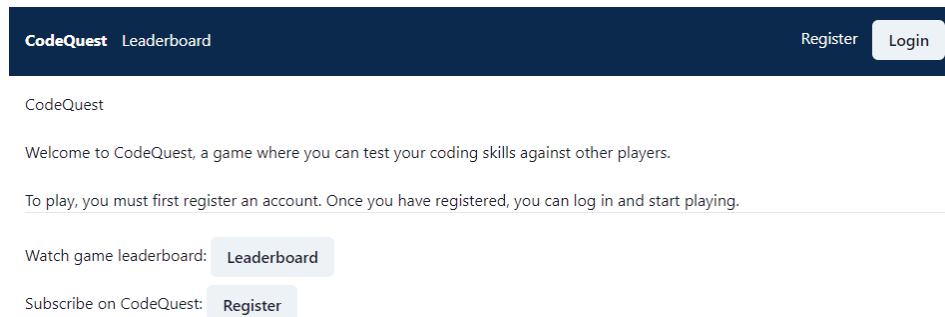
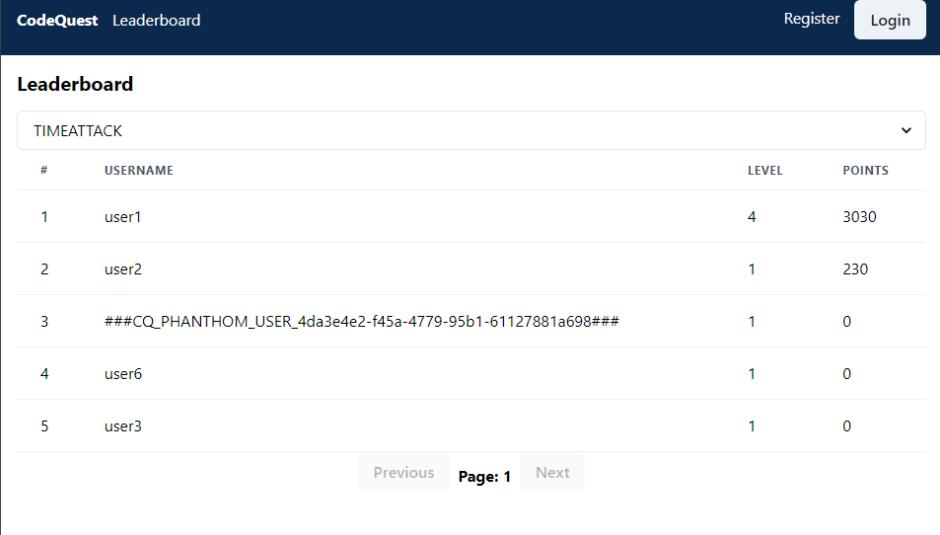


Figura 3.4: Home dell'applicazione

L'applicazione ha una homepage 3.4 che presenta il gioco e guida l'utente verso la registrazione oppure alla visualizzazione della classifica generale.

La classifica come mostrato in Figura 3.5 mostra gli utenti con le relative posizioni e punteggi.



The screenshot shows a leaderboards page for the game TIMEATTACK. At the top, there are 'Register' and 'Login' buttons. Below that, the title 'Leaderboard' is displayed, followed by a dropdown menu set to 'TIMEATTACK'. The main content is a table with columns: '#', 'USERNAME', 'LEVEL', and 'POINTS'. The data is as follows:

#	USERNAME	LEVEL	POINTS
1	user1	4	3030
2	user2	1	230
3	###CQ_PHANTHOM_USER_4da3e4e2-f45a-4779-95b1-61127881a698##	1	0
4	user6	1	0
5	user3	1	0

At the bottom of the table, there are 'Previous' and 'Next' buttons, and the text 'Page: 1'.

Figura 3.5: Classifica

Cliccando sul nome utente di ogni giocatore è possibile visionare il profilo, Figura 3.6.

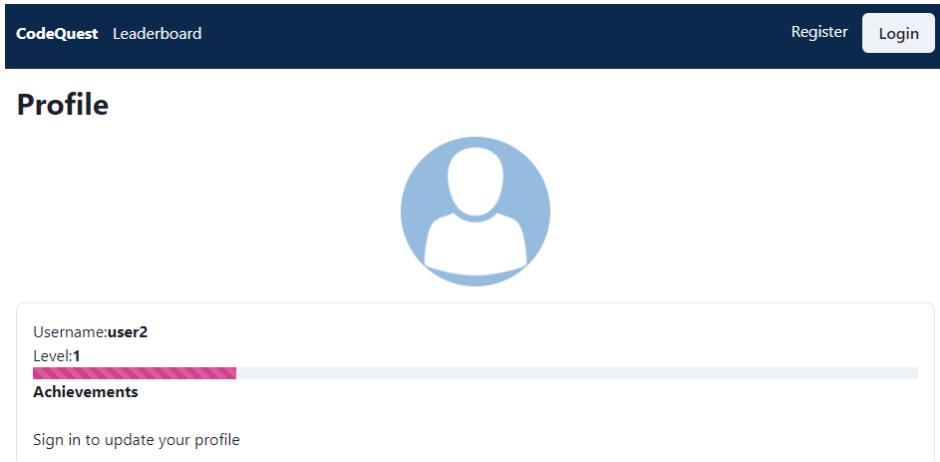


Figura 3.6: Profilo giocatore

La pagina del profilo mostra l'avatar, lo username, il livello attuale del giocatore, quanti punti esperienza ha attualmente per raggiungere il livello successivo ed infine eventuali badge.

Cliccando su *Register* si viene reindirizzati alla schermata di registrazione, Figura 3.7, nella quale vanno inseriti i dati richiesti per poter registrare un nuovo utente.

The screenshot shows a registration form titled "Registration". It includes fields for "Username *", "Email *", "Password *", and "Confirm Password *". Each field has a placeholder text: "Enter your username", "Enter your email", "Enter your password", and "Confirm your password" respectively. There are "Register" buttons at the bottom of each group of fields and a "Register" button at the very bottom of the page. The top navigation bar features the "CodeQuest" logo and a "Leaderboard" link, along with "Register" and "Login" buttons.

Figura 3.7: Form di registrazione

Per garantire una corretta registrazione degli utenti, è necessario che sia lo username che l'e-mail siano univoci. Inoltre, le password inserite devono corrispondere.

La registrazione automatica su Code Quest è disponibile per gli utenti che hanno già un profilo su Keycloak e che sono autenticati. Questo significa che se un utente ha un account su Keycloak, può accedere automaticamente a Code Quest senza la necessità di effettuare una registrazione separata, infatti basterà fare il login. Questa funzionalità consente anche di poter registrarsi, se configurato su keycloak, tramite provider esterni come Google, Github, Facebook ecc.

Al momento, non sono state implementate specifiche policy di sicurezza per le password, ma è consigliabile adottare tali norme per migliorare la sicurezza dell'applicazione. Inoltre, esse possono essere impostate direttamente da Keycloak.

Se l'utente inserisce dati non validi, il sistema lo informerà attraverso un pop-up, generato dalla libreria SweetAlertJS.

Questo meccanismo di feedback immediato aiuta gli utenti a correggere eventuali errori durante la fase di registrazione.

Una volta effettuata la registrazione si verrà reindirizzati alla schermata di login fornita da Keycloak, Figura 3.8.

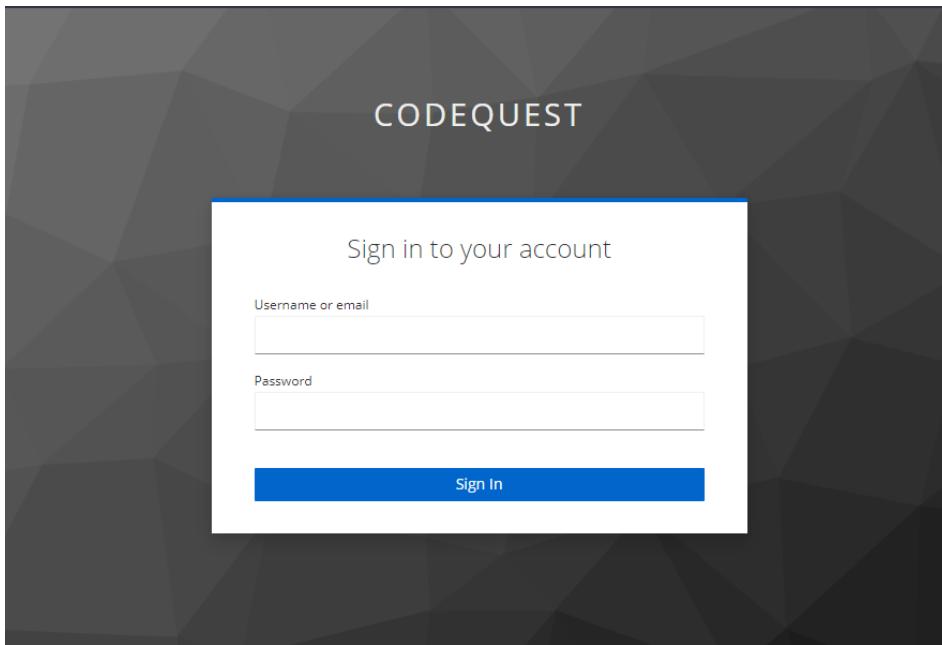


Figura 3.8: Schermata di login

Una volta effettuato l'accesso ci sono due possibili scenari:

- Primo scenario: se si tratta di un utente normale si potrà vedere il profilo, aggiornare l'avatar ed eliminare l'account. Questo consente di avere il pieno controllo dei propri dati da parte dell'utente, il quale deve poter visualizzare, modificare ed eliminare, Figura 3.9.
- Secondo scenario: se si dispone di privilegi di amministratore, che conferiscono il pieno controllo sul sistema. Attualmente, con questi privilegi, è possibile eseguire una serie di operazioni di gestione. Gli amministratori possono eliminare gli utenti 3.10, aggiungere, modificare e cancellare i badge. Inoltre, hanno la possibilità di visualizzare le partite giocate, ordinate dalla più recente. Queste funzionalità forniscono agli amministratori gli strumenti necessari per gestire efficacemente il sistema.

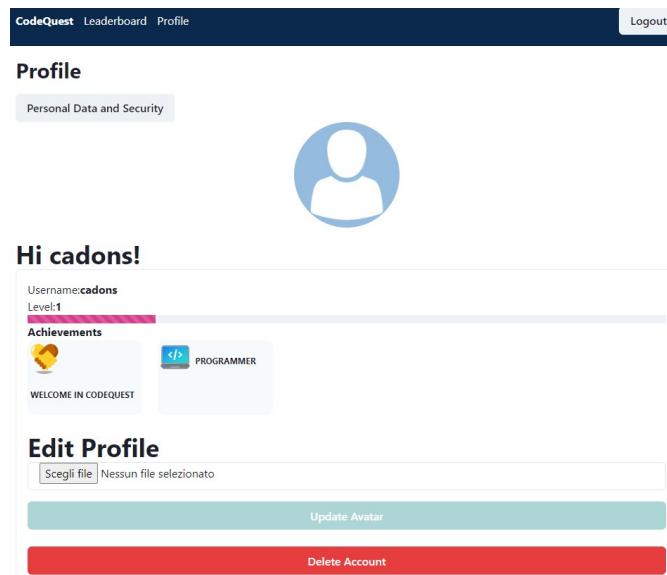


Figura 3.9: Schermata di modifica del profilo

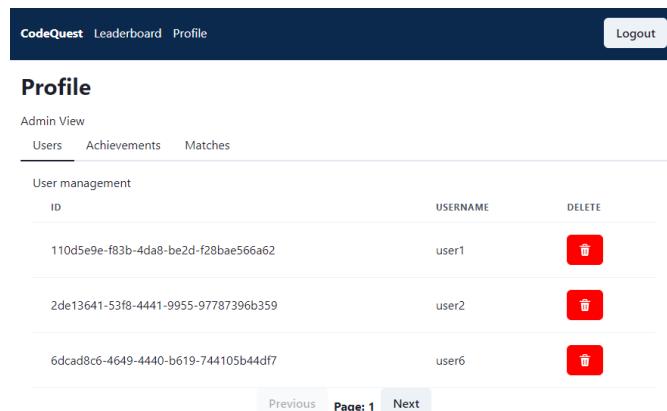


Figura 3.10: Schermata di gestione degli utenti

In conclusione, il servizio web fornisce le funzionalità essenziali per la gestione del sistema. Tuttavia, c'è ampio spazio per futuri sviluppi e miglioramenti. In futuro, il servizio potrebbe essere ampliato per consentire l'esecuzione di operazioni più complesse e la visualizzazione di dati aggiuntivi, migliorando ulteriormente l'efficienza e l'efficacia del sistema.

3.4 Autenticazione e sicurezza

Come evidenziato nei paragrafi precedenti, l'autenticazione è un aspetto che è stato trattato particolare attenzione nel corso dello sviluppo. Questa enfasi sull'autenticazione è motivata dalla necessità di avere un sistema centralizzato che si occupi esclusivamente della protezione delle risorse. Un sistema di questo tipo permette di registrare un profilo una sola volta e di utilizzarlo su più servizi.

Questo concetto è noto come Single Sign-On (SSO), un approccio che migliora notevolmente l'esperienza dell'utente riducendo la necessità di ricordare molteplici credenziali e facilitando l'accesso a diverse piattaforme e servizi.

In questa sezione verrà spiegato il funzionamento del protocollo OAuth 2.0, OpenID Connect e di Keycloak.

3.4.1 OAuth 2.0 e OpenID Connect

Oggi giorno la sicurezza delle applicazioni è sempre più importante e con il tempo si sono sviluppati diversi protocolli che consentono di mantenere sicure le informazioni degli utenti. Uno di questi è OAuth 2.0, un protocollo standard per l'autorizzazione. Questo protocollo è progettato con un focus particolare sugli sviluppatori di applicazioni client, offrendo un framework semplice per proteggere le applicazioni [11].

L'obiettivo principale di OAuth 2.0 è di permettere alle applicazioni di terze parti di accedere a aree limitate di un servizio web, come i dati dell'utente su un social network, senza la necessità di gestire o conoscere le credenziali dell'utente [12].

Nel contesto di OAuth, ci sono due elementi chiave: il **server di autorizzazione** e il **server risorsa**. Il **server di autorizzazione** è l'entità che autentica il proprietario della risorsa e rilascia il token di accesso, e se necessario, il token di aggiornamento, una volta che l'utente è stato autorizzato. Può autenticare il client e validare le concessioni di autorizzazione [12].

Il **server risorsa** è l'entità che può rispondere alle richieste di risorse protette utilizzando il token di accesso. Il server delle risorse valida il token di accesso e, se valido, serve le richieste [12]. In OAuth 2.0, ci sono due tipi di token: i **token di accesso** e i **token di aggiornamento**. I **token di accesso** sono utilizzati per accedere alle risorse protette. Sono stringhe che rappresentano l'autorizzazione dell'utente. Possono avere diversi formati e possono contenere diverse informazioni, consentendo il supporto a diversi sistemi di autenticazione [12].

I **token di aggiornamento**, invece, sono utilizzati per ottenere nuovi token di accesso. Vengono rilasciati al client durante l'operazione di autenticazione e sono inviati al server quando il token di accesso in possesso del client è scaduto. Se il token di aggiornamento è valido, il server restituirà un nuovo token di accesso, senza richiedere nuovamente l'autenticazione [12].

Il protocollo segue il flusso mostrato nella seguente Figura 3.11

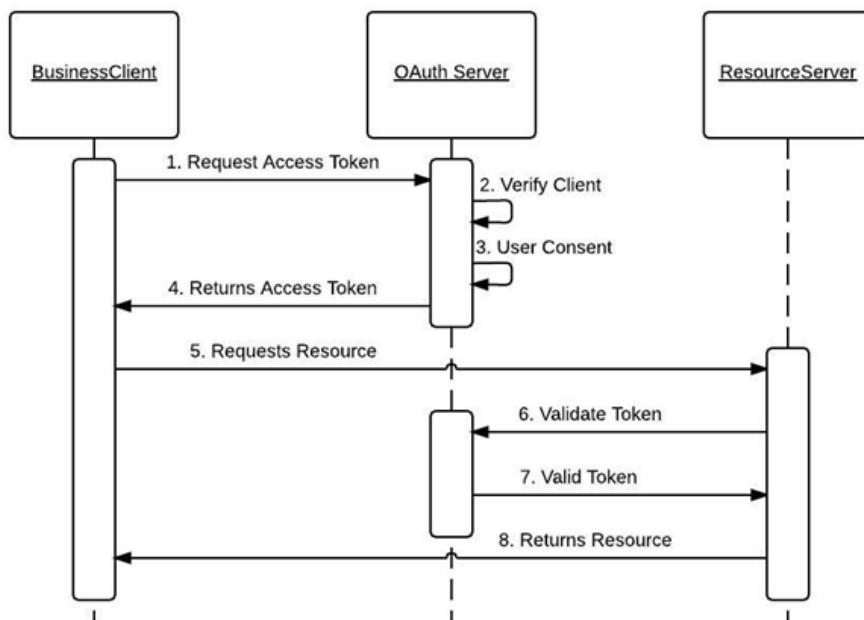


Figura 3.11: Diagramma di funzionamento di OAuth 2.0

Il processo di autorizzazione in OAuth 2.0 inizia con la richiesta di un token di accesso da parte dell'applicazione client al server di autenticazione. Questo passaggio avviene dopo che l'utente ha fornito le proprie credenziali, solitamente attraverso un processo di login, e ha concesso all'applicazione client il permesso di accedere alle sue risorse.

Una volta che l'utente è autenticato e l'autorizzazione è confermata, il server di autenticazione emette un token di accesso. Questo token è una stringa alfanumerica che rappresenta l'autorizzazione specifica concessa dall'utente.

È importante notare che il token di accesso non contiene informazioni sensibili dell'utente, ma piuttosto codifica i diritti di accesso dell'applicazione client [12].

Dopo aver ottenuto il token di accesso, l'applicazione client può procedere con la richiesta delle risorse desiderate. Questo viene fatto inviando la richiesta al server delle risorse, insieme al token di accesso per l'autenticazione.

Il server delle risorse, a sua volta, verifica la validità del token di accesso. Se il token di accesso è valido, la richiesta dell'applicazione client viene elaborata; in caso contrario, verrà rifiutata. Questo processo assicura che solo le applicazioni client autorizzate possano accedere alle risorse protette, migliorando la sicurezza dell'intero sistema [12].

I token di aggiornamento non sono obbligatori. Se non sono abilitati, l'utente dovrà autenticarsi nuovamente quando il token di accesso scade [12].

