Opposite Elements
“Programmazione ad Oggetti”

Barone Benedetta, Lucarelli Sebastiano, Pedini Emily

28 ottobre 2025

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

Il software realizzato è un platform-puzzle game che prende ispirazione dal gioco originale "Fireboy and Watergirl". Per il superamento del livello è necessario far cooperare i due player Fireboy e Watergirl. Tale gioco è suddiviso in 3 livelli, per accedervi è necessario superare il precedente.
Superare il livello consiste nel raggiungimento delle rispettive porte, quella contrassegnata di rosso per fireboy e quella di azzurro per watergirl. Gli obiettivi da raggiungere sono: la raccolta di diamanti acqua e fuoco e il completamento del livello in un tempo prestabilito; per fare ciò i due giocatori devono cooperare premendo pulsanti, attivando leve ed evitando ostacoli che possono portare al game over.

Requisiti funzionali

- All'interno dei livelli è possibile raccogliere diamanti di acqua per watergirl e di fuoco per fireboy. Il mancato raccoglimento di diamanti porta al non superamento di tale obiettivo.
- Altro obiettivo da superare è il tempo: i due personaggi cooperando devono essere in grado di raggiungere le porte in un tempo prestabilito.
- Gli oggetti ostacolo per i personaggi sono:
  - Le pozze, che possono essere di tre tipi: acqua il cui tocco da parte di fireboy causa game over, fuoco il cui tocco da parte di watergirl causa game over e acido il cui tocco da parte di entrambi i giocatori causa game over.
  - Le piattaforme mobili che se non attivate bloccano il passaggio.

- Gli oggetti utili al completamento del livello sono:
  - Bottoni che se premuti attivano le piattaforme mobili e che se rilasciati le disattivano riportandole alle posizioni originali.
  - Leve che si comportano come bottoni ma che mantengono attivo il proprio stato e anche quello delle piattaforme mobili.
  - Ventole che creano getti d'aria che permettono ai giocatori di rimbalzare per raggiungere una determinata piattaforma.
- Il gioco mette a disposizione:
  - Un menù di pausa con opzioni di restart, continue e ritorno alla home per leggere le istruzioni di gioco.
  - Una schermata di selezione livello che permette di selezionare i livelli sbloccati e di rigiocare eventuali livelli.

Requisiti funzionali opzionali

Tra le funzionalità opzionali implementate nel progetto, troviamo:

- Le leve sopracitate.
- Gli effetti grafici come l'animazione dei movimenti dei giocatori, l'apertura delle porte, l'attivazione delle ventole e delle leve.

Le funzionalità opzionali che non è stato possibile implementare all'interno del monte ore del progetto sono:

- Audio di gioco per determinate azioni, come salti e raccolta di gemme.
- Implementazione di livelli extra.

Requisiti non funzionali

- Per rendere l'esperienza di gioco più gradevole, quest'ultimo dovrà essere fluido.
- Gestione ottimale delle immagini garantendo una visualizzazione fluida ed efficiente.

1.2 Modello del Dominio

In Opposite Elements i due giocatori interagiscono solo con l'ambiente di gioco, quindi con tutti gli oggetti presenti all'interno del livello.
Ogni interazione porta a diversi comportamenti:

- Le collisioni con le pozze possono portare al game over.
- Le collisioni con le porte portano al superamento del livello.
- Le collisioni con i diamanti portano al superamento del relativo obiettivo.
- Le collisioni con leve e bottoni permettono ad essi stessi di attivarsi, così da innescare lo spostamento delle relative piattaforme mobili.
- Le collisioni con le ventole portano alla generazione di un vento che fa rimbalzare i giocatori.
- Le collisioni con i bordi del livello limitano lo spazio di movimento dei personaggi.