Quando il token di accesso non è più valido e i token di aggiornamento sono abilitati, il client invia una richiesta di aggiornamento inviando il token di refresh. Se il server di autorizzazione accetta il token, rilascerà un nuovo token di accesso e, optionalmente, un nuovo token di aggiornamento. I token di aggiornamento sono riservati esclusivamente al server di autenticazione e non devono mai essere inviati al server delle risorse [12]. Il flusso che segue il client per aggiornare il token di accesso è quello mostrato in Figura 3.12

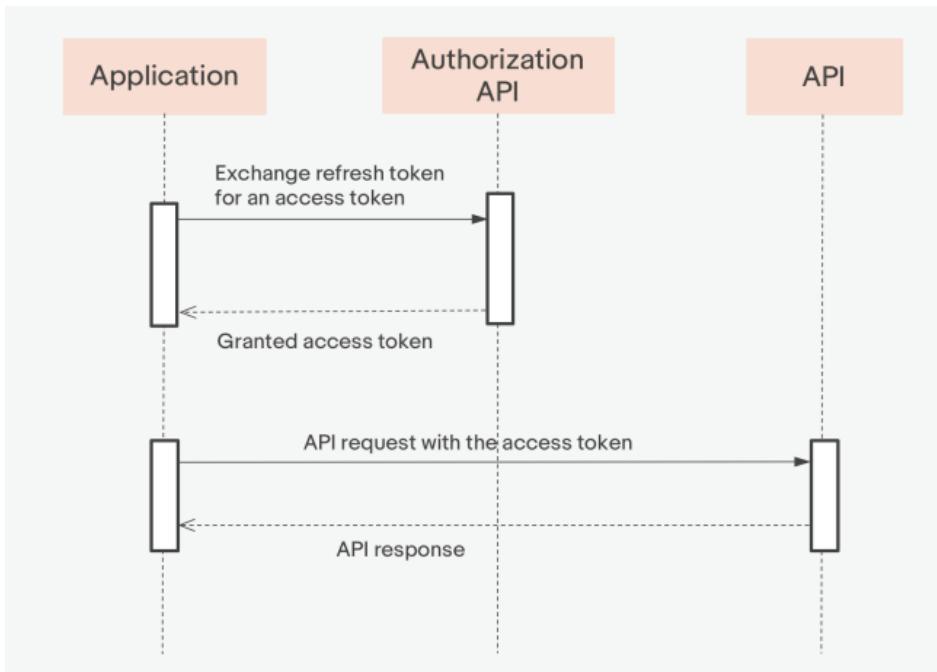


Figura 3.12: Diagramma di funzionamento di OAuth 2.0 con il token di refresh

Nel contesto di Code Quest, il protocollo utilizzato non è direttamente OAuth 2.0, ma **OpenID Connect (OIDC)**, un protocollo che si basa sul framework OAuth 2.0. OIDC semplifica il processo di verifica dell'identità degli utenti, facendo affidamento sull'autenticazione eseguita da un server di autenticazione [13].

OpenID Connect può essere considerato come una versione più avanzata del protocollo SAML. Mentre SAML era limitato all'accesso solo per le applicazioni

web, OIDC estende le sue funzionalità per consentire l'accesso anche alle applicazioni native e mobili [13]. Questa funzionalità aggiuntiva ha permesso l'integrazione dell'autenticazione tramite OpenID Connect in Unity, reindirizzando l'utente in una sessione del browser che autorizza il client.

Un elemento fondamentale di OpenID Connect è il **realm**, un modello che rappresenta la parte di spazio URL per cui è valida una richiesta di autenticazione OpenID. Un realm è progettato per dare all'utente finale un'indicazione dell'ambito della richiesta di autenticazione [14].

Il processo di autenticazione e autorizzazione in OpenID Connect differisce leggermente da quello di OAuth 2.0, in quanto deve gestire anche l'autenticazione dell'utente. Come illustrato in Figura 3.13, l'utente inizia con una richiesta di autenticazione. Se l'accesso è concesso, il server avvia una sessione per l'utente e rilascia un codice di autorizzazione. Successivamente, il client utilizza questo codice per fare una richiesta al *token endpoint*, che ha il compito di restituire il token di accesso. Infine, l'utente può utilizzare il token di accesso per accedere al server risorsa [13].

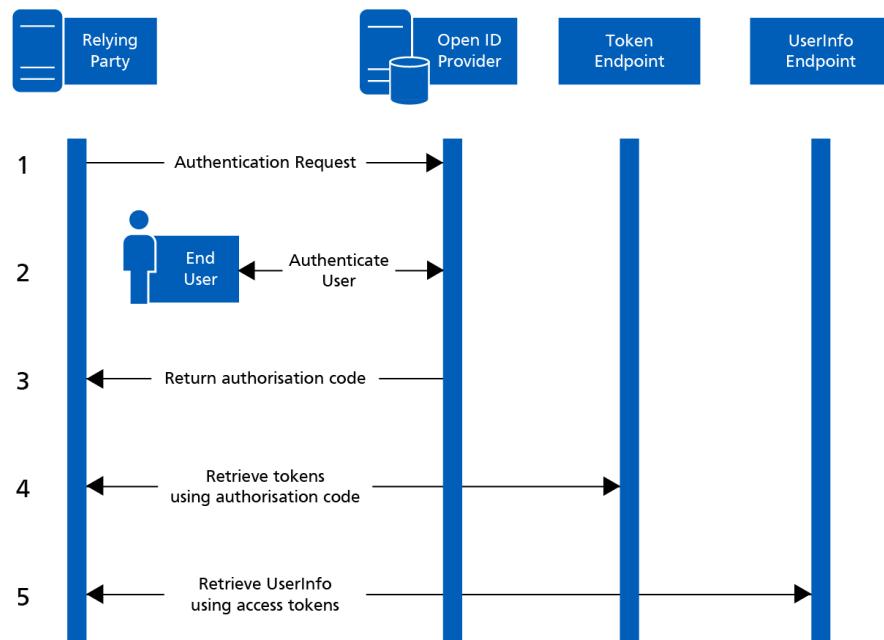


Figura 3.13: Diagramma di funzionamento di OAuth 2.0 con il token di refresh

Il token rilasciato è un token JWT (JSON Web Token), una stringa alfanumerica codificata che contiene diverse informazioni, come l'ID dell'utente, la data di

scadenza del token e i ruoli dell'utente. Queste informazioni sono codificate nel token in un formato che può essere facilmente decodificato in JSON e utilizzato per l'autenticazione e l'autorizzazione [13].

In conclusione, l'importanza della sicurezza delle applicazioni in un mondo sempre più digitalizzato non può essere sottovalutata. Protocolli come OAuth 2.0 e OpenID Connect rappresentano strumenti fondamentali per garantire la protezione delle informazioni degli utenti e per gestire in modo efficace l'autenticazione e l'autorizzazione.

Questi protocolli offrono un framework robusto e flessibile per la gestione dell'accesso alle risorse, consentendo allo stesso tempo una facile integrazione con diverse applicazioni e servizi.

Nelle sezioni successive verrà discusso il ruolo di Keycloak e su come è stato integrato questo protocollo con i vari elementi.

3.5 Keycloak

Keycloak è una soluzione software open source progettata per fornire autenticazione, autorizzazione, gestione dell'identità e gestione degli accessi in maniera sicura e scalabile.

Software come Keycloak e simili sono molto usati nelle moderne applicazioni distribuite, poiché consente di gestire gli utenti e mantenere al sicuro i dati sensibili in un unico servizio centralizzato [15].

Il software offre diverse funzionalità, nei prossimi paragrafi verranno analizzate quelle principali.

3.5.1 Single-Sign On

Il Single Sign-On (SSO) è un meccanismo di autenticazione che consente agli utenti di accedere a più applicazioni o servizi utilizzando un unico set di credenziali. Questo approccio riduce la complessità e aumenta l'efficienza per gli utenti, eliminando la necessità di ricordare e inserire diverse combinazioni di nome utente e password. Keycloak è una delle piattaforme che supportano e implementano la funzionalità SSO [16].

3.5.2 Identity Brokering e Social Login

Keycloak offre diverse opzioni di integrazione che consentono agli utenti di autenticarsi utilizzando provider di identità esterni come Google, Facebook, GitHub e sistemi come LDAP e Active Directory. Questa funzionalità, spesso chiamata "social login" o "autenticazione federata", semplifica notevolmente il processo di autenticazione per gli utenti, eliminando la necessità di creare e ricordare un nuovo set di credenziali per ogni servizio. Inoltre, Keycloak supporta l'autenticazione basata su standard industriali come OpenID Connect e SAML 2.0, portando con se i vantaggi discussi nei paragrafi precedenti [16].

3.5.3 Admin console

Uno dei principali vantaggi di Keycloak è l'interfaccia web di gestione. La sua semplicità di utilizzo consente agli amministratori di configurare facilmente l'intero sistema compresi i provider esterni. Ciò significa che l'integrazione con quest'ultimi può essere realizzata senza dover scrivere o modificare il codice dell'applicazione, rendendo il processo di configurazione molto più semplice e diretto [16].

3.5.4 Account console

Keycloak fornisce un'interfaccia web che gli utenti possono utilizzare per gestire i propri profili. Questa interfaccia consente agli utenti di aggiornare le proprie informazioni personali, cambiare le password, gestire le sessioni attive e persino configurare opzioni di sicurezza avanzate come l'autenticazione a più fattori [16].

3.5.5 Servizi di autorizzazione e API

Keycloak è un software molto versatile, infatti, esso espone un servizio RESTful API, che offre la possibilità di interfacciarsi con il servizio consentendo di sviluppare un'interfaccia di gestione degli utenti personalizzata. Attraverso queste API, gli sviluppatori possono effettuare una vasta gamma di operazioni, come creare, modificare, eliminare utenti oppure gestire gruppi e ruoli. In Code Quest la procedura di registrazione ed eliminazione dei profili si basa su questo servizio.

Keycloak supporta anche la gestione dei permessi, consentendo agli amministratori di controllare esattamente quali risorse un utente o un gruppo di utenti può accedere. Questo livello di controllo può essere esteso non solo alle risorse dell'applicazione, ma anche ai vari componenti e servizi di Keycloak. Ad esempio, è possibile limitare l'accesso all'interfaccia di amministrazione di Keycloak o ad alcune specifiche API RESTful a determinati utenti o gruppi, migliorando così la sicurezza complessiva del sistema [16].

3.5.6 Conclusioni

In conclusione, l'adozione di Keycloak nel contesto del servizio web di Code Quest ha un'importanza fondamentale. Grazie alla sua capacità di semplificare significativamente la gestione degli utenti, questa scelta ha consentito di delegare a Keycloak la gestione delle informazioni di identificazione dell'utente, riducendo così la complessità dell'applicazione e minimizzando la quantità di dati sensibili gestiti direttamente dall'applicazione stessa.

La presenza di Keycloak all'interno dell'architettura di Code Quest porta con sé ulteriori benefici. In particolare, permette all'applicazione di non doversi occupare direttamente delle diverse operazioni legate alla gestione degli utenti, come l'autenticazione, l'autorizzazione, la gestione delle sessioni, e l'autenticazione multi-fattore, poiché queste funzionalità sono gestite in modo sicuro e efficiente da Keycloak.

Nelle prossime sezioni, verrà approfondito come configurare Keycloak per Code Quest e come impostare i vari servizi per interfacciarsi correttamente con il sistema di Single Sign-On. Queste informazioni permetteranno di comprendere pienamente il valore che Keycloak apporta al progetto e come può semplificare lo sviluppo e la gestione delle applicazioni distribuite.

3.6 Configurazione del servizio web e di Keycloak

La configurazione del servizio web è una fase fondamentale per il corretto funzionamento di tutta l'architettura. In questa sezione verrà mostrato nel dettaglio come effettuare il deploy del servizio e come configurare il servizio di autenticazione.

3.6.1 Docker-compose

Il servizio web è progettato per un'installazione agevole in un ambiente basato su container, utilizzando specificamente Docker. Per effettuare il deploy, assicurarsi che Docker, insieme al supporto per Docker Compose, sia installato sulla macchina host.

Per iniziare, aprire la cartella del servizio web tramite il terminale. All'interno, è presente un file chiamato *docker-compose.yml*. Questo file contiene l'intera configurazione del servizio web, suddivisa nei vari sotto-servizi che necessitano di essere installati.

Ci sono alcuni parametri che vanno modificati, principalmente gli indirizzi IP, che dovranno corrispondere all'IP della macchina host, o in fase di produzione, al nome del dominio e le credenziali d'accesso.

È importante notare che l'uso di "*localhost*" per indicare la macchina stessa non è consigliato all'interno dell'ambiente Docker. Questo perché in tale contesto, "*localhost*" si riferisce al container stesso, non al host. Usare Docker è paragonabile a gestire diverse macchine virtuali indipendenti, ognuna delle quali ospita un servizio specifico.

Durante la fase di distribuzione finale, si dovrà anche configurare il servizio HTTPS con il relativo certificato. È fondamentale completare quest'ultima operazione durante questa fase, in particolare per il servizio di autenticazione. In caso contrario, i dati che viaggiano sulla rete, inclusi quelli di autenticazione, saranno visibili agli utenti malintenzionati che monitorano il traffico di rete, ovvero coloro che effettuano lo "*sniffing*".

In fase di produzione, sarebbe anche utile dividere il dominio in sotto-domini, in modo da avere un dominio per ciascuno dei sotto-servizi. Ad esempio, si potrebbe avere l'autenticazione su *auth.Code Quest.ch*.

Nel corso della migrazione dei servizi da HTTP a HTTPS, potrebbe essere necessario modificare le porte esposte dai vari servizi. In alternativa è possibile configurare il proxy per funzionare in HTTPS e lasciare il traffico interno a docker in chiaro. Quest'ultima operazione rimane una procedura sicura se la rete

di container è isolata e viene eseguita sulla stessa macchina host. Inoltre, keycloak nel file di configurazione ha una riga commentata che lo imposta per la distribuzione dietro ad un proxy; in fase di deploy togliere il commento da questa e commentare la riga dedicata allo sviluppo. I servizi esposti nel file di deploy sono riassunti nella seguente tabella:

Nome Servizio	Descrizione
api	Servizio Spring che fornisce le API REST di Code Quest
frontend	Sito web
db	Database di Code Quest
keycloak	Servizio di autenticazione
postgres_svr	Database di keycloak
nginx	Il proxy attualmente indirizza tutte le richieste al frontend, tranne quelle provenienti da /api/, che vengono dirette alle API accessibili tramite http://myhost/api/

Tabella 3.2: Descrizione dei servizi

Dopo aver configurato adeguatamente tutti i servizi, si può procedere all'installazione eseguendo il comando '*docker-compose up*'. Questo comando istruisce Docker a costruire le immagini del servizio API e del frontend, e successivamente avviare tutti i servizi. Durante questa fase di avvio, è importante attendere che tutti i servizi siano completamente inizializzati e operativi, senza alcun errore.

Se uno o più servizi dovessero incontrare problemi, tenteranno di riavviarsi automaticamente. Questa impostazione può essere modificata cambiando la proprietà "restart" all'interno del file *docker-compose.yml*. Se un servizio non riesce ad avviarsi correttamente, è consigliabile controllare i log mostrati sul terminale o nei singoli container per individuare eventuali errori di configurazione.

3.6.2 Configurazione di Keycloak

Al primo avvio si riscontreranno continui errori sul server delle API. Questo è dovuto al fatto che il servizio di autenticazione non stato ancora configurato per operare con questi elementi. Per avviare correttamente l'intero servizio, è necessario eseguire le seguenti operazioni:

- Accedere alla pagina di Keycloak.
- Collegarsi alla Administration Console.
- Effettuare l'accesso come amministratore utilizzando la password imposta-ta nel file di configurazione.
- Selezionare "*master*" e creare un nuovo Realm.
- Caricare la configurazione che si trova nella cartella
`"backend/keycloakConfiguration/Code QuestRealm.json"`
- Selezionare "create". Da questo momento in poi, il Realm sarà disponibile e i servizi dovrebbero avviarsi senza errori.
- Aggiungere un utente nella sezione "*Users*" che abbia le stesse credenziali inserite nei campi `KEYCLOAK_ADMIN_USER` e
`KEYCLOAK_ADMIN_PASSWORD` nel servizio API.
- Assegnare a questo utente il ruolo di *SERVER*. Nota: questo utente non dovrebbe mai essere utilizzato per l'amministrazione o il gameplay, è riser-vato solo per la gestione degli utenti tramite API.
- Infine, nei client "*frontend*" e "*Code Quest_client*", impostare gli indirizzi appropriati nei campi Root URL e Home URL.

A questo punto, Keycloak è pronto per accettare le registrazioni dal frontend. Tutte le funzionalità di sicurezza, come l'autenticazione a più fattori o l'integrazione con provider esterni, possono essere gestite tramite la console web appena utilizzata. Inoltre, è vivamente consigliato modificare la password dell'amministratore del SSO con una molto forte e applicare il principio del minimo privilegio per limitare il rischio di abusi di potere o di attacchi esterni.

3.7 Integrazione di OpenID nei vari servizi

3.7.1 Resource Server e Spring

La configurazione del server che fornisce le API del servizio è stata eseguita utilizzando la libreria OAuth2.0 Resource Server. Questa libreria è un componente di Spring Security e consente di configurare un server di risorse conforme al protocollo OAuth 2.0. Le specifiche della configurazione di sicurezza del servizio sono contenute nella classe *SecurityConfig*, situata all'interno del package security. Questa classe include tutte le impostazioni necessarie per garantire la sicurezza del servizio e per il suo funzionamento con i protocolli discussi precedentemente.

Nel metodo *securityFilterChain*, è possibile trovare la configurazione dettagliata. Questo include le regole che determinano quali percorsi devono essere protetti e con quale tipo di chiamata REST e la configurazione del server di risorse che opera attraverso token JWT. In relazione a ciò, è stato scelto di disabilitare le sessioni, rendendo il server stateless. Di conseguenza, tutte le chiamate richiederanno sempre il token di accesso. All'interno della stessa classe, c'è un metodo chiamato *jwtDecoder*. Questo ha il compito di decodificare il token usando l'*issuerURL*, un URL fornito dal provider di autenticazione, in questo caso, Keycloak.

Per gestire i permessi degli utenti, all'interno del package security, è presente la classe *JwtAuthConverter*. Questa ha il compito di prelevare i permessi dell'utente dal token JWT e di comunicare a Spring che la richiesta in entrata ha tali permessi. Questo ci permette di filtrare le richieste in base ai permessi dell'utente.

L'interfaccia *ResourceOwnerVerifier* e la sua implementazione ci permettono di verificare se l'utente che sta accedendo alla risorsa è il legittimo proprietario. Nel caso degli amministratori, possono modificare tutte le risorse. Questa verifica deve essere fatta prima della modifica, nel metodo di servizio.

Nel package keycloak, c'è un'interfaccia che consente di interagire con il provider di autenticazione e un'implementazione per Keycloak. Questa classe esegue chiamate REST a Keycloak per aggiungere ed eliminare utenti.

Infine, nel file di configurazione di Spring, sono presenti tutti i parametri necessari per il funzionamento del servizio, tra cui l'*issuer-uri* e il *jwt-set-uri*. C'è anche una sezione dedicata ai JWT, nella quale si trova l'ID del client, nel quale sono elencati i permessi.

3.7.2 OpenID Connect e React

L'applicazione web sviluppata per il frontend implementa l'autenticazione tramite l'uso del protocollo OpenID Connect (OIDC) grazie alla libreria *react-oidc-context*. Nel codice principale dell'applicazione, index.jsx, viene definita la configurazione di OIDC. Essa comprende parametri fondamentali quali l'autorità (l'URL del server OIDC), l'ID del client (rilevato dal server OIDC), l'URI di rein-dirizzamento su cui si verrà indirizzati dopo l'autenticazione, il realm e l'URI di post logout.

```

1 const oidcConfig = {
2   authority: process.env.REACT_APP_OIDC_URL as string,
3   client_id: process.env.REACT_APP_OIDC_CLIENT_ID as string,
4   redirect_uri: window.location.protocol+... ,
5   realm: process.env.REACT_APP_OIDC_REALM as string,
6   post_logout_redirect_uri:window.location.protocol... ,
7   // ...
8 };

```

Listing 3.1: index.jsx

Questi dettagli vengono passati al componente '*AuthProvider*' che avvolge l'intera applicazione e gestisce l'autenticazione con OIDC.

```

1 root.render(
2 <React.StrictMode>
3   <AuthProvider {...oidcConfig} >
4     <ChakraProvider theme={theme}>
5       <BrowserRouter>
6         <App/>
7       </BrowserRouter>
8     </ChakraProvider>
9   </AuthProvider>
10 </React.StrictMode>
11 );

```

Listing 3.2: index.jsx

Per l'autenticazione e l'autorizzazione degli utenti sono implementate diverse funzioni. Ad esempio, '*isAdmin*' verifica se un utente ha il ruolo di amministratore decodificando il token JWT fornito dal server OIDC. Un processo analogo viene seguito per determinare se un utente esiste, utilizzando la funzione '*isUser*'. Inoltre, la funzione '*getUserId*' estrae l'ID dell'utente dal token JWT.

```

1 export function isAdmin(auth: AuthContextProps)
2 :boolean{
3     // ...
4 }
5 export async function isUser(auth: AuthContextProps)
6 : Promise<boolean> {
7     // ...
8 }
9 export function getUserId(auth: AuthContextProps)
10 :string{
11     // ...
12 }
```

Listing 3.3: SecurityUtility.jsx

La registrazione e il login degli utenti sono gestiti dal componente ‘*LoginCallback.tsx*’. Esso utilizza l’hook ‘*useAuth*’ per ottenere lo stato corrente dell’autenticazione dell’utente e, se l’utente risulta autenticato, prova a registrarlo nel sistema con la funzione ‘*registerIfNotExists*’

```

1 export function LoginCallback() { ....
2     const auth = useAuth();
3     useEffect(() => { ....
4         const registration = async () => {
5             if (auth.user) {
6                 await registerIfNotExists();
7             }
8         }
9         registration();
10    }, [
11        auth.isAuthenticated,
12        auth.activeNavigator,
13        auth.isLoading,
14        auth.signinRedirect]);
15 }
```

Listing 3.4: LoginCallback.jsx

In conclusione, l’applicazione React utilizza OIDC per gestire l’autenticazione degli utenti e assegnare loro ruoli specifici.

3.7.3 OpenID Connect e Unity

Nel processo di integrazione dell'autenticazione del servizio web con il client Unity in cui viene eseguito il gioco, è stata utilizzata una libreria esterna chiamata *IdentityModel.OidcClient*. Questa dipendenza gestisce l'autenticazione tramite OIDC nelle applicazioni C#.

Sebbene esista un supporto nativo per il protocollo OIDC in Unity, tale supporto è orientato verso servizi già in funzione, dotati di certificato ed esposti su internet, il che lo rende meno adatto allo sviluppo in locale su un servizio proprio. Inoltre, tale servizio richiede la registrazione del progetto su un account Unity, limitando così la portabilità del progetto.

Per facilitare la configurazione dei componenti, è stato sviluppato un componente *ScriptableObject*, denominato *OidcConfiguration*, che contiene tutti i parametri necessari per la configurazione del servizio di autenticazione.

```

1 public class OidcConfiguration : ScriptableObject
2 {
3     public string IssuerURL;
4
5     public string ClientId;
6
7     public string ClientSecret;
8
9     public string Scope;
10
11    public bool RequireHttps;
12
13    public string TokenStoragePassword= "default_psw";
14
15    public string TokenStorageSalt= "default_salt";
16 }
17

```

Listing 3.5: OidcConfiguration.cs

Questa classe contiene tutti i dati necessari per configurare il servizio di autenticazione OIDC. Oltre ai parametri precedentemente menzionati, include il *ClientSecret*, un parametro per i client di tipo confidenziale. In questo caso, il campo *ClientSecret* rimarrà vuoto, poiché vengono utilizzati solo client pubblici.

Il campo *Scope* specifica lo scope che il client utilizzerà, che in questo caso sarà *"token"*. Il parametro booleano *RequireHttps* consente di disabilitare la verifica

del certificato HTTPS, che è abilitata di default. Durante la fase di sviluppo, questo parametro rimarrà disabilitato, poiché non si dispone di un certificato firmato valido.

Infine, la sezione dedicata alla cifratura del token di archiviazione consente di impostare una password e un valore di salting per cifrare il token di accesso all'interno del database interno di Unity. La cifratura viene effettuata tramite la libreria *ZPlayerPrefs*, un'interfaccia che permette di avere i dati nel database *PlayerPrefs* cifrati, mantenendo l'uso della libreria identico a quello nativo.

La classe *AuthenticationController* gestisce l'autenticazione all'interno del progetto. Questa classe si occupa di verificare la validità dei token e di aggiornarli se necessario, oltre a svolgere le operazioni richieste per l'autenticazione dell'utente.

Il processo di login inizia con la verifica dell'esistenza di token di refresh salvati nel database. Se esiste un token di refresh, viene effettuata una richiesta per l'aggiornamento. In caso di successo, il nuovo token viene salvato; in caso contrario, e in assenza di token di refresh, viene eseguita l'operazione di login.

Il login avviene tramite il browser web. Quando bisogna autenticarsi il client apre una nuova finestra sul browser avviando una sessione nella quale l'utente effettua il login su Keycloak. Dopo aver inserito correttamente le credenziali, l'utente viene reindirizzato a una pagina HTML di conferma resa disponibile da un server web avviato dal client Unity. Tale server viene arrestato non appena viene stabilita una connessione. La porta su cui il server è esposto viene selezionata casualmente tra quelle disponibili, al fine di evitare conflitti con altri processi.

Dopo l'autenticazione, il browser comunica i dati di accesso al client, che li salva nei *PlayerPrefs*, cifrando i dati sensibili. La cifratura dei token all'interno dei *PlayerPrefs* è fondamentale, poiché i token sono dati che non devono essere leggibili da entità estranee e i *PlayerPrefs*, essendo non cifrati, possono essere facilmente accessibili.

Una volta completata l'autenticazione, viene avviata una *Croutine* che attende la scadenza del token di accesso prima di inviare una richiesta per il suo rinnovo.

Il processo di logout funziona in maniera simile al login, ma cancella tutti i dati di accesso e scollega il client da Keycloak.

La procedura di login è riassunta nella seguente immagine 3.14:

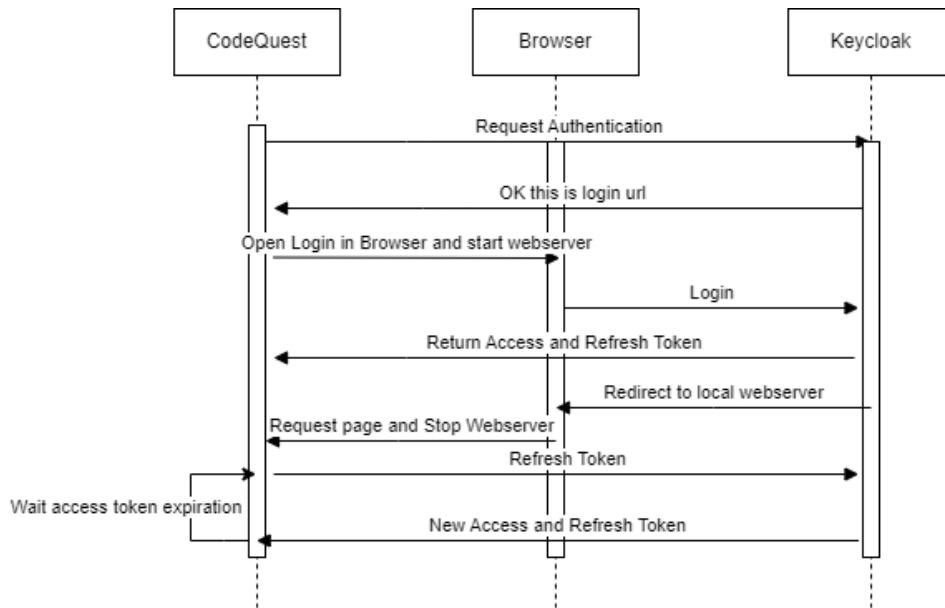


Figura 3.14: Schema di autenticazione client Unity

In conclusione, questo sistema di autenticazione permette di condividere il profilo dell’utente tra vari servizi, semplificando l’accesso tramite un unico set di credenziali su una piattaforma comune. Inoltre, il sistema è facilmente configurabile, in quanto è possibile creare più configurazioni direttamente dall’editor, grazie all’uso degli *ScriptableObject*.

3.8 Conclusioni

In conclusione, questo capitolo ha delineato in dettaglio ciascun elemento dell’infrastruttura che sostiene il servizio web di Code Quest. È stata analizzata la struttura del database, che offre la possibilità di archiviare i dati relativi agli utenti e alle partite. Inoltre è stato illustrato il funzionamento del backend, supportato da un server Spring che fornisce le API, e del frontend web realizzato in React.

In aggiunta, è stato presentato il sistema di autenticazione basato su Keycloak e OIDC, un elemento importante che garantisce l’integrità dell’infrastruttura, rendendola sicura e facile da gestire.

L’adozione di Keycloak ha semplificato la configurazione dei vari servizi, grazie all’impiego del protocollo OpenID.

Nei capitoli successivi, ci si concentrerà sulla tecnologia e le meccaniche che stanno dietro alle nuove funzionalità di Code Quest.

Capitolo 4

SmartGrid e algoritmo A*

Come precedentemente menzionato nel capitolo che descrive il contesto di Code Quest, si è deciso di modificare la meccanica di movimento, introducendo una griglia come base per le interazioni.

Tale decisione è stata motivata da vari fattori: una maggiore flessibilità nella gestione degli elementi interattivi, un ampliamento delle possibilità algoritmiche e una semplificazione dell'ambiente circostante. Un'altra considerazione fondamentale è legata al passaggio da un sistema non deterministico a un ambiente deterministico.

Con l'approccio di movimento basato sulla fisica, il sistema era non deterministico, e pertanto, a causa della natura imprevedibile delle simulazioni fisiche, operazioni come il movimento del robot non portavano sempre allo stesso risultato.

Grazie all'implementazione della griglia, ora la gestione del movimento è semplificata, e con sole tre funzioni si hanno gli strumenti per avere un controllo completo per potersi muoversi nell'ambiente di gioco.

Nelle sezioni successive, verrà analizzato in dettaglio questo componente, insieme alle meccaniche ad esso collegate. Inoltre, verrà effettuata l'analisi dell'algoritmo A* che permette di trovare percorsi sulla griglia, funzionalità utilizzata da elementi che hanno la capacità di muoversi autonomamente all'interno della griglia.

4.1 SmartGrid

Il componente chiave per il nuovo sistema di movimento in Code Quest è denominato *SmartGrid*. Esso permette di creare una griglia interattiva che analizza l'ambiente circostante in tempo reale, bloccando automaticamente le celle che contengono elementi dotati di un componente di collisione, come per esempio un muro o il robot stesso.

Le SmartCells rappresentano l'elemento portante della SmartGrid. Infatti, la SmartGrid è sostanzialmente una matrice di SmartCell. Ogni SmartCell è un oggetto che non eredita il componente MonoBehaviour, ma è un'entità standard. Questo permette di gestire la classe come un oggetto tradizionale, liberandola dalle restrizioni e dagli obblighi imposti dalle classi che ereditano MonoBehaviour, come l'esigenza di assegnare la classe a un oggetto presente nella scena. Questo aspetto è fondamentale per mantenere alte performance.

Il compito principale delle SmartCells, definito nel metodo *UpdateOccupationStatus*, consiste nel verificare se esistono elementi che collidono con il *BoxCast* generato dalla cella stessa, la quale ha dimensioni pari al lato di ogni cella, specificato dall'utente. Se viene rilevata una collisione, la cella controlla se l'oggetto è di tipo *IGridInteractiveItem*, un'interfaccia utilizzata dagli elementi che si muovono attivamente sulla griglia. Attraverso questa interfaccia, la griglia assegna all'oggetto la posizione sulla griglia in termini di indici e indica la griglia su cui l'oggetto si trova.

Se l'oggetto è interattivo, la cella lo posiziona al suo centro e salva all'interno del campo *_interactiveItem* l'istanza dell'oggetto occupante. In questo modo, è possibile recuperare l'oggetto occupante semplicemente richiamando la cella. Che si tratti di un *IGridInteractiveItem* o di un semplice oggetto con un componente Collider, la cella si auto-blocca, impostando la proprietà *IsOccupied* a true. Quando non vengono più rilevate collisioni, la cella si libera e resetta i campi che contengono l'istanza dell'oggetto occupante.

Riguardo alla classe SmartGrid, essa è un componente che deve essere instanziato nella scena, ereditando pertanto MonoBehaviour. Questa classe è caratterizzata da una matrice di SmartCells, la quale viene aggiornata ad ogni frame tramite il metodo *UpdateOccupationStatus*. La griglia viene creata attraverso il metodo *Create*, che genera un reticolo con un numero di celle specificato dai parametri *_width* e *_height* e dimensioni delle celle specificate nel parametro *_edge*.

Il metodo *ShowGrid* permette di visualizzare la griglia, che può essere generata anche in modalità editor, fornendo quindi un comportamento statico. Se

necessario, lo stato può essere aggiornato manualmente utilizzando il pulsante Update situato nella finestra Inspector del componente. Questa funzionalità può rivelarsi particolarmente utile in fase di sviluppo, per osservare e regolare l'interazione degli oggetti con la griglia. Tuttavia, è importante notare che, quando l'oggetto viene creato, la griglia verrà ricreata automaticamente al primo frame, anche se era stata precedentemente generata in modalità editor.

La griglia viene visualizzata in due modi: attraverso rettangoli generati dalla classe Gizmos di Unity, utile per scopi di debug, e mediante una shader, uno script che contiene calcoli matematici e algoritmi per determinare il colore di ogni pixel renderizzato, basandosi sull'input di illuminazione e sulla configurazione del Materiale [17]. Questo script viene eseguito dalla GPU, esternamente alla pipeline di rendering principale [18]. Tramite questo elemento si riesce ottenere ottime prestazioni anche con griglie di grandi dimensioni in fase di rendering.

La SmartGrid fornisce anche alcuni metodi di utilità per verificare se una cella in una specifica posizione è occupata, ottenere la cella più vicina a un punto dato o determinare l'orientamento di un oggetto sulla griglia. Quest'ultimo metodo, *GetDirection*, capisce, in base all'angolo dell'oggetto sull'asse y, se questo sta puntando verso il lato superiore, lato destro, lato sinistro o lato inferiore della griglia.

In conclusione SmartGrid è un componente molto versatile e applicabile in numerosi scenari di game development, poiché è possibile inserirla in una qualsiasi scena 3D e sarà in grado di adattarsi in base al ambiente che la circonda come si può vedere nella Figura 4.1.

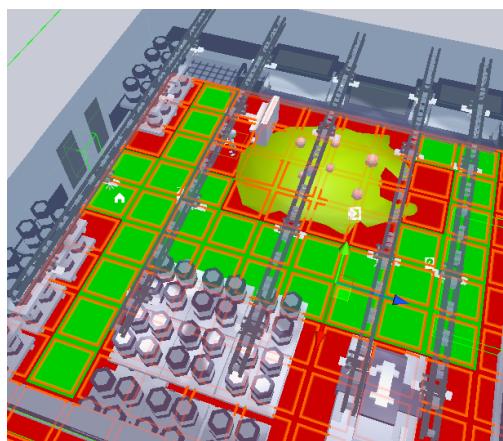


Figura 4.1: SmartGrid che si adatta all'ambiente circostante

4.2 Pathfinder e algoritmo A*

La transizione verso un'implementazione basata su griglia nel nostro gioco ha comportato modifiche sostanziali nell'approccio alla programmazione dell'intelligenza artificiale dei personaggi capaci di muoversi all'interno del mondo virtuale. Fino a quel momento, è stato utilizzato il sistema Navmesh di Unity per gestire l'IA. Questo strumento è particolarmente utile per calcolare percorsi in scenari complessi, consentendo agli elementi di individuare e raggiungere un punto specifico nell'ambiente di gioco.

Tuttavia, in un mondo basato su griglia, il movimento degli elementi non può più essere libero, ma deve seguire le linee della griglia stessa. Si potrebbe pensare di utilizzare un sistema di pathfinding basato su Navmesh anche in questo contesto, ma tale soluzione non risponde a pieno alle esigenze specifiche di un ambiente di questo tipo. Infatti, l'uso di Navmesh potrebbe comportare la necessità di istanziare ulteriori waypoint sulla griglia, con conseguente impatto sulle prestazioni. Inoltre, ci sarebbe il rischio che il movimento degli elementi non sia perfettamente allineato con la griglia, portando a scenari indesiderati in cui gli elementi "tagliano" attraverso le celle.

Ma allora, come si può calcolare un percorso all'interno di una griglia, evitando gli ostacoli e utilizzando esclusivamente le celle libere? La risposta è l'implementazione di un algoritmo noto come *A* Search*, progettato specificamente per risolvere problemi di pathfinding all'interno di un grafo.

4.2.1 A* Search

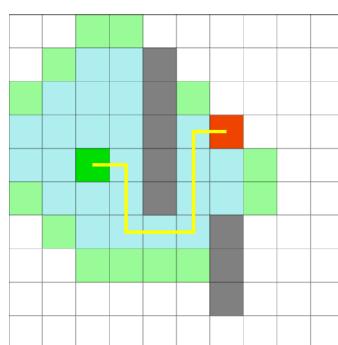


Figura 4.2: Esempio di percorso calcolato dall'algoritmo su una griglia con degli ostacoli

L'algoritmo A* è un metodo di ricerca con molteplici applicazioni, tra cui la ricerca del percorso più breve in un grafo. Esso può essere particolarmente utile per determinare la traiettoria più efficace in un videogioco all'interno di una griglia.

Nel nostro contesto immaginiamo la griglia come un grafo in cui ogni cella è un nodo e ogni collegamento tra celle adiacenti è un arco. L'algoritmo A* calcola il costo minimo $f(n)$ per raggiungere ogni nodo, dove

$$f(n) = g(n) + h(n)$$

Il costo effettivo $g(n)$ rappresenta per raggiungere quel nodo, mentre $h(n)$ è un'euristica, o una stima, del costo per raggiungere la destinazione finale [19].

L'euristica è fondamentale per l'efficacia dell'algoritmo A* poiché può velocizzare notevolmente il processo di ricerca riducendo il numero di nodi che l'algoritmo deve esplorare. Un'euristica comune è la distanza di Manhattan, che è particolarmente utile quando si possono fare solo movimenti in quattro direzioni (su, giù, destra, sinistra) [19].

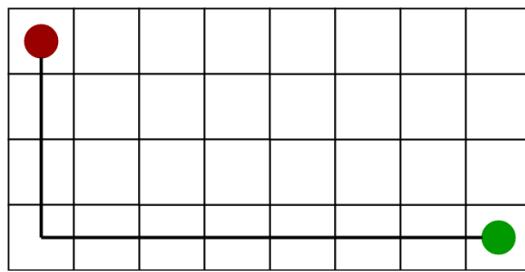


Figura 4.3: Rappresentazione della distanza di Manhattan tra due punti

La distanza di Manhattan tra due punti è la somma del numero di celle che si deve muovere orizzontalmente e verticalmente per passare da una cella all'altra [19], il nome è ispirato al quartiere di New York poiché calcola la distanza all'interno una rete di strade cittadine, infatti è anche chiamata distanza dell'isolato urbano [19].