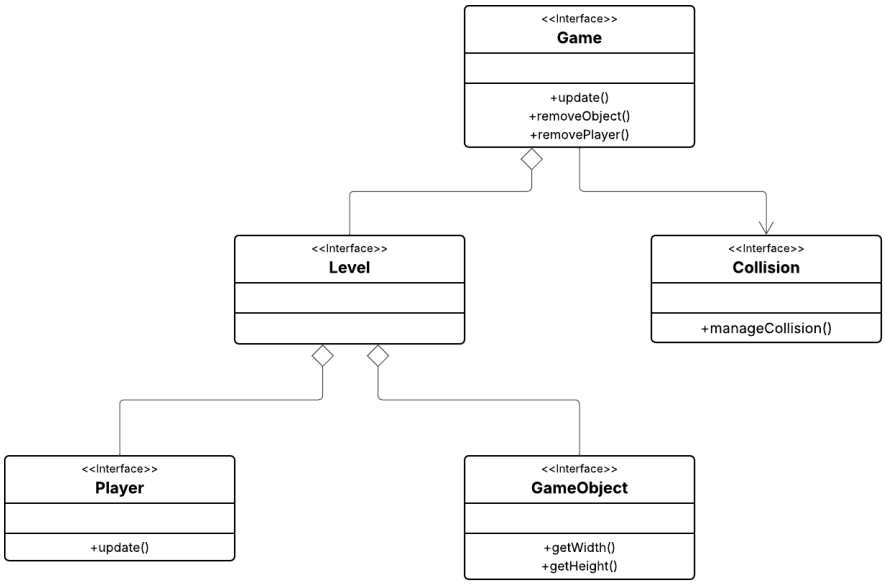

Figura 1.1: Schema UML del dominio applicativo.

Capitolo 2

Design

2.1 Architettura

L'architettura di Opposite Elements segue il pattern architetturale MVC. Nel Modello sono contenuti tutti gli elementi di gioco (personaggi, piattaforme, gemme, bottoni, porte...), le regole che ne determinano il comportamento e le interazioni. Il modello si occupa di aggiornare lo stato degli oggetti e dei giocatori. Inoltre gestisce la fisica di base, ossia movimenti, gravità e collisioni. La gestione del livello di gioco, per quanto riguarda la componente model, viene gestita in Game.
Nel Controller si coordina l'esecuzione del gioco, gestendo il ciclo di aggiornamento e gli input dell'utente. Il controller aggiorna periodicamente il modello verificandone le fasi di gioco, come l'avvio, la pausa, la selezione di un livello, il completamento di un livello...
Il controller funge da intermediario tra model e view, principalmente in GameEngine, permettendo alla view di occuparsi unicamente della presentazione grafica, rimanendo così un componente passivo. La view, come accennato precedentemente, rimane completamente indipendente da model e controller, limitandosi a raffigurare i contenuti e a notificare gli eventi mediante callback.

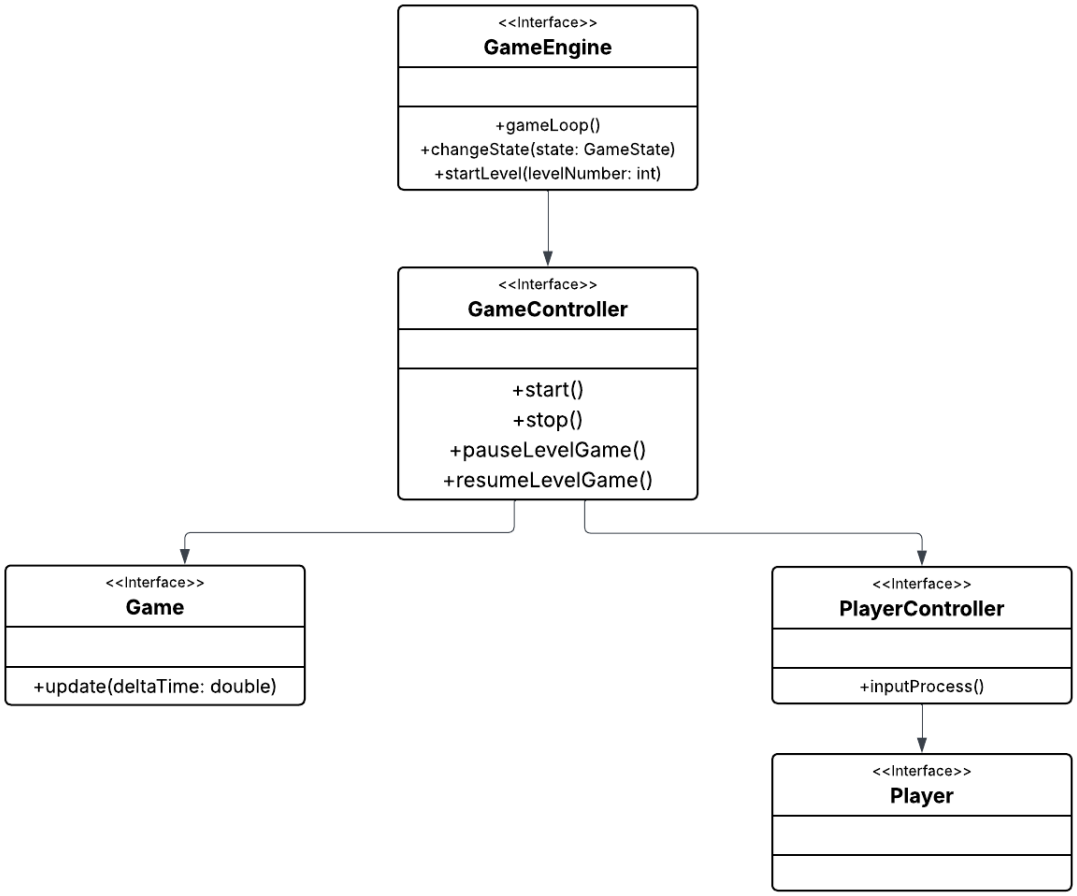

Figura 2.1: Schema UML architetturale di Opposite Elements.

2.2 Design dettagliato

2.2.1 Barone Benedetta

Gestione dei personaggi e logica di movimento
Problema: Il gioco prevede la rappresentazione di due personaggi distinti ma che condividono la stessa logica di base come movimento, salto, gravità, collisioni ma mantenendo le proprie caratteristiche.

Soluzione: Viene definita l'interfaccia Player per stabilire i comportamenti comuni a tutti e due i personaggi, specificandone la gestione degli input, della posizione e dello stato. Invece la classe astratta AbstractPlayer implementa la logica condivisa come il movimento orizzontale e verticale, la gestione delle collisioni con il terreno e con i limiti dello schermo e infine l'applicazione della gravità. Le classi Fireboy e Watergirl estendono l'abstract, differenziandosi solo per il tipo di giocatore, fire e water. L'uso di una classe astratta mi consente di riutilizzare tutto il comportamento condiviso tra i personaggi, facilitando modifiche o aggiornamenti futuri.