La distanza Manhattan può essere calcolata con la seguente formula:

$$h = |\text{current_cell.x} - \text{goal.x}| + |\text{current_cell.y} - \text{goal.y}|$$

Ecco un esempio di pseudocodice per l'algoritmo A*:

```
1 function A*(start, end):
2     openList = {}
3     closedList = {}
4     openList.append(start)
5     while openList is not empty:
6         current = node in openList with the lowest F value
7         if current is equal to end:
8             return rebuild_path(current)
9         openList.remove(current)
10        closedList.append(current)
11        for node in neighbors(current):
12            if node in closedList:
13                continue to next node
14            attempt_G = current.G + distance_between(current, node)
15            if node not in openList or attempt_G < node.G:
16                node.previous = current
17                node.G = attempt_G
18                node.H = distance_manhattan(node, end)
19                node.F = node.G + node.H
20                if node not in openList:
21                    openList.append(node)
22    return null
```

Listing 4.1: A*

```
1 function rebuild_path(node):
2     # Rebuilds the path starting from the
3     #final node and going back to the starting node
4     path = []
5     current = node
6     while current.previous exists:
7         path.prepend(current)
8         current = current.previous
9     return path
```

Listing 4.2: rebuildPath

La complessità computazionale dell'algoritmo A* nel caso peggiore è $O(b^d)$, dove b è il fattore di ramificazione (il numero medio di successori per nodo) e d è la profondità della soluzione ottima (il numero minimo di passaggi necessari per raggiungere l'obiettivo a partire dal nodo iniziale) [19].

In generale, un fattore di ramificazione inferiore può accelerare l'algoritmo. La complessità spaziale dell'algoritmo A* può essere notevole, soprattutto in ambienti di calcolo di grandi dimensioni. Per questo motivo, sono state sviluppate varie versioni dell'algoritmo A* che mirano a ridurre la quantità di memoria necessaria [19].

4.2.2 A* e SmartGrid

Tornando al contesto della SmartGrid, al suo interno è contenuto un oggetto di tipo *PathFinder*, che implementa una versione personalizzata dell'algoritmo A*. Questo algoritmo è stato adattato per lavorare con la griglia di gioco.

L'interazione tra l'algoritmo A* e le celle della griglia è gestita attraverso l'interfaccia *IAStarGridCell*. Questa interfaccia definisce le proprietà che memorizzano i pesi associati a ciascuna cella, necessari per il calcolo del percorso, e il collegamento al nodo precedente. Se una cella viene selezionata come parte del percorso ottimo, il nodo che la precede viene memorizzato in questa proprietà. Questo consente di costruire il percorso ottimo utilizzando una struttura ricorsiva, che viene poi restituita come una lista di celle.

Nella classe *SmartGrid* è definito il metodo *FindPath*. Questo metodo, dati in input due oggetti *IAstarGridCell*, calcola il percorso migliore per passare da una cella all'altra. Se le due celle non sono connesse, il metodo restituirà una lista vuota, indicando che non è possibile raggiungere la cella di destinazione partendo da quella di origine.

In conclusione, l'algoritmo A* riveste un ruolo importante in Code Quest. Esso può essere utilizzato per una varietà di scopi, dall'implementazione dell'intelligenza artificiale per i personaggi virtuali, come alieni e NPC, all'implementazione di bot che giocano al posto dei giocatori, guidando le loro decisioni e strategie. Nella Figura 4.4 viene mostrato un esempio di percorso calcolato sulla SmartGrid.

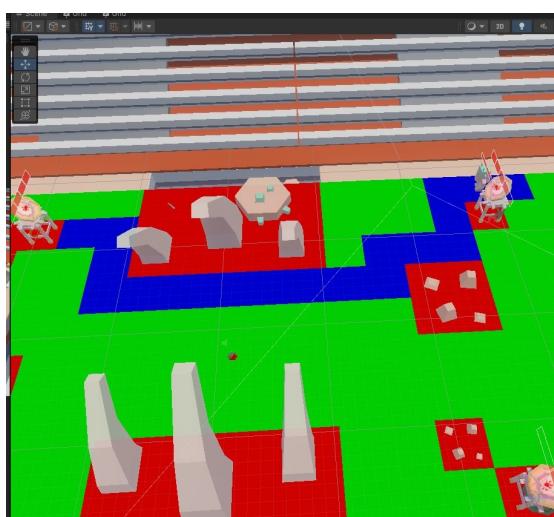


Figura 4.4: Esempio di percorso: In blu viene segnato il tracciato calcolato dall'alieno per raggiungere la torre

Capitolo 5

Gestione del multiplayer online

Lo sviluppo di un gioco multiplayer rappresenta una sfida complessa che richiede la gestione di molteplici variabili rispetto a un gioco a giocatore singolo. Fattori come l'interazione in rete, la gestione delle connessioni e delle disconnessioni, nonché la sincronizzazione degli elementi sono solo alcuni degli aspetti da gestire.

Oggi, software come Unity e Unreal Engine offrono librerie native che consentono di gestire in modo completo e integrato tutti gli aspetti del networking. Tuttavia, l'uso di queste librerie può risultare spesso complicato e richiedere un notevole impegno in termini di tempo e risorse.

Negli ultimi anni, sono state sviluppate diverse librerie che offrono API ad alto livello per semplificare la realizzazione di applicazioni multigiocatore. Tra queste rientra Photon, una libreria ampiamente utilizzata in numerosi titoli.

In questo capitolo verrà analizzato come è stata gestita l'applicazione client, sviluppata in Unity con l'ausilio della libreria Photon, e come essa si interfaccia con il servizio REST precedentemente discusso.

5.1 Architetture per lo sviluppo di videogiochi multiplayer

Lo sviluppo di giochi multigiocatore è caratterizzato dalla presenza di diversi tipi di architetture distribuite. La scalabilità, il tempo di risposta e la riduzione dei costi sono aspetti che devono essere presi in considerazione quando si progetta un videogioco multigiocatore [20]. Le architetture sono di 3 tipi: Client-Server, multi-server e Peer-to-Peer [20], in figura sono rappresentate le tre tipologie.

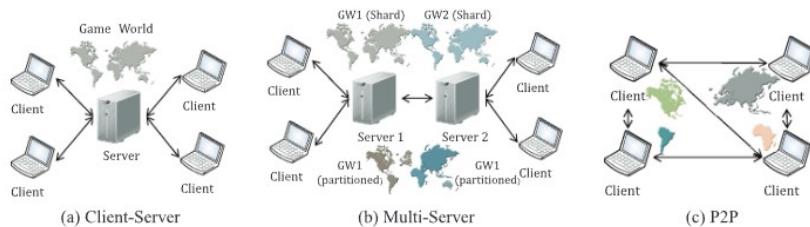


Figura 5.1: Tipi di architettura: (a) client server, (b) multi-server, (c) peer to peer

L'architettura client server usa il server per immagazzinare gli elementi che possono essere modificati, avatar e molto altro. I client si connettono al server per mantenere lo stato del mondo di gioco aggiornato, risolvendo eventuali conflitti. Lo svantaggio di questo tipo di architettura è il numero limitato di utenti che può gestire, problematica comunemente affrontata aggiungendo più server per migliorare la scalabilità [20].

L'architettura multi-server divide il mondo in istanze o regioni, le quali sono servite da server separati. Questo tipo di distribuzione dei server consente di avere una migliore tolleranza ai guasti rispetto a un sistema a server singolo. I server possono essere divisi in istanze e i giocatori possono interagire solo con quelli connessi alla stessa istanza. Questo tipo di architettura gestisce la logica del gioco lato server e possiede l'istanza del mondo di gioco direttamente sul server [20].

L'ultimo tipo di architettura è quello peer to peer ossia che ogni client interagisce con gli altri senza passare direttamente da un server centrale. Infatti, ogni client si comporta sia come client che come server. Ciò significa che ogni client deve tenere la copia dello stato della partita e aggiornare gli altri nodi. Questo tipo di architettura offre un'alta scalabilità poiché distribuisce il carico su tutti i nodi connessi. Quando si connette un nuovo utente si connette si aggiunge una nuova risorsa al sistema, migliorando la scalabilità [20].

Esistono anche soluzioni ibride che combinano questa tipologia di architetture nello specifico Photon, tecnologia utilizzata per lo sviluppo di Code Quest che

verrà discussa nelle prossime sezioni. Esso rientra in questa categoria poiché ha un server, ma l'interazione tra i vari client può avvenire sia via server che in peer to peer. Inoltre, Photon tramite il servizio cloud possiede server sparsi in tutto il mondo garantendo la scalabilità e la disponibilità in più aree del globo.

In conclusione, le architetture per i videogiochi multiplayer sono un elemento fondamentale da considerare in fase di progettazione. La scelta deve essere fatta in modo da soddisfare i giocatori ed essere resiliente ad eventuali problemi di scalabilità e guasti.

5.2 Photon

Photon è una popolare libreria di networking utilizzata nello sviluppo di giochi multiplayer. È prodotta da Exit Games e offre una serie di funzionalità che semplificano l'implementazione di vari aspetti del multiplayer, come la connettività, la sincronizzazione dello stato del gioco, la gestione delle lobby, e molto altro [21].

Photon può essere utilizzato in vari ambienti di sviluppo di giochi, compresi Unity, Unreal Engine. Le sue API ad alto livello offrono un metodo agevole per implementare funzionalità multiplayer, consentendo agli sviluppatori di concentrarsi maggiormente sulla logica e sul gameplay del gioco piuttosto che sulla gestione di dettagli di rete più tecnici [21].

Uno dei maggiori vantaggi di Photon è la sua capacità di scalare per supportare giochi di tutte le dimensioni, dalle piccole applicazioni indie ai titoli AAA. Offre servizi sia in cloud che self-hosted, con un alto grado di personalizzazione per adattarsi alle esigenze specifiche di ogni progetto. Il suo set di strumenti è progettato per garantire una latenza minima e un'esperienza di gioco fluida, anche quando i giocatori sono distribuiti in diverse parti del mondo [21]. Photon ha inoltre il vantaggio di essere supportato da una comunità attiva e da una documentazione dettagliata.

5.3 Gestione connessione alle partite

Photon offre diverse soluzioni per lo sviluppo di applicazioni, e nel caso in esame, è stata utilizzata la libreria PUN (Photon Unity Network). Questa tecnologia è progettata specificamente per funzionare con Unity, e offre un insieme di callback e componenti che facilitano la sincronizzazione degli oggetti.

Un vantaggio significativo dell'utilizzo di una libreria come PUN è la sua integrazione completa con l'ambiente di sviluppo di Unity. Photon, per esempio,

consente una sincronizzazione completa del componente Transform, che rappresenta la matrice di trasformazione di un oggetto, gli oggetti fisici, e molto altro.

Come accennato nella presentazione di Photon, la libreria offre servizi cloud. Ciò significa che possiamo utilizzare server gestiti da Exit Games per lo sviluppo, in base ai limiti del piano gratuito. Questo permette un alto grado di scalabilità e riduce il lavoro necessario per la configurazione.

Prima di analizzare la gestione della connessione delle partite all'interno di Code Quest, è utile definire alcuni concetti fondamentali di Photon:

- **Master Server e Game Server:** Il master server gestisce le partite in corso sui vari game server e fornisce un indirizzo del server di gioco quando ci si unisce o si crea una room. Il client tramite Photon passerà automaticamente al corrispettivo server di gioco [22].
- **Rooms:** Photon funziona su un sistema basato su stanze, o "Rooms", che risiedono sul server. Queste stanze virtuali permettono di ricevere e inviare informazioni ai membri della stessa stanza, ma non consentono interazioni con l'esterno. Photon fornisce un sistema di matchmaking casuale che permette di unirsi a una stanza a caso, e se non è possibile accedere a nessuna stanza, ne viene creata una nuova [23].
- **Lobby:** Una Lobby è una lista di stanze che risiede sul Master Server. Non consente interazioni dirette tra i giocatori, ma serve come luogo di raccolta per le stanze di gioco [23].
- **Master Client:** Il Master Client è un client che può essere utilizzato per prendere decisioni che influenzano tutti gli altri client. In generale, è il client che gestisce l'intera partita [23].

La comprensione di questi concetti è fondamentale per gestire efficacemente le connessioni e le interazioni in un gioco multiplayer realizzato con Photon.

La gestione delle connessioni di partita è stata implementata attraverso un'interfaccia chiamata *INetworkManager*, che include vari metodi comprese alcune callback utilizzate da Photon. Questa interfaccia viene implementata da una classe chiamata *NetworkManager*, la quale si interfaccia con tutti gli altri componenti del gioco. *NetworkManager* utilizza il pattern Proxy, permettendo così implementazioni differenti dei suoi vari metodi.

Per realizzare un implementazione specifica, bisogna creare un componente aggiuntivo che implementa l'interfaccia *INetworkManager* che andrà collocato nello stesso GameObject dove risiede il *NetworkManager*. In automatico, il Net-

`NetworkManager` assegnerà all'oggetto interno di tipo `INetworkManager`. Se il `GameObject` che contiene l'implementazione non viene trovato, il gioco restituirà un errore e interromperà l'esecuzione. Il componente che implementa l'interfaccia deve estendere la classe `BasicNetworkManager` che estende a sua volta `MonoBehaviorPunCallbacks`. Quest'ultima include le definizioni dei metodi di Photon e le rispettive implementazioni di base. La classe `BasicNetworkManager` possiede le implementazioni comuni a tutti i `NetworkManager`.

Nel caso in esame ci sono due implementazioni del `NetworkManager`: `CompetitiveNetworkManager` e `CooperativeNetworkManager`, in Figura 5.2 è mostrato il diagramma delle classi di questo componente. Questi vengono utilizzati per gestire rispettivamente le partite competitive e cooperative. Il `GameObject NetworkManager` non viene distrutto quando si cambia scena. Al contrario, viene mantenuto, permettendo così anche a oggetti di altre scene di interagire con esso.

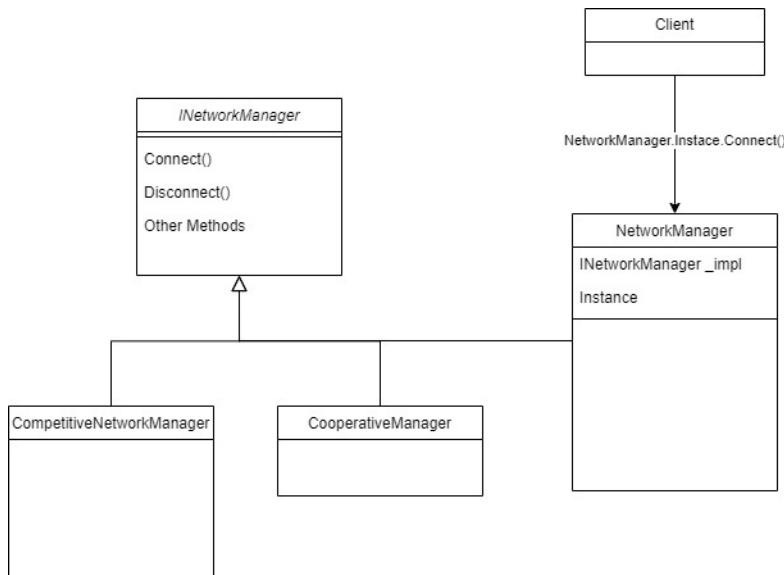


Figura 5.2: Diagramma delle classi nel `NetworkManager`

In conclusione, l'oggetto di gestione della connessione è fondamentale per il gioco, poiché consente la connessione ai server di gioco e di gestire tutti gli eventi di connessione, disconnessione e cambi del `MasterClient`.

5.4 Gestione della sincronizzazione

In una partita multiplayer, la sincronizzazione di vari elementi, sia essi oggetti visibili in scena, variabili di stato o valori di timer, è di vitale importanza. Infatti, questi elementi devono essere coordinati tra i vari partecipanti per garantire

un'esperienza di gioco fluida e coerente [22]. Photon Unity Networking (PUN) mette a disposizione diversi strumenti per facilitare questa operazione. Tra questi, i componenti PhotonTransformView e PhotonRigidbodyView permettono di sincronizzare la posizione degli oggetti e la fisica dei rigidbody [22].

Alla base del sistema di sincronizzazione c'è il componente PhotonView. Implementando una specifica versione del pattern Observer adatta al networking, PhotonView consente di sincronizzare gli elementi tra i vari client [22]. Tutti gli oggetti creati durante una partita o che necessitano di sincronizzare la loro posizione o altri elementi devono quindi avere questo componente incorporato nel loro GameObject [22].

Ma come si procede se si desidera sincronizzare una variabile o un elemento definito in uno script? La risposta risiede nelle RPC (Remote Procedure Call). Le RPC sono metodi che possono essere invocati da un client ed eseguiti su un determinato target, grazie al sistema di messaggi di Unity [22]. Per poter eseguire una RPC, un metodo deve essere contrassegnato con l'attributo [PunRPC] e chiamato attraverso un oggetto di tipo PhotonView, utilizzando la seguente sintassi: '*photonView.RPC("nome metodo", RpcTarget.All, eventuali parametri)*' [24].

È importante notare che i parametri devono essere di tipo semplice, con alcune eccezioni specificate nella documentazione di Photon. In caso si desideri passare un oggetto personalizzato, è possibile convertirlo in JSON utilizzando la classe JsonConvert (consigliata) o JsonUtility, trasmetterlo come stringa e successivamente deserializzarlo.

Le RPC possono essere inviate a vari target, come riassunto di seguito:

- `RpcTarget.All`: Invia l'RPC a tutti i client nella stessa stanza, inclusi gli altri client e il client chiamante [24]
- `RpcTarget.Others`: Invia l'RPC a tutti gli altri client nella stessa stanza, escludendo il client chiamante [24]
- `RpcTarget.MasterClient`: Invia l'RPC solo al Master Client nella stanza [24].
- `RpcTarget.AllBuffered`: Simile a All, ma l'RPC viene memorizzata nel server. Quindi, se un nuovo client si unisce alla stanza dopo l'invio dell'RPC, riceverà comunque tale RPC [24]
- `RpcTarget.OthersBuffered`: Simile a Others, ma con l'aggiunta che l'RPC viene memorizzata nel server [24]
- `RpcTarget.AllViaServer`: Invia l'RPC a tutti i client nella stessa stanza tramite il server [24].

- `RpcTarget.AllBufferedViaServer`: Combina le funzionalità di `AllViaServer` e `AllBuffered` [24].

Ecco un esempio di come impostare un metodo che supporta una chiamata RPC e la relativa chiamata:

```
1 using UnityEngine;
2 using Photon.Pun;
3 public class Chat:MonoBehavior{
4     private PhotonView photonView;
5
6     /*Other code...*/
7
8     [PunRPC]
9     private void Message(string a, string b)
10    {
11        Debug.Log(string.Format("ChatMessage {0} {1}", a, b));
12    }
13    public void SendMessage()
14    {
15        photonView.RPC("Message", RpcTarget.All, "Bob", "Hello RPC");
16    }
17}
18
```

Listing 5.1: SendRPC.cs

Nell'esempio tutti i client scriveranno il messaggio con le informazioni ricevute dalla chiamata RPC nella console. Un metodo alternativo per sincronizzare gli elementi è attraverso l'uso di *RaiseEvent* [24], ma è un'opzione a basso livello, basata su codici scritti in byte e meno intuitiva rispetto alle chiamate RPC [24].

In conclusione, la sincronizzazione è un elemento cruciale nello sviluppo di giochi multiplayer. Grazie a PUN e alle RPC, gli sviluppatori possono gestire questo processo in modo semplice e ad alto livello [24].