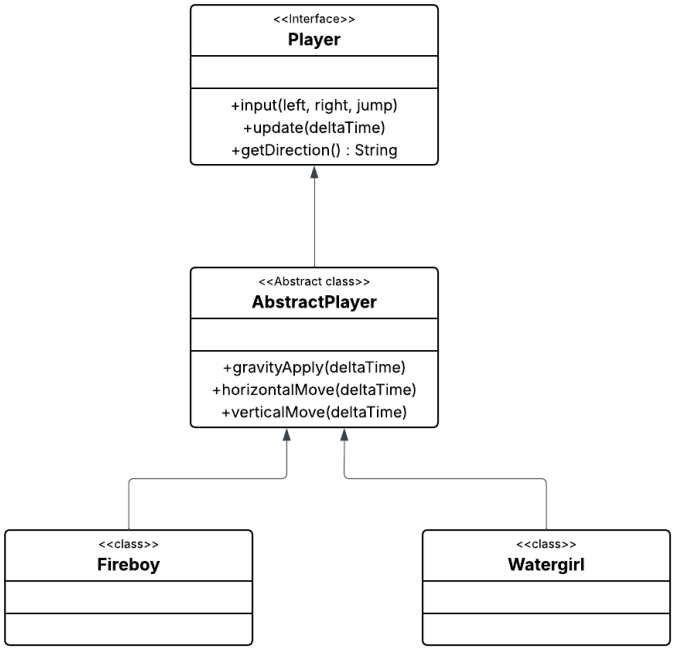

Figura 2.2: Schema UML per la creazione e gestione dei personaggi.

Gestione degli input da tastiera
Problema: Il gioco, per ogni personaggio, deve gestire diversi input da tastiera, permettendo a ciascuno di muoversi in modo indipendente.

Soluzione: La gestione degli input è realizzata nella classe PlayerControllerImpl, che implementa l'interfaccia KeyListener per catturare gli eventi di pressione e rilascio dei tasti. Ogni tasto è associato ad un'azione specifica per i due giocatori e lo stato dei tasti viene mantenuto tramite variabili booleane che indicano se un determinato comando è attivo o meno. Attraverso il metodo inputProcess() vengono aggiornati i due personaggi in base agli input rilevati.

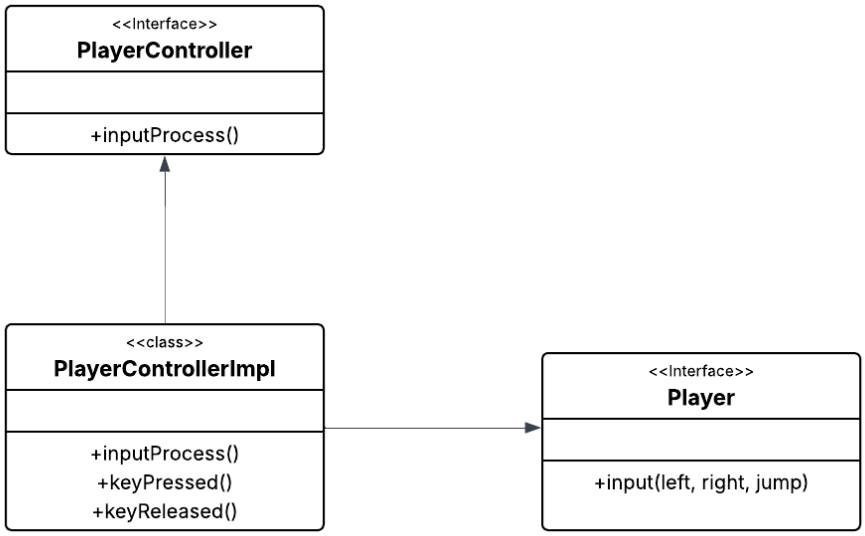

Figura 2.3: Schema UML per la gestione degli input.

2.2.2 Lucarelli Sebastiano

Creazione degli oggetti di gioco
Problema: Il gioco prevede la creazione e la gestione di diversi oggetti con cui il giocatore può interagire (bottoni, leve, piattaforme mobili...). La scorretta gestione di questi elementi può comportare un'importante duplicazione di codice (considerando l'elevato quantitativo di oggetti), oltre che di difficoltà durante la manutenzione e durante il caricamento degli stessi all'interno dei vari livelli giocabili.

Soluzione: Per affrontare questi aspetti è stata adottata una struttura modulare basata sulla creazione di un'interfaccia e di una sua implementazione all'interno di una classe astratta, denominate rispettivamente "GameObjects" e "AbstractGameObjects". All'interno della classe astratta sono stati inseriti tutti i metodi comuni ai singoli oggetti, così da evitare eventuali duplicazioni di codice ed andando a facilitare la creazione di nuovi oggetti. Questa struttura permette anche di semplificare il richiamo dei metodi durante la futura fase di creazione dei livelli, evitando così la creazione di metodi con stesse funzionalità ma con nomi differenti. Ogni oggetto è stato successivamente gestito con una classe distinta contenente i metodi necessari al funzionamento dello specifico elemento. Utilizzando questa struttura la creazione degli oggetti di gioco risulta più scalabile, permettendo con facilità di aggiungere eventuali futuri oggetti.

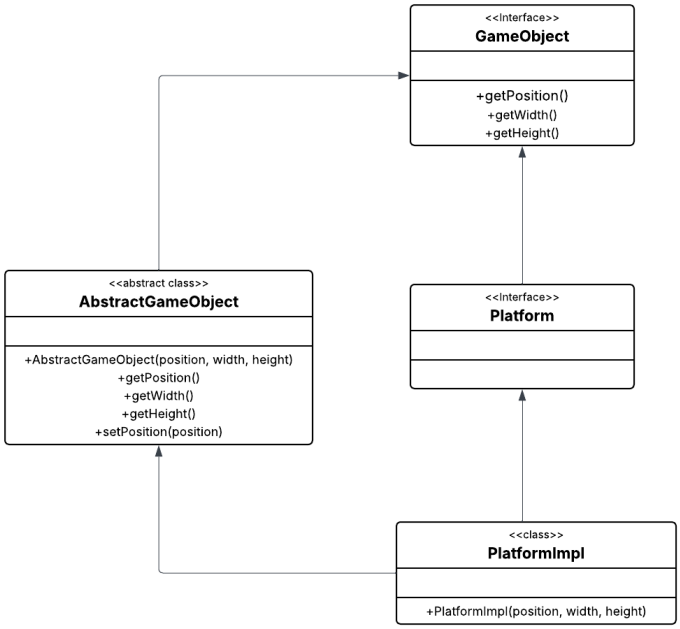

Figura 2.4: Schema UML per la creazione degli oggetti di gioco. Platform rappresenta un oggetto qualsiasi.

Gestione dei livelli

Problema: Quando un livello viene avviato è necessario andare ad inizializzare ogni singolo oggetto per renderlo successivamente visibile. La gestione manuale di questo caricamento può risultare complesso oltre che comportare un elevato quantitativo di errori.

Soluzione: Per ovviare questa problematica si è deciso di andare a gestire i livelli utilizzando un formato in griglia, la quale viene suddivisa in celle contenenti un riferimento al relativo oggetto di gioco. Ogni livello viene inizializzato quindi con la lettura di due file di testo contenti il primo la struttura in griglia base del livello e il secondo l'elenco degli oggetti e le relative posizioni in griglia. L'insieme degli oggetti letti viene poi inizializzato all'interno della classe "LevelLoaderImpl", responsabile della creazione di due liste contenenti l'elenco degli oggetti e dei giocatori. In questo modo diventa molto più semplice effettuare la manutenzione dei livelli, permettendo di modificarne l'intera struttura cambiando solamente i riferimenti nei relativi file di testo. Durante la fase di caricamento del livello gestito della LevelLoaderImpl si vanno anche a caricare le immagini dal classpath visualizzate poi nelle rispettive posizioni in griglia.