5.5 Interazione con il servizio web

L'integrazione del servizio web riveste un ruolo fondamentale nel salvataggio dei risultati delle partite e delle relative statistiche. Nel client Unity vengono effettuate chiamate di tipo GET e POST. Al fine di semplificare e centralizzare le operazioni, è stata creata una classe singleton denominata *HttpManager*. Questa classe espone due metodi, *MakeGetRequest* e *MakePostRequest*, che accettano vari parametri, come il percorso della richiesta, le operazioni da eseguire dopo il completamento della richiesta, un eventuale token di accesso, i dati in formato JSON sotto forma di stringa, eccetera. È necessario ricordare che questi metodi devono essere invocati attraverso *StartCoroutine*, in quanto sono operazioni asincrone che richiedono del tempo e non vogliamo che bloccino il *MainThread*. Tuttavia, per utilizzare questo servizio, sono state create altre classi di servizio che eseguono operazioni specifiche, come ottenere i dati dell'utente, scaricare e impostare le immagini che vengono gestite dalla classe *UserRESTRequests* mentre il salvataggio dei risultati delle partite viene gestito dalla classe *MatchRESTRequests*. Si raccomanda di utilizzare queste classi invece di *HttpManager* per interagire con il servizio. Nel seguente esempio, viene mostrata la chiamata al metodo per salvare una partita:

```
1 /*...Other Code....*/
2     MatchRESTRequests.Instance.SaveMatch(match, _usersStats);
3 /*...Other Code....*/
```

Listing 5.2: SaveMatch.cs

Se si devono eseguire operazioni sui dati ricevuti, è necessario introdurre meccanismi di attesa e controllo dei dati poiché caricare i dati provenienti dalla richiesta http non è un'operazione istantanea, ma richiede del tempo. Una possibile soluzione è quella di creare una coroutine di attesa che attende finché il contenuto dell'oggetto rimane invariato. Nella Figura 5.3 è illustrata l'interazione delle classi con la classe *HttpManager* con le relative chiamate tramite avvio con coroutine

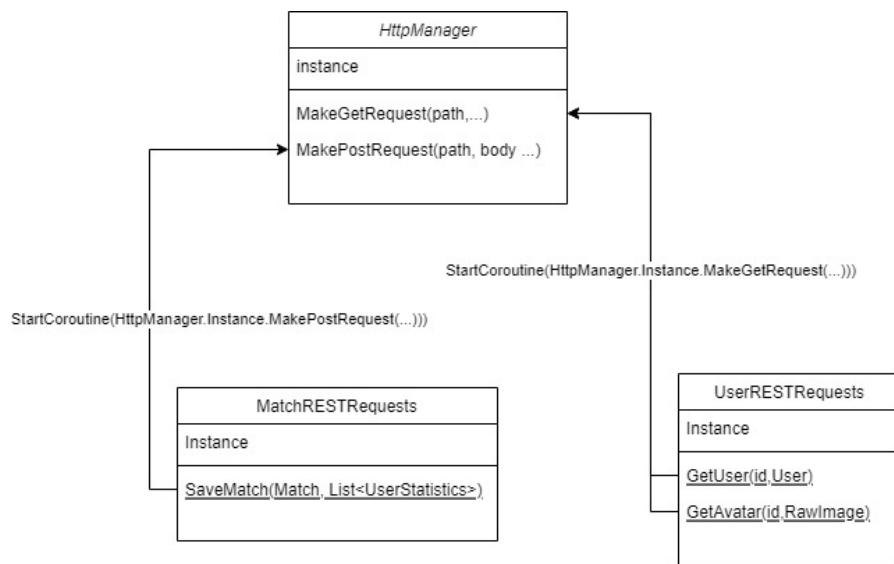


Figura 5.3: Diagramma delle classi e funzionamento HttpManager

In conclusione, l'integrazione del servizio web è un elemento fondamentale se si desidera salvare i dati di gioco e le informazioni relative alle partite. Grazie alle classi di servizio appositamente progettate per interagire con il servizio web, è possibile effettuare operazioni di salvataggio e recupero dati in modo semplificato, a patto di considerare certi dettagli di sincronizzazione.

5.6 Configurazione del client su servizi web e server di gioco differenti

Il client di gioco è stato predisposto per essere utilizzato su istanze del servizio web differenti e server di gioco diversi.

Esso, infatti, al primo avvio andrà a creare dei file JSON all'interno della cartella di persistenza del gioco, una cartella che cambia per ogni sistema operativo all'interno del quale è possibile salvare file facilmente recuperabili tramite le API di unity.

Nello specifico andrà a creare una cartella `config` all'interno della quale ci saranno 3 file JSON: `GameServer.json`, `WebserviceConfiguration.js` e `SecurityConfiguration.json`. Nel primo file, sarà presente l'`AppID`, un codice alfanumerico che può essere acquisito dalla dashboard di Photon. La principale funzione di questo codice è quella di identificare univocamente l'applicazione all'interno dei server cloud di Photon. Attraverso questo `AppID`, sarà possibile connettersi solo con giocatori che hanno lo stesso codice. Questo codice non deve essere quasi mai impostato dall'utente a meno che non si voglia isolare determinati utenti in

un ambiente isolato da altri giocatori. Inoltre, i giocatori con lo stesso AppID dovranno anche avere lo stesso servizio web.

Questo approccio consente una separazione chiara tra le diverse applicazioni all'interno dell'ambiente cloud di Photon, garantendo che le connessioni e le interazioni si verifichino solo tra gli utenti che fanno parte della stessa applicazione.

Il secondo file, invece, contiene l'URL dell'endpoint delle API del servizio web.

Il terzo file contiene tutti i parametri necessari per connettersi al servizio di autenticazione tramite OIDC. Nello specifico va inserito l'issuerURL, il clientId, l'eventuale clientSecret, lo scope e se il protocollo da usare è HTTPS, che in fase di distribuzione dovrebbe essere sempre impostato a true.

Per rendere più semplice la configurazione del client anche da parte di utenti non esperti nell'ambito delle configurazioni è stato sviluppato un tool java che consente la configurazione tramite interfaccia grafica.

Il tool si compone di 3 finestre che gestiscono in modo indipendente ogni file, nella Figura 5.4 si può vedere la struttura dell'interfaccia grafica.

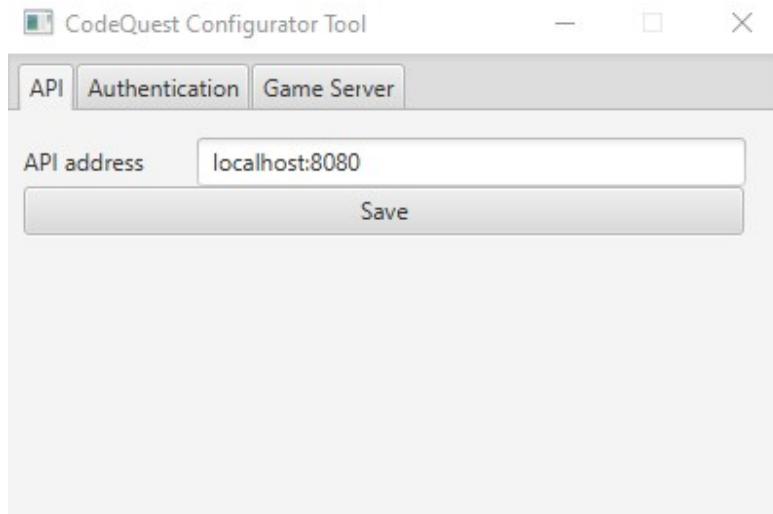


Figura 5.4: Schermata del tool

Nella schermata API è possibile configurare l'indirizzo delle API REST.

Nella schermata Authentication è possibile impostare i parametri di autenticazione, Figura 5.5

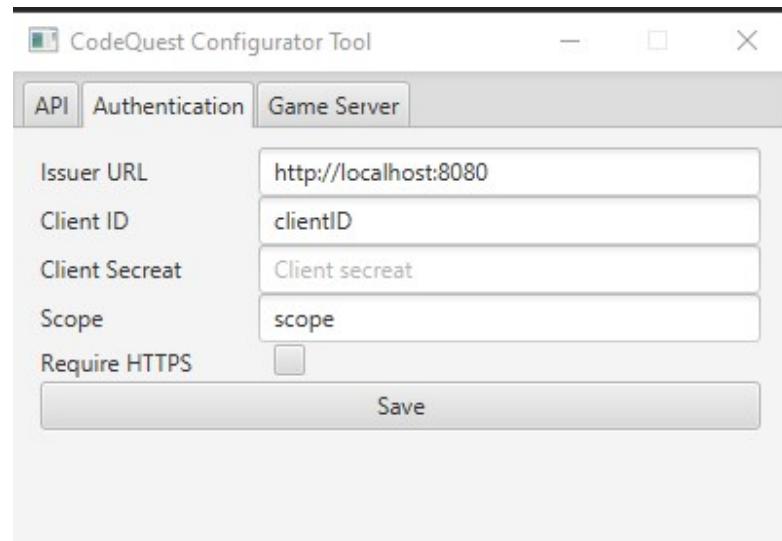


Figura 5.5: Schermata per la gestione del server di autenticazione

Mentre nella schermata Game Server si può impostare l'AppID, Figura 5.6.

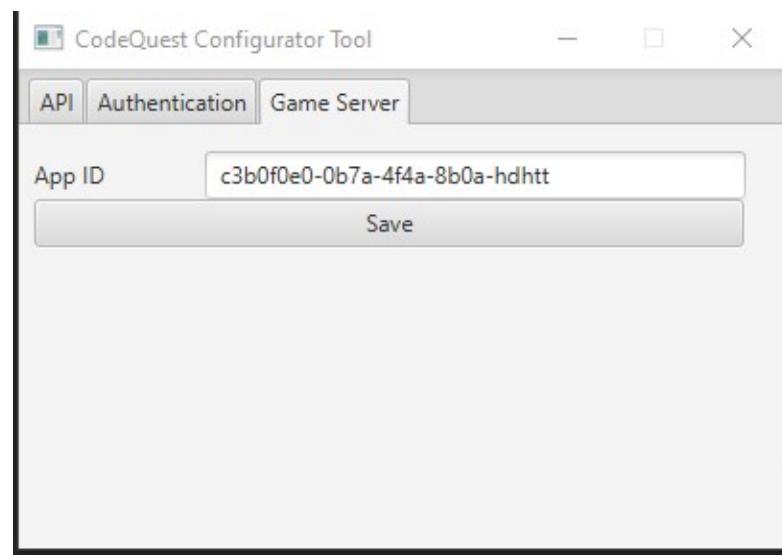


Figura 5.6: Schermata per la gestione del server di gioco

Tutti i campi obbligatori sono sottoposti a controlli in modo da evitare che vengano salvati file non validi, in caso di violazioni o errori il software comunicherà tramite pop-up a comparsa. Il tool può essere compilato per tutti i sistemi supportati da Code Quest, quindi Windows, Mac OS e Linux.

In conclusione, questo strumento offre un controllo completo sulla registrazione dei progressi degli utenti su server specifici, consentendo anche l'isolamento dei server di gioco, se necessario. Questo approccio modulare offre l'opportunità di creare un ambiente di gioco personalizzato all'interno delle scuole.

In ogni caso non appena si scarica l'eseguibile sono presenti dei parametri di default gestibili dallo sviluppatore in fase di build dell'eseguibile.

5.7 Conclusioni

In conclusione, questo capitolo ha presentato una panoramica della gestione del multiplayer online, con un focus specifico sulla tecnologia Photon e sul suo utilizzo in Unity. Sono state esplorate le componenti chiave del Photon, tra cui il suo sistema di gestione delle stanze e delle lobby e l'interfaccia INetworkManager, illustrando come queste funzionalità possono essere utilizzate per gestire l'interazione dei giocatori online. È stata analizzata l'importanza di sincronizzare tutti gli elementi di gioco principali, come le posizioni dei personaggi, lo stato del gioco e altro ancora. Questo aspetto richiede un'attenzione particolare poiché una cattiva gestione causerebbe un'esperienza di gioco poco gradevole. Nei capitoli successivi, verranno analizzate in dettaglio le modalità sviluppate, come gestiscono i vari elementi di gioco e come effettuano la sincronizzazione di quest'ultimi.

Capitolo 6

Matchmaking

In questo breve capitolo, verrà analizzato l'approccio alla gestione delle partite sia nella modalità competitiva che in quella cooperativa all'interno di Code Quest.

Per la gestione del matchmaking, ci affidiamo a Photon, che ci consente di impostare delle regole durante la fase di ricerca di una partita, permettendoci di filtrare le room non pertinenti. Ogni modalità di gioco ha una propria "Lobby", all'interno della quale sono presenti le room della stessa tipologia. Come già spiegato nel capitolo dedicato a Photon, le lobby rappresentano delle liste di room lato server; all'interno di una lobby i giocatori non possono interagire a meno che non siano connessi alla stessa room.

Nel caso di Code Quest c'è una lobby dedicata alle partite in modalità TimeAttack, una per la modalità Stop And Go e un'altra per le partite cooperative. Questa struttura assicura l'isolamento delle room con una modalità diverse da quella che si sta cercando.

La ricerca delle partite avviene casualmente, e nel caso in cui non venga trovata subito una partita, il giocatore resta in attesa che un altro giocatore si connetta alla room creata da Photon. Di seguito, viene mostrato un frammento di codice che avvia la ricerca di una partita:

```
1 public void SearchGame()
2 {
3     if (!PhotonNetwork.IsConnected)
4     {
5         Debug.LogError("Not connected to server");
6         return;
7     }
8     _SearchMatch = true;
9     Debug.Log("Searching Match....");
10    if (PhotonNetwork.InRoom || PhotonNetwork.InLobby)
11    {
12        LeaveAllAndReconnect();
13    }
14    else
15    {
16        TypedLobby lobby = new TypedLobby(
17            NetworkManager.Instance.MatchGameMode.ToString(),
18            LobbyType.Default);
19        PhotonNetwork.JoinLobby(lobby);
20    }
21 }
```

Nel codice sopra riportato, parametrizziamo la connessione alla lobby specificando il nome della lobby a cui vogliamo collegarci, una volta connessi a una lobby Photon sceglierà, se presente, la room a cui connetterci.

In sintesi, il processo di ricerca delle partite avviene tramite la separazione delle varie room all'interno di lobby dedicate alla modalità che si vuole giocare. Questa strategia garantisce un matchmaking flessibile e di facile gestione, consentendo ai giocatori di accedere rapidamente a partite coerenti con le loro preferenze di gioco.

Capitolo 7

Sistema di spawn

La gestione dei punti spawn in un gioco multigiocatore è un elemento fondamentale. Infatti, senza di essi tutti i giocatori verrebbero generati nello stesso punto senza distinguere la squadra di appartenenza e neanche lo stato attuale della mappa.

In Code Quest la gestione degli spawn è affidata al masterclient, infatti quando un utente carica la scena di gioco, esso non creerà da subito l'istanza del robot, ma seguirà una serie di passaggi.

Il primo passaggio è quello di chiamare la funzione RPC requestSpawn, l'utente masterclient aggiungerà a una coda la richiesta di spawn, questo è stato fatto per poter gestire la concorrenza delle richieste, e avvierà una coroutine che procederà all'assegnamento del punto di spawn.

I punti di spawn variano in base alla modalità, infatti nella modalità competitiva sono statici e sono posizionati agli angoli della griglia, mentre nella modalità cooperativa vengono popolate le celle che circondano il punto di spawn, centro compreso.

Nella coroutine il masterclient andrà a cercare uno spawn libero per l'utente che si trova in testa alla coda, una volta trovato registrerà lo spawn come occupato, rimuoverà l'utente dalla coda di attesa e procederà all'invio della posizione dello spawn tramite RPC inviando anche l'id dell'utente che deve eseguire la routine Spawn. Nel seguente codice viene riassunto quanto spiegato:

```
1 [PunRPC]
2 public void RequestSpawn(string playerID)
3 {
4     if (PhotonNetwork.IsMasterClient)
5     {
6         if (PhotonNetwork.InRoom)
7         {
8             Debug.Log("[Spawner] Request Spawn from " + playerID);
9             _spawnRequests.Enqueue(playerID);
10            StartCoroutine(SpawnDropper());
11        }
12    }
13 }
14 protected virtual int GetSpawnIndex()
15 {
16 //Edited in competitive mode, for teams separation
17     return 0;
18 }
19 private IEnumerator SpawnDropper()
20 {
21     CleanZombieSpawns(); //Remove requests of disconnected players
22     while (_spawnRequests.Count > 0)
23     {
24         int index = GetSpawnIndex();
25         if (index == -1)
26         {
27             Debug.LogError("Invalid CurrentIndex");
28             yield break;
29         }
30         else
31         {
32             //Try to free places
33             var players = PhotonNetwork.PlayerList;
34             //WaitDisconnection until a spawn is free
35             while (_lockedSpawn.ContainsValue(_spawnIndex + index))
36             {
37                 _spawnIndex = (_spawnIndex + 1) % _spawnPositions.Count;
38                 yield return null;
39             }
40             _lockedSpawn[_spawnIndex + index] = true;
41             yield return null;
42         }
43     }
44 }
```

```
39     }
40     Vector3 position = _spawnPositions[_spawnIndex + index]
41             % _spawnPositions.Count];
42     Quaternion rotation = _spawnRotations[_spawnIndex + index]
43             % _spawnRotations.Count];
44     if (!_lockedSpawn.ContainsKey(_spawnRequests.Peek()))
45         _lockedSpawn.Add(_spawnRequests.Peek(), _spawnIndex + index);
46     photonView.RPC(
47         "Spawn",
48         RpcTarget.All,
49         _spawnRequests.Dequeue(),
50         position, rotation,
51         _spawnIndex + index
52     );
53     _spawnIndex = (_spawnIndex + 1) % _spawnPositions.Count;
54 }
55 yield return null;
56 SynchSpawner();
57 }
58 //Free spawn if player is not in the room
59 _spawnRequests.Clear();
60 }
61 }
```

Per quanto riguarda la funzione di spawn essa procederà a configurare il robot e posizionarlo nella posizione assegnata. Nel seguente codice è riportato il metodo di spawn.

```

1 [PunRPC]
2 public void Spawn(string playerID,
3                     Vector3 spawnPosition,
4                     Quaternion spawnRotation,
5                     int index){
6     if (PhotonNetwork.LocalPlayer.UserId == playerID && !_hasSpawned)
7     {
8         GameObject robot = PhotonNetwork.Instantiate(
9                         string.Format("Prefab/{0}",
10                             _robotPrefab.name),
11                         spawnPosition,
12                         spawnRotation);
13         _myspawn = new Tuple<string, int>(playerID, index);
14         RobotController controller = robot.AddComponent<GridRobotController>();
15         MultiplayerRobotEventsManager eventsManager =
16             robot.AddComponent<MultiplayerRobotEventsManager>();
17         robot.AddComponent<SelectRobot>();
18         AddExtra(robot);
19         //Configure
20         controller.SetSpeed(_robotConfiguration.Speed);
21         eventsManager.SetCollisions(this, _robotConfiguration.MaxCollisions);
22         eventsManager.SetRespawn(this, _robotConfiguration.CanRespawn);
23         if (_robotConfiguration.CanRespawn)
24         {
25             SetSpawn(spawnPosition, spawnRotation);
26         }
27         _hasSpawned = true;
28     }
29 }
```

Quando avviene un cambio di masterclient allora le informazioni relative agli spawn verranno scritte sul nuovo client direttore. Le informazioni condivise che necessitano di essere sincronizzate, come gli spawn occupati, sono stati gestiti salvando i dati all'interno delle proprietà della stanza.

In conclusione, il sistema di spawn copre una parte fondamentale del gioco. Esso è in grado di posizionare il robot del giocatore in posizioni sempre valide senza sovrapposizioni con altri giocatori.

Capitolo 8

Gestione delle partite competitive

Nelle sezioni precedenti, si è affermato che Code Quest si compone di una modalità multiplayer con componenti competitive. Questa modalità è progettata con l'obiettivo principale di innescare una spinta motivazionale nei giocatori, spingendoli a mettere in gioco non solo le proprie abilità di programmazione e risoluzione dei problemi, ma anche quelle tattiche e gestionali. Il fulcro di questa esperienza è l'adozione di un modello di gioco ispirato al Tower Defense, un genere ampiamente noto e diffuso in contesti simili.

In questo capitolo, verrà analizzato il funzionamento delle partite, delle regole che ne regolamentano lo svolgimento e degli elementi di supporto utilizzabili dai giocatori. verrà approfondito anche il processo di smistamento dei giocatori all'interno delle squadre, esaminando il modo in cui si realizza questa fase fondamentale di preparazione, che precede la partita vera e propria.

8.1 Regole del gioco

La modalità competitiva di Code Quest trae ispirazione dai giochi Tower Defense, in cui l'obiettivo principale è difendere le proprie torri/basi e conquistare quelle avversarie. Questo stile di gameplay offre ampie possibilità di variazione, e Code Quest offre due varianti: *Time Attack* e *Stop And Go*.

L'obiettivo in entrambe le varianti è identico: conquistare le tre torri nemiche o abbattere il robot avversario. Le modalità si differenziano per l'approccio di programmazione; infatti, nel Time Attack è consentita una modifica continua del codice che controlla il robot per adattarsi alle situazioni mutevoli, mentre nel Stop And Go la fase di programmazione è vincolata alle fasi di time-out attivabili tramite il pulsante “CQ” posizionato nella parte inferiore dell'interfaccia di gioco Figura 8.1, rendendo il gameplay più simile a una partita di scacchi.



Figura 8.1: Interfaccia del gioco: la prima icona rappresenta il bottone "CQ" per richiedere il time-out di programmazione

Ogni partecipante dispone di un proprio robot da gestire, impiegandolo sia per difendere il proprio territorio che per attaccare quello avversario. In questa modalità sono messi a disposizione tre oggetti bonus per giocatore:

- 3 alieni
- 10 rocce
- 5 torrette fisse

Nella sezione successiva, sarà esaminato dettagliatamente il funzionamento di questi tre elementi. Per stimolare l'attenzione e la precisione nella scrittura del codice è stato aggiunto un limite di collisioni possibili. Infatti, nel caso in cui un giocatore tenti di muoversi in celle già occupate della griglia, sia da nemici che dagli elementi dello stesso giocatore, il robot accumulerà un punto danno. Questo stato di deterioramento è reso visibile attraverso effetti particellari attivati

nella parte posteriore del robot. L'accumulo di danno si manifesta attraverso l'emissione di fumo, seguito da un accendersi di fuoco a metà danno e, infine, il robot esploderà quando il limite massimo di danno sarà raggiunto. Questa soglia può essere configurata all'interno delle regole di gioco, tramite l'oggetto di tipo *Scriptable Object* dedicato.

Ogni partita ha una durata massima predefinita in minuti, anch'essa personalizzabile all'interno delle impostazioni di configurazione, così come è possibile regolare il tempo extra in caso di pareggio. Per ogni torre distrutta, il giocatore ottiene un punto, così come quando il robot avversario esaurisce le collisioni a sua disposizione. Le condizioni di terminazione della partita sono le seguenti:

- Scadenza del tempo: il giocatore con il punteggio più alto vince.
- Distruzione uno dei due robot.
- Abbandono dell'avversario: in questo caso, si vince 3 a 0 a tavolino.

In conclusione, l'aspetto flessibile di Code Quest risiede nella possibilità di regolare i parametri delle diverse modalità attraverso le adeguate configurazioni, consentendo la creazione di nuove varianti e l'eventuale coinvolgimento di più giocatori.

In questo gruppo di modalità, vengono messe in gioco diverse abilità da parte dei giocatori, tra cui strategia, velocità di analisi dei problemi e implementazione della relativa soluzione. Questo approccio di gioco può agevolmente portare i giocatori a sviluppare una mentalità incentrata su affrontare sfide in maniera frammentata, suddividendo un problema complesso in sotto problemi più gestibili. Questo metodo di risoluzione, basato su piccole soluzioni cumulative che alla fine convergono in una soluzione complessiva, è di fondamentale importanza nell'ambito della programmazione.

Attraverso l'esperienza di gioco offerta da Code Quest, i giocatori potranno acquisire l'abilità di affrontare problemi complessi decomponendoli in sotto problemi più piccoli e risolverli in modo sequenziale. Questo approccio è fondamentale quando si parla di programmazione e di conseguenza è importante che i giocatori di Code Quest acquisiscano questo metodo. La capacità di pensare in termini di micro-problemi e di costruire gradualmente soluzioni più ampie non solo migliora l'efficienza nell'ambito del gioco, ma si traduce anche in una capacità di risolvere problemi più ampia e trasferibile ad altri campi oltre a quello della programmazione.

Inoltre, queste modalità di gioco incoraggiano l'adattamento continuo, poiché i giocatori devono considerare diverse variabili e adottare strategie fluide in base alle situazioni in evoluzione. Ciò rispecchia la realtà della programmazione, in cui le esigenze e le sfide possono cambiare rapidamente e richiedere soluzioni flessibili.

In definitiva, le abilità sviluppate attraverso questa esperienza di gioco consentono di acquisire un approccio di pensiero analitico e sequenziale fondamentale nella risoluzione di problemi complessi e nella programmazione. Nella sezione successiva, verranno presentati gli elementi messi a disposizione del giocatore e le loro funzioni quando impiegati.

8.2 Elementi di gioco

Nella sezione precedente, sono stati menzionati gli elementi messi a disposizione dei giocatori, che possono essere utilizzati durante la partita per agevolare le fasi di attacco e difesa. Questi elementi includono sia oggetti dinamici, con comportamenti autonomi, che oggetti statici.

Esaminiamo il primo elemento: gli alieni. Gli alieni sono oggetti che, una volta creati, agiscono indipendentemente e sfuggono al controllo diretto del giocatore. Una volta posizionati sulla griglia, attraverso l'utilizzo dell'algoritmo A* precedentemente discusso, individueranno il nemico più vicino e calcoleranno un percorso per raggiungerlo. Gli alieni si muoveranno seguendo il percorso calcolato, adattandosi nel caso in cui ostacoli o avversari intermedi compaiano sulla loro strada. Se si imbattono in alleati, ricalcoleranno il percorso per evitare collisioni. Una volta giunti a destinazione, eseguono una scansione dell'area compiendo una rotazione di 360 gradi in modo sequenziale, con passaggi di 90 gradi al secondo. Questo processo consente loro di individuare il bersaglio o l'oggetto che intendono distruggere. Una volta rilevato il nemico, iniziano sparare, infliggendo danni all'obiettivo. Una volta che la salute del nemico raggiunge lo zero, scateneranno un'esplosione che colpisce sia il nemico che gli stessi alieni che sono diretti verso l'obiettivo appena distrutto. Questa scelta è stata fatta per evitare un eccessivo utilizzo degli alieni, spingendo il giocatore a concentrarsi maggiormente sulla parte di programmazione.

Il secondo elemento sono i meteoriti: oggetti statici il cui scopo è ostacolare i nemici e gli alieni nemici, che possono essere eliminati sparandoci contro.

Il terzo e ultimo elemento sono le torrette fisse. Questi elementi sono dinamici nonostante siano posizionati staticamente; infatti, sorvegliano le celle a 0, 90, 180, 270 e 360 gradi rispetto alla loro posizione. Dopo essere state posizionate,

le torrette avviano una fase di ricerca che comporta una rotazione di 90 gradi, al fine di controllare la presenza di nemici nella direzione che stanno puntando.

Quando le torrette individuano un nemico, la loro rotazione automatica viene interrotta e inizia il fuoco diretto verso il nemico. Dopo aver sparato il numero di colpi massimo, quantità che può essere regolata attraverso il prefab dell'oggetto, le torrette si autodistruggono. Per quanto riguarda l'uso di questi elementi da parte del giocatore, all'interno del linguaggio CQ è stato aggiunto un metodo chiamato "spawn". Questo metodo richiede due parametri: l'indice dell'oggetto e la posizione in cui si desidera collocarlo. Le posizioni sono numerate come segue:

- 0: di fronte
- 1: a destra
- 2: dietro
- 3: a sinistra

Qualsiasi posizione o indice non validi verranno segnalati dal compilatore CQ. L'interfaccia grafica che monitora queste risorse è situata centralmente nella parte inferiore della finestra di gioco, come mostrato nella Figura 8.2. Ogni elemento della lista è composto da 2 caselle di testo e un'immagine.

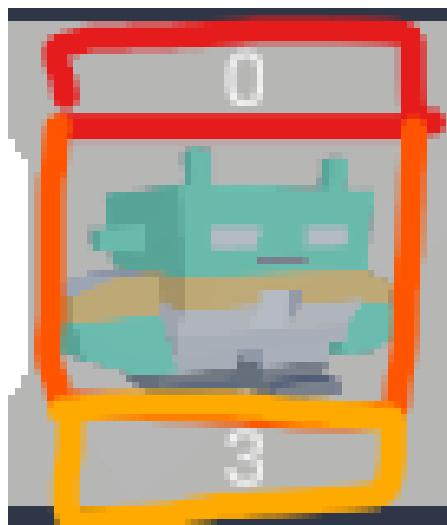


Figura 8.2: Rappresentazione di un elemento consumabile nell'interfaccia di gioco

L'immagine mostra un'anteprima grafica, la casella di testo superiore mostro l'indice dell'oggetto e quella inferiore la quantità disponibile.

In conclusione, questi elementi sono stati introdotti per conferire un maggiore dinamismo al gameplay, introducendo ulteriori variabili da considerare. L'imposizione di alcuni limiti su determinati elementi è stata fatta appositamente per evitare che la partita sia dominata dalle intelligenze artificiali degli elementi, invitando i giocatori alla programmazione del robot e delle sue mosse.

8.3 Assegnamento squadre e avvio di una partita

L'assegnazione delle squadre riveste un ruolo fondamentale all'interno di una modalità competitiva. Nel nostro caso, è stato sviluppato un sistema che supporta la formazione di squadre anche se al momento le modalità di gioco sono limitate a scontri uno contro uno. Questo sistema segue un processo articolato in tre fasi:

1. Registrazione
2. Mescolamento
3. Assegnamento

La fase di registrazione è un passaggio che coinvolge tutti i giocatori partecipanti alla partita. Appena il client completa il caricamento della scena di gioco, viene eseguita una chiamata RPC al metodo *PlayerReadyRPC*. Questo passo è fondamentale per sincronizzare tutti i client e ottenere il numero esatto di giocatori pronti all'interno della scena di gioco. Durante questa procedura, un contatore dedicato viene incrementato per tener traccia dei giocatori che sono pronti. Una volta che il numero di giocatori pronti raggiunge il numero totale di giocatori connessi nella room, viene chiamato il metodo *PlayerReady*. Questo metodo è virtuale e ciò è può essere sovrascritto dalle classi che gestiscono le diverse modalità di gioco, offrendo la flessibilità necessaria per adattare il comportamento alle esigenze specifiche di ciascuna modalità. Il seguente pezzo di codice mostra quanto spiegato:

```
1 protected virtual void PlayersReady()
2 {
3     if (!_matchStarted)
4     {
5         StartMatch();
6     }
7 }
8 [PunRPC]
9 public void PlayerReadyRPC()
10 {
11     _readyPlayers++;
12     if (_readyPlayers == PhotonNetwork.PlayerList.Length)
13     {
14         foreach (Photon.Realtime.Player player in PhotonNetwork.PlayerList)
15         {
16             //add if player missed at the start
17             if(!_players.ContainsKey(
18                 player.CustomProperties["UserID"].ToString()
19             ))
20             {
21                 _players.Add(player.CustomProperties["UserID"].ToString(), player);
22             }
23         }
24         PlayersReady();
25     }
26 }
```

Nella versione base del metodo *PlayersReady*, la sua funzione principale è verificare se la partita è pronta per essere avviata e, in caso affermativo, procedere all'avvio. Tuttavia, nella modalità competitiva, vi sono ulteriori passaggi che vengono eseguiti prima dell'avvio effettivo del match. Prima di avviare la partita, il masterclient si occuperà del processo di mescolamento delle squadre. In questa fase, i giocatori vengono distribuiti casualmente in due squadre differenti.

Una volta completato il mescolamento, viene effettuata una chiamata RPC che trasmette i dati delle squadre. Questi dati vengono poi salvati da tutti i client, incluso il masterclient. Questo passo assicura che ogni giocatore abbia accesso alle informazioni sulle squadre a cui sono stati assegnati i giocatori. Di seguito

sono riportate le funzioni che si occupano del processo di mescolamento delle squadre:

```

1  private void Shuffle<T>(List<T> list)
2  {
3      System.Random random = new System.Random();
4      int n = list.Count;
5      while (n > 1)
6      {
7          n--;
8          int k = random.Next(n + 1);
9          T value = list[k];
10         list[k] = list[n];
11         list[n] = value;
12     }
13 }
14 protected override void PlayersReady()
15 {
16     if (PhotonNetwork.IsMasterClient && !MatchStarted)
17     {
18         AssignTeam();
19     }
20 }
21 private void AssignTeam()
22 {
23     if (PhotonNetwork.IsMasterClient)
24     {
25         List<List<string>> _serializableList = new List<List<string>>();
26         // Step 1: Shuffle the player list
27         Shuffle(_players.Values.ToList());
28         // Step 2: Divide players into teams
29         List<Team> teams = new List<Team>();
30         for (int i = 0; i < _gameMode.NumberOfTeams; i++)
31         {
32             teams.Add(new Team());
33             _serializableList.Add(new List<string>());
34         }
35         // Step 3: Assign players to teams

```

```
37     int teamIndex = 0;
38     foreach (Photon.Realtime.Player player in PhotonNetwork.PlayerList)
39     {
40         Team currentTeam = teams[teamIndex];
41         currentTeam.AddMember(player);
42         // Rotate through teams
43         teamIndex = (teamIndex + 1) % _gameMode.NumberOfTeams;
44         _serializableList[teamIndex].Add
45         (player.CustomProperties["UserID"].ToString()
46     );
47 }
48 //update clients
49 byte[] teamAssignmentsBytes;
50 using (MemoryStream stream = new MemoryStream())
51 {
52     BinaryFormatter formatter = new BinaryFormatter();
53     formatter.Serialize(stream, _serializableList);
54     teamAssignmentsBytes = stream.ToArray();
55 }
56 Debug.Log("Team Assignments: " + teamAssignmentsBytes.Length);
57 photonView.RPC(
58     "AssignTeamToClients",
59     RpcTarget.All,
60     teamAssignmentsBytes,
61     _gameMode.TeamSize
62 );
63 }
64 }
```

Il seguente codice invece, mostra una versione ridotta del metodo AssignTeamToClients:

```
1 [PunRPC]
2 private void AssignTeamToClients(byte[] teams, int memberForTeam)
3 {
4     List<List<string>> teamAssignments;
5     /**Code to deserialize and copy teams*/
6     //.....
7     Debug.Log("Team assigned " + _myTeam.TeamName);
8     LoadArena();
9     CameraChooser.Instance.SetCamera(_myTeam.TeamName);
10    if (PhotonNetwork.IsMasterClient)
11    {
12        StartMatch();
13    }
14 }
```

Conclusa la fase di mescolamento e assegnamento il sistema procede al caricamento della mappa e alla configurazione delle telecamere. Inoltre, il master-client precederà ad avviare la partita facendo partire un conto alla rovescia di 5 secondi.

8.4 Conclusioni

Il capitolo appena presentato ha approfondito diversi aspetti cruciali della modalità competitiva di Code Quest. Sono state esaminate le regole del gioco e la struttura delle partite, delineando un quadro chiaro di come l'esperienza di gioco si sviluppa. Inoltre, sono stati presentati gli elementi di supporto, come gli alieni e le torrette automatiche, evidenziando il bilanciamento necessario per mantenere l'obiettivo centrale del progetto Code Quest: l'apprendimento della programmazione.

Inoltre, è stato spiegato come le meccaniche di gioco implementate siano strettamente integrate con l'apprendimento della programmazione stessa. Questo ha lo scopo di guidare i giocatori verso l'importante abilità di scomporre problemi complessi in più piccoli e gestibili, che è una competenza cruciale nel mondo della programmazione e in molti altri campi.

Infine, è stato esaminato nel dettaglio il processo di assegnazione delle squadre all'inizio di ogni partita, elemento fondamentale quando si parla in un contesto multigiocatore con componenti competitive.

Capitolo 9

Generazione procedurale dei livelli

9.1 PCG (Procedural Content Generation)



Figura 9.1: Esempio di alberi generati proceduralmente

Cosa rappresenta il termine "Procedural Content Generation" (PCG)? Nel contesto dei videogiochi e della computer grafica, la creazione di ambientazioni 3D e di altri elementi può risultare un'attività lunga e ripetitiva. Nel corso del tempo, sono stati sviluppati diversi algoritmi mirati alla generazione di contenuti, noti come Procedural Content Generation (PCG).

Questi algoritmi permettono di creare in modo automatico mappe, alberi e altri elementi, alleggerendo il carico di lavoro degli artisti [25]. Nonostante la generazione di alcune tipologie di contenuti, come ad esempio gli alberi, sia diventata una pratica comune nell'industria dei videogiochi, la maggior parte delle solu-

zioni di PCG sono progettate per adattarsi specificamente al contesto in cui operano [25].

Questi algoritmi presentano vantaggi e svantaggi: da un lato, offrono la possibilità di creare contenuti diversificati che rispettano precisi requisiti. Dall'altro, però, esiste il rischio che in determinate situazioni, l'impredicibilità degli algoritmi possa portare alla creazione di elementi non perfettamente adatti, pur rispettando i requisiti [25].

In conclusione, questi algoritmi rappresentano strumenti di grande utilità per generare livelli ed elementi quali stanze e alberi all'interno dei videogiochi. Esistono varie implementazioni algoritmiche per realizzare tali contenuti, e ciascuna di esse è particolarmente adatta a creare elementi specifici.

Nelle sezioni successive, verrà approfondito gli algoritmi utilizzati per generare le mappe all'interno della modalità cooperativa, ossia il Perlin Noise e il BSP (Binary Space Partitioning).

9.2 BSP (Binary Space Partitioning)

Il processo di generazione di mappe che coinvolge la creazione di stanze può essere affrontato attraverso varie strategie, e una di queste è l'utilizzo dell'algoritmo BSP (Binary Space Partitioning).

L'obiettivo principale di questo algoritmo è suddividere uno spazio euclideo in due insiemi convessi utilizzando dei sottospazzi come partizioni [26]. Con insiemi convessi si intende un insieme nel quale, per ogni coppia di punti, il segmento che li congiunge è interamente contenuto nell'insieme [27]

Questo tipo di problema non è limitato solamente alla generazione di livelli nei videogiochi, ma trova applicazione anche nell'ambito della computer grafica e delle simulazioni [28]. L'approccio dell'algoritmo BSP consente di operare in modo ricorsivo, suddividendo uno spazio euclideo in due insiemi convessi.

BSP si basa su un albero binario nel quale ogni nodo rappresenta un sottospazio. Le foglie rappresentano gli spazi finali in cui lo spazio iniziale, rappresentato dalla radice, è stato diviso [29]. La suddivisione dell'area avviene in maniera ricorsiva, e ogni nodo viene suddiviso indipendentemente dal fratello con cui condivide il genitore [29].

Per quanto riguarda la complessità, nel caso di Code Quest è stato utilizzato una BSP ortogonale, ovvero un BSP nel quale i tagli sono fatti solo in verticale o in orizzontale. Questo caso è stato studiato da M. Paterson, F. Yao i quali hanno dimostrato che questo tipo di BSP ha una complessità di $O(n \log^2(n))$

dato un set di n elementi [30]. Nell'immagine 9.2 è possibile vedere uno spazio partizionato e il relativo albero.

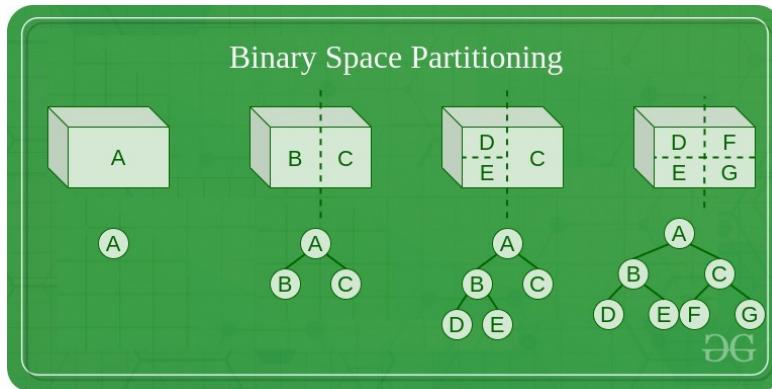


Figura 9.2: Spazio partizionato con il relativo BSPT(BSP Tree)

Un aspetto chiave dell'algoritmo è che, all'interno del progetto, vengono implementate due tipologie di operazioni di partizionamento scelte casualmente: taglio verticale e taglio orizzontale. Queste operazioni consentono di ottenere aree diverse ad ogni esecuzione dell'algoritmo. Inoltre, viene eseguito il partizionamento di una stanza solo quando la sua area supera il valore massimo consentito e il numero di stanze create è inferiore al valore massimo, impostabile come parametro. Nel caso in cui questi vincoli non vengano rispettati la ricorsione si interrompe.

Nei seguenti pezzi di codice sono riportati i metodi che implementano l'algoritmo BSP all'interno del plugin di creazione delle mappe sviluppato all'interno di Code Quest:

```

1 private Room Split(Room room)
2 {
3     if (_rooms.Count >= _maxRoomsCount)
4     {
5         return null;
6     }
7     Room output = null;
8     if (room.Width < _minWidth || room.Height < _minHeight)
9     {
10        return null;
11    }
12    if ((room.Area <= MaxArea &&

```

```
13     room.Area >= MinArea &&
14     room.Width <= _maxWidth &&
15     room.Width >= _minWidth &&
16     room.Height <= _maxLength &&
17     room.Height >= _minHeight))
18 {
19     output = room;
20     _rooms.Add(room);
21 }
22 else
23 {
24     if (Random.Range(0, 2) == 0)
25     {
26         output = SplitHorizontally(room);
27     }
28     else
29     {
30         output = SplitVertically(room);
31     }
32 }
33 return output;
34 }
35 private Room SplitHorizontally(Room room)
36 {
37     //Split it
38     int splitPoint = Random.Range(
39             room.Left + _minWidth,
40             room.Right - _minWidth);
41     Room leftChild = new Room(
42             room.Left,
43             splitPoint,
44             room.Bottom,
45             room.Top,
46             _smartGrid);
47     Room rightChild = new Room(
48             splitPoint,
49             room.Right,
50             room.Bottom,
51             room.Top,
```

```
52         _smartGrid);
53     room.LeftChild = Split(leftChild);
54     room.RightChild = Split(rightChild);
55     if (room.LeftChild != null)
56         leftChild.Parent = room;
57     if (room.RightChild != null)
58         rightChild.Parent = room;
59     //Return it
60     return room;
61 }
62 private Room SplitVertically(Room room)
63 {
64     //Split it
65     int splitPoint = Random.Range(
66             room.Bottom + _minHeight,
67             room.Top - _minHeight);
68     Room bottomChild = new Room(
69             room.Left,
70             room.Right,
71             room.Bottom,
72             splitPoint,
73             _smartGrid);
74     Room topChild = new Room(
75             room.Left,
76             room.Right,
77             splitPoint,
78             room.Top,
79             _smartGrid);
80     room.LeftChild = Split(bottomChild);
81     room.RightChild = Split(topChild);
82     if (room.LeftChild != null)
83         bottomChild.Parent = room;
84     if (room.RightChild != null)
85         topChild.Parent = room;
86     //Return it
87     return room;
88 }
```

In conclusione, l'adozione del BSP ha dimostrato la sua efficacia nel suddividere lo spazio euclideo in modo strategico, creando delle stanze che contribuiscono alla struttura complessiva del livello di gioco.

L'introduzione di vincoli e condizioni, come il partizionamento delle stanze solo quando l'area supera il limite massimo e il numero massimo di stanze, riflette la capacità di adattare l'algoritmo alle esigenze specifiche del gioco. Questo approccio consente di bilanciare la varietà e la coerenza delle mappe generate, garantendo una diversificazione strutturale senza compromettere l'integrità delle stanze e dell'esperienza di gioco nel complesso.

Nelle prossime sezioni, verrà esaminato come queste aree vengono interconnesse e come si arriva a creare il livello finale del gioco.

9.3 Perlin Noise

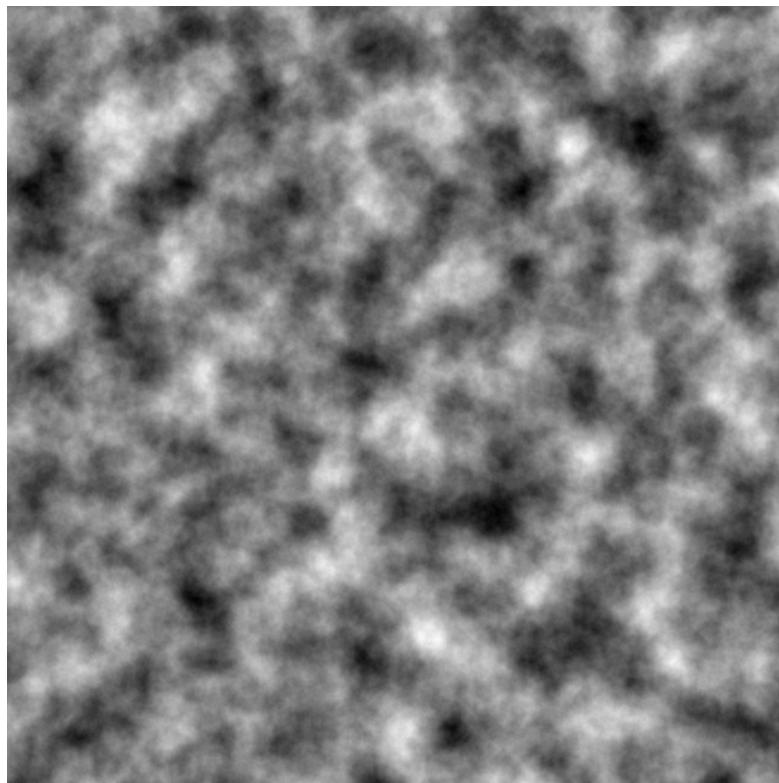


Figura 9.3: Esempio di Perlin Noise

Una tecnica molto utilizzata nella generazione procedurale è il Perlin Noise. Esso è un rumore di tipo gradiente, ossia una funzione di rumore utilizzata in computer grafica per creare texture procedurali come effetti organici, nuvole e altri.

È stato sviluppato da Ken Perlin nel 1980 ed è parecchio utilizzato in computer grafica [31]. Il rumore viene creato combinando più ottave di rumore casuale attenuato. Ogni ottava ha una frequenza e un'ampiezza diversa ed è generata prendendo una griglia di valori casuali e interpolandoli per creare una funzione liscia e continua. La funzione di rumore risultante viene quindi scalata e aggiunta all'ottava precedente, con ogni ottava successiva che ha una frequenza più alta e un'ampiezza più bassa. Il risultato finale è una funzione di rumore liscia e continua che varia dolcemente in tutte le direzioni e può essere utilizzata per creare un'ampia gamma di effetti naturalistici [31].

Nello sviluppo di videogiochi questo tipo di rumore è molto utilizzato nella generazione di livelli per definire ad esempio l'altezza dei rilievi.

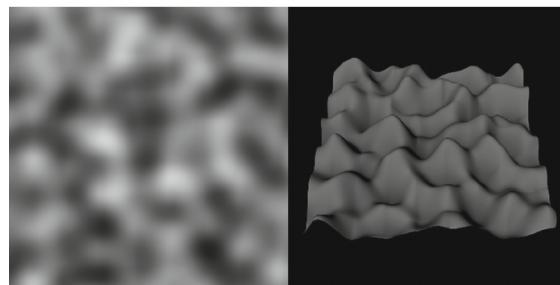


Figura 9.4: Terreno generato con il Perlin Noise

Nel caso di Code Quest questo tipo di rumore è stato utilizzato per creare l'ambiente circostante al livello di gioco, come si può vedere dalla Figura 9.5.



Figura 9.5: Esempio di livello generato con il plugin

Sono stati scelti 3 colori per rappresentare l'erba, la terra e le rocce in modo da rendere più vario il livello. L'uso del Perlin Noise all'interno di Unity è molto semplice poiché già implementato dalla struct Mathf. Il seguente pezzo di codice mostra come è stato usato il Perlin Noise all'interno della generazione del livello.

```
1 private void FillWorld(HashSet<SmartCell> corridors)
2 {
3     // Adjust the scale to control the Perlin noise pattern
4     float scale = 0.1f;
5     // Randomize the offset for variety
6     float xOffset = UnityEngine.Random.Range(0f, 1000f);
7     float yOffset = UnityEngine.Random.Range(0f, 1000f);
8     GameObject terrain = new GameObject("Terrain");
9     terrain.transform.SetParent(_world.transform);
10    for (int i = 0; i < _smartGrid.Grid.GetLength(0); i++)
11    {
12        for (int j = 0; j < _smartGrid.Grid.GetLength(1); j++)
13        {
14            bool zombieCell = true;
15            if (corridors.Contains(_smartGrid.Grid[i, j]))
16            {
17                zombieCell = false;
18            }
19            else
20            {
21                foreach (Room room in _rooms)
22                {
23                    if (room.IsInside(i, j))
24                    {
25                        zombieCell = false;
26                    }
27                }
28            }
29            if (zombieCell)
30            {
31                if (_smartGrid.Grid[i, j] != null)
32                {
33                    // Calculate Perlin noise value using scaled and offset indices
34                    float height = Mathf.PerlinNoise(
35                        (float)i * scale + xOffset,
36                        (float)j * scale + yOffset);
37                    GameObject obj = null;
38                    if (height < 0.2f)
```

```

39         obj= Instantiate(_grassPrefab,
40                         _smartGrid.Grid[i, j].LocalPosition,
41                         Quaternion.identity);
42     else if(height < 0.5f)
43         obj= Instantiate(_soilPrefab,
44                         _smartGrid.Grid[i, j].LocalPosition,
45                         Quaternion.identity);
46     else
47         obj= Instantiate(_rockPrefab,
48                         _smartGrid.Grid[i, j].LocalPosition,
49                         Quaternion.identity);
50     obj.transform.localScale = new Vector3(
51                         obj.transform.localScale.x,
52                         _enviromentHeightMultiplier * height,
53                         obj.transform.localScale.z);
54     obj.transform.SetParent(terrain.transform);
55     _worldElements.Add(obj);
56 }
57 }
58 }
59 }
60 }

```

La funzione controlla tutte le celle della SmartGrid e se una cella non appartiene a una stanza o a un corridoio allora verrà contrassegnata come zombie. Successivamente verrà creato l'oggetto corrispondente all'altitudine restituita dal perlin noise sulla cella selezionata. L'altitudine può essere regolata regolando il parametro *enviromentHeightMultiplier* che assume valori del dominio 1-10.

In conclusione, il perlin noise è uno strumento molto utile e versatile quando si parla di generazione procedurale. Nel contesto di Code Quest consente di ottenere ambienti di contorno differenti per ogni mappa che si genera mantenendo le forme naturali e smussate. Nella prossima verrà trattato come avviene il processo di generazione di una mappa completa con il plugin sviluppato.

9.4 Generazione delle mappe e Conclusioni

La generazione delle mappe all'interno di Code Quest è attualmente un'operazione possibile solo all'interno dell'editor, poiché l'attuale algoritmo non crea

sempre mappe adatte alle missioni cooperative. Infatti, sarebbe necessario un ulteriore sviluppo che consenta di stabilire se una mappa è adatta a un livello cooperative di Code Quest oppure no.

L'attuale plugin consente allo sviluppatore di creare mappe casuali che potranno essere salvate e inserite nella lista delle mappe da scegliere in fase di avvio della partita. Questo approccio consente di personalizzare il livello generato, consentendo il posizionamento di elementi mancanti e spostare eventuali elementi posizionati in zone poco adatte. L'algoritmo di generazione effettua diverse operazioni: la prima operazione è creare le partizioni sulla griglia eseguendo l'algoritmo BSP.

Successivamente genera i corridoi che connettono le varie stanze. Questa operazione viene effettuata prendendo il centro di una stanza casuale e viene cercato tra le rimanenti il centro della stanza che ha la distanza Manhattan minima. L'algoritmo termina quando non ci sono più stanze da connettere, garantendo la connessione di tutte le stanze. Creati i corridoi verranno create le stanze nella scena di gioco. Inoltre verranno create eventuali porte e corridoi esterni, nel caso in cui una stanza non sia adiacente ad un'altra.

Successivamente verrà chiamato il metodo *FillWorld* che creerà l'ambiente circostante tramite il perlin noise. Infine, verranno inseriti i punti di partenza e di arrivo; il seguente codice mostra quanto spiegato.

```

1 public void BuildWorld()
2 {
3     if (_smartGrid != null)
4     {
5         _smartGrid.SetIsEditable(true);
6         Partition(); //BSP
7         HashSet<SmartCell> corridors = BuildCorridors(); //Corridors
8         foreach (Room room in _rooms) //Rooms
9         {
10             _worldElements.Add(room.Build(corridors, _wallPrefab, _doorPrefab));
11         }
12         FillWorld(corridors); //World
13         _worldElements.Add(Instantiate(_spawnIndicatorPrefab, //Spawn
14                                         _spawn.LocalPosition, Quaternion.identity));
15         _worldElements.Add(Instantiate(_destinationIndicatorPrefab,
16                                         _destination.LocalPosition,
17                                         Quaternion.identity));
18     /*rest of the code*/
}

```

```
19 }  
20 }  
21 }
```

Una volta configurato il livello andrà creato un prefab che andrà aggiunto alla cartella CoopMaps contenuta nella cartella Resources. Inoltre, andrà aggiunto anche alla lista dei livelli da generare all'interno del *CoopMatchManager*.

In conclusione, questo sistema di generazione procedurale consente di creare livelli molto velocemente ottenendo nella maggior parte dei casi un risultato accettabile. Lo sviluppatore attualmente deve controllare che la mappa sia giocabile e ha la libertà di modificarla aggiungendo, rimuovendo e modificando i vari elementi.

Una possibile evoluzione potrebbe essere quella di generare livelli casualmente ad ogni partita, ma per avere un buon risultato bisognerebbe sviluppare anche un sistema che stabilisca la logica della partita con ad esempio la grammatica generativa, argomento non trattato in questa sede, in modo da avere dei livelli sempre differenti, ma al contempo giocabili e coinvolgenti.

In questo capitolo è stato trattato l'aspetto della generazione procedurale, un ambito in continuo sviluppo che consente di risparmiare parecchio tempo nella creazione di contenuti.

L'uso di algoritmi come BSP e Perlin Noise ha consentito lo sviluppo di un tool interno a unity che consente di creare livelli in modo rapido con anche il mondo che circonda le varie stanze, lasciando concentrare lo sviluppatore sullo sviluppo e configurazione del livello.

Capitolo 10

Gestione delle partite cooperative

10.1 Regole del gioco

La modalità cooperativa offerta da Code Quest è stata ideata per creare un'esperienza di gioco collaborativa, in cui i giocatori lavorano insieme per raggiungere un obiettivo condiviso. L'obiettivo principale di questa modalità è guidare tutti i partecipanti a raggiungere una specifica area sulla mappa. Partendo da un punto di partenza comune, i giocatori dovranno avanzare attraverso le varie stanze per sbloccare le vie che li condurranno all'obiettivo designato (vedi Figura 10.1).



Figura 10.1: Schermata modalità cooperativa

Ciascun giocatore dovrà occuparsi di problemi diversi lungo il percorso, ma tutti convergono verso il medesimo scopo: completare la missione raggiungendo

il luogo prestabilito. Il punto di partenza è contrassegnato da un fumogeno rosso, mentre l'obiettivo da raggiungere è indicato da un fumogeno verde (vedi Figura 10.2). A differenza della modalità competitiva, è possibile disconnettersi da una partita cooperativa senza terminarla, a condizione che tutti i giocatori non escano dalla sessione. Photon eliminerà automaticamente la stanza solo quando tutti i partecipanti saranno usciti o se la partita termina.

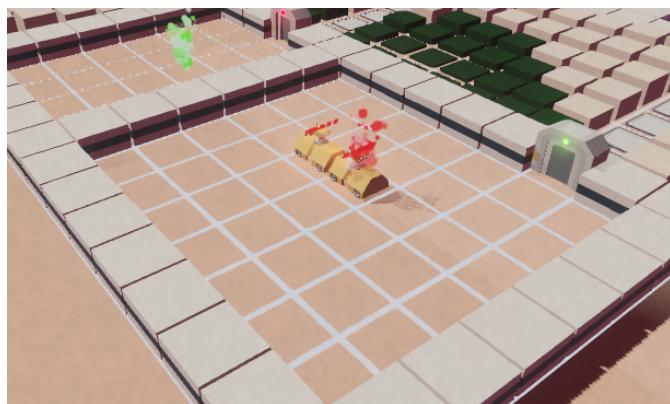


Figura 10.2: Rappresentazione del punto di partenza (fumo rosso) e del punto di arrivo (fumo verde)

Ogni partita cooperativa è vincolata da un limite di tempo, durante il quale i giocatori devono completare la missione. Se riescono a raggiungere l'obiettivo entro questo intervallo, la missione sarà considerata completata con successo. In caso contrario, la missione terminerà senza registrare statistiche o punteggi. Il sistema di matchmaking non memorizza le partite abbandonate prima del termine, ma i giocatori hanno la possibilità di riconnettersi a una partita abbandonata in precedenza, poiché random. Attualmente, la modalità offre tre mappe che vengono selezionate casualmente all'avvio della partita. In futuro, potranno essere aggiunte nuove mappe o un sistema completamente automatizzato per la generazione delle missioni e delle mappe.

Diversamente dalla modalità competitiva, la cooperativa presenta elementi di gioco differenti. Non è possibile posizionare alieni o torrette, ma ci sono oggetti funzionali al completamento del livello, come chiavi, porte e fotocellule. Questi elementi verranno descritti nel dettaglio nel paragrafo successivo. Un elemento di fondamentale importanza in questa modalità è la chat testuale, uno strumento che agevola la comunicazione tra i giocatori. Questo canale di comunicazione rappresenta un veicolo essenziale per collaborare efficacemente, consentendo ai partecipanti di scambiarsi informazioni, supportarsi reciprocamente nella ste-

sura delle soluzioni e discutere i ruoli e i compiti assegnati a ciascun membro della squadra.

In conclusione, la modalità cooperativa di Code Quest mira a coinvolgere i giocatori in un'esperienza di collaborazione, spingendoli a dividere un problema complesso in sotto problemi più gestibili. Attraverso il coordinamento, la comunicazione e la divisione delle sfide, i giocatori possono affinare le loro abilità di risoluzione dei problemi, programmazione e di lavoro di squadra.

10.2 Elementi di gioco

Nella modalità cooperativa di Code Quest, sono stati introdotti tre elementi molto legati: le chiavi, le porte e le fotocellule. Questi elementi sono stati ideati per mettere i giocatori nella condizione di dipendere l'uno dall'altro per poter raggiungere l'obiettivo finale

Iniziamo con le **chiavi**: esse sbloccano le porte chiuse e vengono raccolte quando il robot occupa la cella in cui sono posizionate. Le chiavi sono raccolte tramite un trigger, che si attiva alla collisione con il robot. Una volta che il trigger viene attivato la chiave viene disabilitata e si sblocca la porta a cui è legata la chiave, questa informazione viene condivisa globalmente nella room, impostando su true il parametro delle proprietà della room legato alla porta. Questi parametri sono contenuti all'interno delle *CustomProperties* della stanza, ossia degli elementi chiave valore sincronizzati lato server all'interno di un dizionario. Nella Figura 10.3 è mostrato un esempio di una chiave.



Figura 10.3: Chiave posizionata in una mappa di gioco

Nella Figura 10.4 è rappresentata una combinazione di una porta e di una fotocellula.

Le **porte**, a loro volta, rappresentano una tappa critica nel percorso dei giocatori. Possono essere configurate in vari modi, dando un tocco di variabilità alle sfide. Alcune porte possono rimanere aperte una volta che si attiva il trigger di apertura, altre si chiudono automaticamente dopo un certo periodo di tempo e altre ancora richiedono l'uso delle fotocellule per mantenere il loro stato aperto. Il cambiamento di colore delle luci sulla parte superiore delle porte, da rosso a verde, è un chiaro segnale visivo per i giocatori riguardo allo stato delle porte.

Le **fotocellule** aggiungono un elemento di profonda collaborazione. Questi trigger, contrassegnati da una luce viola, mantengono le porte aperte solo quando un robot è all'interno dell'area specificata. Questo significa che i giocatori dovranno coordinarsi e lavorare insieme per superare le sfide che richiedono l'uso delle fotocellule. Questo elemento di gioco enfatizza l'importanza della cooperazione e della comunicazione tra i giocatori.



Figura 10.4: Rappresentazione di una fotocellula (luce viola) porta (luce verde)

Riassumendo, gli elementi introdotti nella modalità cooperativa di Code Quest sono stati pensati per mettere in primo piano la collaborazione tra i partecipanti. Le chiavi, le porte e le fotocellule creano situazioni in cui i giocatori devono lavorare insieme, coordinandosi e comunicando efficacemente per superare gli ostacoli e raggiungere l'obiettivo.

10.3 Chat e comunicazione

Nel contesto cooperativo, la comunicazione riveste un ruolo fondamentale, dato che senza di essa sarebbe difficile raggiungere la maggior parte degli obiettivi. All'interno del nostro contesto, la coordinazione tra i membri è stata agevolata attraverso una chat testuale, che è stata implementata grazie all'utilizzo del plugin Photon Chat. Con poche righe di codice è stato possibile mettere in comunicazione i giocatori nella varie partite.

Ogni partita ha una propria stanza identificata dal nome "*Match_*" seguito dal codice della stanza. Quando un giocatore entra nella scena di gioco, viene automaticamente aggiunto al canale della stanza e rimosso quando si disconnette.

La funzione "*ConnectToChat*" ha il compito di connettersi alla chat al momento del caricamento della scena:

```
1 public void ConnectToChat()
2 {
3     if (PhotonNetwork.InRoom)
4     {
5         chatClient = new ChatClient(this);
6
7         if (PhotonNetwork.NickName.Length == 0)
8             PhotonNetwork.NickName = "Player" + Random.Range(0, 1000);
9
10        isConnected = chatClient.Connect(
11            PhotonNetwork.PhotonServerSettings.AppSettings.AppIdChat,
12            PhotonNetwork.AppVersion,
13            new AuthenticationValues(PhotonNetwork.NickName));
14
15        if (isConnected)
16            Debug.Log("Chat Connecting");
17        else
18            Debug.LogError("Chat connection failed");
19    }
20}
```

Quando la connessione alla chat è stata stabilita, viene eseguita la callback "*OnConnected*". Essa ha il compito di registrare il giocatore al canale della partita e di inizializzare la lista dei messaggi:

```

1 public void OnConnected()
2 {
3     Debug.Log("Chat Connected");
4     currentChannel = "Match_" + PhotonNetwork.CurrentRoom.Name;
5     chatClient.Subscribe(new string[] { currentChannel });
6
7     messages = new List<string>();
8 }
```

Il metodo "*OnGetMessages*" viene chiamato quando ci sono nuovi messaggi in arrivo. Questa callback , se il canale coincide con quello attuale, provvederà ad aggiornare la lista dei messaggi e riformattare le stringhe, aggiungendo l'autore seguito dal messaggio stesso:

```

1 public void OnGetMessages(
2     string channelName,
3     string[] senders,
4     object[] messages)
5 {
6     if (channelName == currentChannel)
7     {
8         HasNewMessages = true;
9         for (int i = 0; i < senders.Length; i++)
10        {
11            this.messages.Add(senders[i] + ": " + messages[i]);
12        }
13    }
14 }
```

Per mantenere attiva la connessione e ricevere i messaggi, viene invocato il metodo "*Service*". Questo avviene all'interno della funzione "*Update*" poiché è un'operazione che richiede un aggiornamento continuo:

```
1 private void Update()
2 {
3     if (PhotonNetwork.InRoom && isConnected)
4     {
5         chatClient.Service();
6     }
7 }
```

L'invio dei messaggi avviene attraverso la funzione "*SendMessage*" che pubblicherà il messaggio passato come parametro sul canale a cui è registrato l'utente:

```
1 public void SendMessage(string message)
2 {
3     if (PhotonNetwork.InRoom)
4     {
5         chatClient.PublishMessage(currentChannel, message);
6     }
7 }
```

L'interazione con l'interfaccia grafica è gestita attraverso la classe denominata *ChatManagerUI*, la quale ha il compito di aggiornare la chat quando un nuovo messaggio arriva e di consentire l'invio dei messaggi stessi.

Per inviare un messaggio, gli utenti possono cliccare sul tasto situato alla destra del campo di inserimento, vedi Figura 10.5, oppure premere il tasto "Invio" sulla tastiera. Al momento della ricezione di un nuovo messaggio, viene attivato un segnale sonoro per notificare l'utente.



Figura 10.5: Pannello della chat

Nel caso in cui la finestra della chat sia chiusa, fornita anche una notifica visiva tramite l'attivazione di un punto rosso sull'icona della chat, posizionata al centro dell'interfaccia, Figura 10.6. Quando la chat viene aperta questo elemento viene disabilitato.



Figura 10.6: Rappresentazione del segnale di notifica

In conclusione, la chat assume un ruolo cruciale all'interno dell'esperienza di gioco cooperativo, dando la possibilità ai giocatori di comunicare e gestire le operazioni da fare. L'uso del plugin Photon Chat ha permesso di configurare in modo semplice tutta l'infrastruttura necessaria per il funzionamento del servizio di messaggistica, consentendo un agevole integrazione di questa importante funzionalità.

10.4 Conclusioni

In questo capitolo è stata analizzata la modalità cooperativa del gioco, descrivendo le meccaniche cooperative introdotte e mettendo in evidenza l'importanza della comunicazione per poter completare le partite.

Come per la modalità competitiva, viene proposto un approccio di programmazione a micro-problemi; poiché si vuole far concentrare il giocatore sulla suddivisione dei problemi complessi in sotto-problemi più piccoli. Inoltre, sono stati introdotti nuovi elementi come le porte, le chiavi e le fotocellule.

Le fotocellule sono un elemento molto importante dal punto di vista della cooperazione poiché richiede la coordinazione di entrambi i giocatori per poter proseguire nel livello.

In fine, è stata analizzata la chat, strumento fondamentale per la comunicazione durante le partite.

Capitolo 11

Risultati e conclusioni

11.1 Risultati

Con l'aggiunta del multiplayer si è giunti alla conclusione della seconda fase di sviluppo di Code Quest, il prodotto attualmente si presenta con tre livelli single-player attentamente progettati, che svolgono un ruolo fondamentale nell'agire come tutorial introduttivo al mondo di gioco, e di 2 modalità multigiocatore.

Nella modalità competitiva, gli utenti possono scegliere tra due varianti: la frenetica sfida del *Time Attack* e il tattico *Stop and Go*. Queste opzioni assicurano un'esperienza diversificata, catturando sia chi cerca la pura velocità che chi preferisce una strategia più riflessiva. La modalità cooperativa si compone di tre livelli, che verranno scelti in maniera casuale ad ogni partita. Questa caratteristica garantisce un grado di varietà e rigiocabilità notevolmente elevato, con il crescere dei livelli proposti.

Il prossimo passo da fare sarà quello di sottoporre le modalità di gioco realizzate a una fase di testing sul campo, coinvolgendo utenti reali. Questa fase consentirà di ottenere un feedback, fornendo indicazioni chiare su quali tra le modalità siano più coinvolgenti e funzionali nel campo dell'apprendimento.

Il processo di sviluppo naturalmente non termina qui, ci sono numerose migliorie che possono essere fatte, ma sicuramente il risultato ottenuto può essere già uno strumento valido da testare sul campo.

11.2 Sviluppi futuri

Code Quest è un progetto che può essere migliorato e aggiornato con nuove funzionalità per renderlo sempre più efficace nell'apprendimento della programmazione. L'uso delle meccaniche di gioco competitive e cooperative e gli elementi di game design introdotti danno parecchia libertà di realizzare nuove modalità e tipologie di livelli.

Come più volte accennato nel corso di questo documento ci sono diversi aspetti che potrebbero essere portati avanti. Il primo è quello di realizzare un sistema automatico di generazione dei livelli che viene eseguito all'avvio della partita, rendendola unica e diversa da tutte le altre. Questo processo richiede uno studio approfondito della generazione procedurale dei livelli, poiché bisogna capire come posizionare gli elementi nel mondo di gioco e quale deve essere il percorso che i giocatori devono seguire, evitando situazioni banali.

Un ulteriore sviluppo riguarda la modalità competitiva che potrebbe essere ampliata a nuove varianti, simili a quelle esistenti. Un'altra interessante integrazione è quella dell'introduzione delle squadre formate da più giocatori. Una modalità interessante in questo contesto potrebbe essere il torneo in modo da rendere il tutto ulteriormente competitivo.

Tuttavia, la competitività e la cooperazione dovrebbero essere testate sul campo in modo da capire quale modalità sia più efficace nel contesto dell'apprendimento. Questo perché, come discusso nei capitoli iniziali, si hanno aspetti positivi e negativi sulla motivazione dello studente. Essendo ancora un tema in fase di studio e non essendoci ancora dati sufficienti per avere un quadro chiaro su quale modalità sia più adatta, Code Quest potrebbe essere un interessante strumento di studio in questo ambito.

Un altro sviluppo interessante è quello di rendere Code Quest compatibile con altri linguaggi, dando la possibilità di programmare anche fuori dall'interfaccia grafica del gioco, tramite l'IDE personale come IntelliJ o Visual Studio. Questo potrebbe ampliare anche la tipologia di utenti andando anche coinvolgere studenti di livelli d'istruzione più avanzati. Questo tipo di integrazione chiaramente richiederebbe uno sviluppo di un'infrastruttura dedicata esterna al gioco stesso e che sia in grado di comunicare con esso. Inoltre, potrebbero essere sviluppate ulteriori modalità più adatte all'ambiente in cui si vuole utilizzare Code Quest.

Per concludere, questo progetto presenta numerosi elementi che possono essere portati a un livello più avanzato. La struttura del codice è stata progettata con l'obiettivo di agevolare al massimo le future modifiche, facilitando l'implementazione di nuove modalità e funzionalità.

Un aspetto fondamentale è rappresentato dalla necessità di condurre test sul campo. Questi test consentiranno di prendere le decisioni più adatte per portare avanti lo sviluppo e di comprendere in che modo le diverse meccaniche di gioco influenzino i giocatori nel contesto dell'apprendimento del coding.

Il progetto ha un potenziale considerevole per ulteriori sviluppi. La struttura del codice agevola l'aggiunta di nuove caratteristiche, mentre i test sul campo rappresentano un tassello fondamentale per orientare le scelte future e per valutare l'impatto delle diverse dinamiche di gioco sulla formazione nell'ambito della programmazione.

11.3 Conclusioni

Code Quest è un progetto che può essere un interessante strumento per introdurre studenti delle scuole medie e superiori alla programmazione.

L'introduzione del nuovo sistema di movimento basato sulla SmartGrid, ha reso più semplice la creazione dei livelli e consente di avere un sistema di movimento deterministico, a differenza del sistema precedente basato sulla fisica che non essendo deterministico in certi casi rischiava di essere frustrante e complicato.

Inoltre, la SmartGrid si è dimostrata uno strumento molto flessibile anche per quanto riguarda lo sviluppo di un sistema per il calcolo del percorso per raggiungere un determinato punto su di essa tramite l'algoritmo A*. Questo tipo di struttura può essere utilizzata in numerosi ambiti esterni a Code Quest poiché è un componente che è stato progettato per essere indipendente in modo da poter essere utilizzato anche in altri progetti.

L'integrazione del servizio web offre un'interfaccia web che permette ai giocatori di monitorare i propri progressi e quelli degli altri utenti. Nell'ambito della sicurezza degli account, si è prestata particolare attenzione: l'introduzione di un sistema centralizzato come Keycloak ha consentito di affidare a questo servizio l'autenticazione e la gestione dei profili degli utenti. Questo approccio ha reso molto semplice la configurazione di tutti gli endpoint, assicurando un'elevata flessibilità. Keycloak, così come altri sistemi SSO, danno la possibilità di integrare sistemi di autenticazione esterni, come Google, Facebook, Github, e altri ancora rendendo sempre più flessibile l'accesso al servizio. Inoltre, è possibile configurare in modo molto semplice diverse funzionalità come: la modalità di autenticazione a due fattori e l'applicazione di policy per le password e altri dati.

Attraverso l'uso di uno strumento dedicato, ogni utente ha la possibilità di configurare il proprio client per collegarsi a un server specifico, sul quale è ospitato il servizio web. Questo approccio apre la strada a una gestione mirata degli utenti, limitando l'accesso a una comunità circoscritta di giocatori. Un possibile scenario potrebbe essere che ogni scuola abbia il proprio ambiente di gioco indipendente, attraverso un servizio web Code Quest autonomo e server di gioco isolati da altre scuole.

L'integrazione di componenti per la generazione procedurale dei livelli non solo accelera il processo di sviluppo, ma fornisce anche una solida base per l'eventuale creazione di un sistema automatico di generazione delle partite. Le modalità di gioco che sono state sviluppate sono state concepite per mantenere alta la motivazione dei giocatori e per stimolarli ad applicare gli elementi della programmazione in maniera sempre più avanzata.

Alcuni dei requisiti iniziali del progetto sono stati posticipati per ulteriori sviluppi futuri. Ad esempio, la modalità torneo è stata intenzionalmente sospesa al fine di comprenderne appieno gli effetti della modalità competitiva sviluppata sull'apprendimento e, se opportuno, sarà introdotta in futuro.

Infine, questo progetto ha consentito di applicare numerose conoscenze di ambiti diversi poiché la creazione di un videogioco, ancor di più se multiplayer, è sicuramente una sfida che necessita la conoscenza di diverse tecnologie.

Bibliografia

- [1] Klaudia Bovermann, Sebastian Habla, and Joshua Weidlich. Effects of competition in gamified online distance learning on intrinsic motivation: A comparative case study. 07 2021.
- [2] Michael Sailer, Jan Ulrich Hense, Sarah Katharina Mayr, and Heinz Mandl. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior*, 69:371–380, 2017.
- [3] Anton Gustafsson, Cecilia Katzeff, and Magnus Bang. Evaluation of a pervasive game for domestic energy engagement among teenagers. *Comput. Entertain.*, 7(4), 12 2010.
- [4] Lisa Legault. Intrinsic and extrinsic motivation. *Encyclopedia of Personality and Individual Differences*, 11 2016.
- [5] Konrad Fischer, Sarah Vaupel, Niels Heller, Sebastian Mader, and François Bry. *Effects of Competitive Coding Games on Novice Programmers*, pages 464–475. 3 2021.
- [6] Codecombat. <https://codecombat.com/>, 7 2023. Visited on: 2023-7-1.
- [7] Codinggame. <https://www.codingame.com/start>, 7 2023. Visited on: 2023-7-1.
- [8] Screeps. Introduction. <https://docs.screeeps.com/introduction.html>, 7 2023. Visited on: 2023-7-1.
- [9] Getting started. <https://spring.io/guides/tutorials/rest/>, 7 2023. Visited on: 2023-7-1.
- [10] Spring.io. <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Service.htm>, 7 2023. Visited on: 2023-7-9.
- [11] OAuth 2.0 — OAuth. <https://oauth.net/2/>. Visited on: 2023-8-21.
- [12] Rfc 6749: The oauth 2.0 authorization framework. <https://datatracker.ietf.org/doc/html/rfc6749>, 7 2023. Visited on: 2023-7-15.

- [13] How OpenID connect works - OpenID foundation. <https://openid.net/developers/how-connect-works/>, February 2023. Visited on: 2023-8-21.
- [14] Openid.net. [https://openid.net/specs/openid-authentication-2\0-12.html](https://openid.net/specs/openid-authentication-2_0-12.html), 7 2023. Visited on: 2023-7-15.
- [15] Keycloak: Open source identity and access management for modern applications and services. <https://github.com/keycloak/keycloak>, 7 2023. Visited on: 2023-7-18.
- [16] Keycloak. <https://www.keycloak.org/>, 7 2023. Visited on: 2023-7-18.
- [17] Unity Technologies. Materials, shaders & textures. <https://docs.unity3d.com/560/Documentation/Manual/Shaders.html>. Visited on: 2023-7-28.
- [18] Unity Technologies. Compute shaders. <https://docs.unity3d.com/Manual/class-ComputeShader.html>. Visited on: 2023-7-28.
- [19] Stuart Russell and Peter Norvig. *Artificial intelligence: A modern approach: United States edition*. Pearson, Upper Saddle River, NJ, 2 edition, 1999.
- [20] Amir Yahyavi and Bettina Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Computing Surveys (CSUR)*, 46, 10 2013.
- [21] Multiplayer game development made easy. <https://www.photonengine.com/>. Visited on: 2023-7-29.
- [22] 7 - player networking. <https://doc.photonengine.com/pun/current/demos-and-tutorials/pun-basics-tutorial/player-networking>. Visited on: 2023-7-29.
- [23] Master client and host migration. <https://doc.photonengine.com/pun/current/gameplay/hostmigration>. Visited on: 2023-7-29.
- [24] Remote procedure calls. <https://doc.photonengine.com/pun/v1/gameplay/rpcsandraiseevent>. Visited on: 2023-7-29.
- [25] Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, 2014.
- [26] Bruce Naylor, John Amanatides, and William Thibault. Merging bsp trees yields polyhedral set operations. *SIGGRAPH Comput. Graph.*, 24(4):115–124, sep 1990.
- [27] Wikipedia contributors. Convex set. https://en.wikipedia.org/w/index.php?title=Convex_set&oldid=1169577292, August 2023. Visited on: 2023-8-21.

- [28] Huang Minjie, Zhou Yaoming, Wang Yongchao, and Liu Zhongtie. An efficient adaptive space partitioning algorithm for electromagnetic scattering calculation of complex 3d models. *Journal of Systems Engineering and Electronics*, 32(5):1071–1082, 2021.
- [29] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph.*, 21(4):153–162, aug 1987.
- [30] Mike Paterson and F. Frances Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1990.
- [31] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, jul 1985.