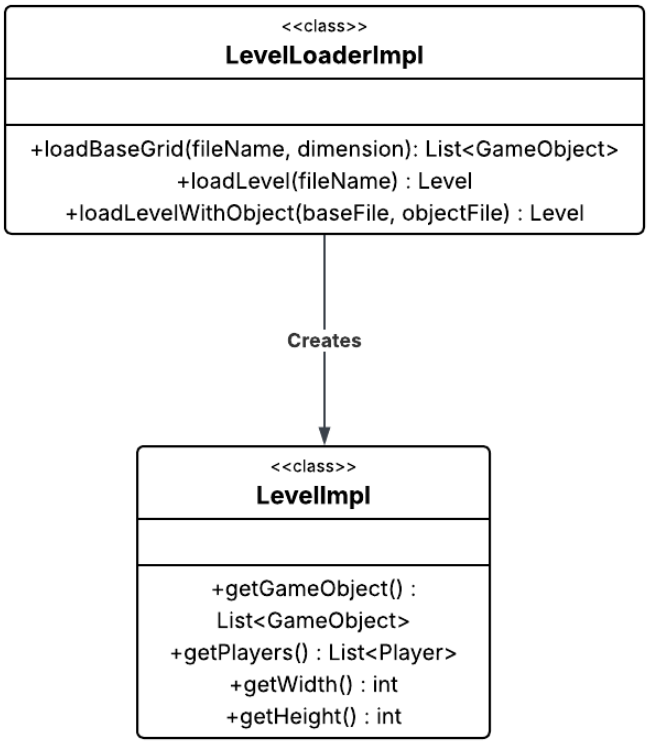

Figura 2.5: Schema UML per il caricamento dei livelli.

2.2.3 Pedini Emily

Creazione delle collisioni
Problema: Nel livello i giocatori devono interagire con gli oggetti di gioco precedentemente analizzati. Ogni collisione con uno specifico oggetto si comporta diversamente.

Soluzione: Per la creazione delle collisioni è stato adottato il metodo "Factory Method Pattern" tramite l'interfaccia CollisionFactory. Ogni metodo della factory rappresenta un tipo specifico di collisione: riceve in ingresso gli oggetti coinvolti (ad esempio un player e un diamante) e restituisce un oggetto Collisions che rappresenta l'effetto da applicare al gioco. In questo modo è più semplice aggiungere nuove tipologie di collisione, dovute a nuovi oggetti di gioco, aggiungendo alla factory gli appositi metodi.

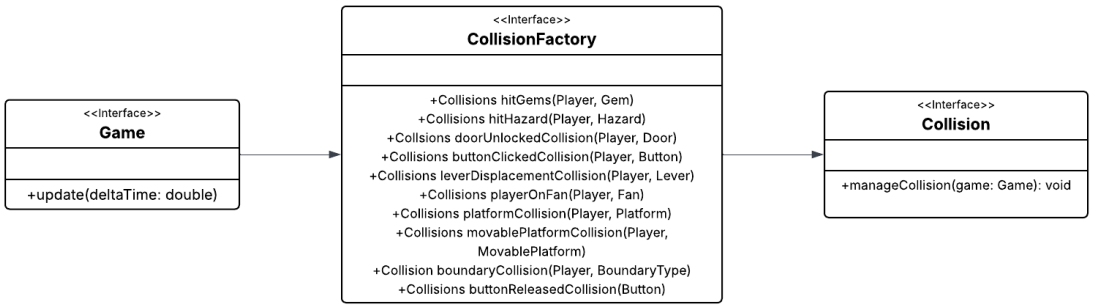

Figura 2.6: Schema UML che rappresenta la creazione di collisioni.

Gestione collisioni
Problema: Una volta rilevata una collisione tra un giocatore e un oggetto di gioco, deve essere applicato l'effetto corrispondente. E' necessario avere un sistema in grado di gestire più collisioni in modo ordinato evitando che l'esecuzione immediata di ognuna interferisca con altre collisioni o con l'aggiornamento della logica di gioco.

Soluzione: Per risolvere questo problema è stata implementata la classe CollisionQueue che si occupa di gestire le collisioni rilevate in un determinato ciclo di aggiornamento. Ogni volta che viene individuata una collisione, il corrispondente oggetto Collisions viene inserito nella coda. Terminata la fase di rilevamento la coda viene elaborata tramite il metodo manageCollisions.

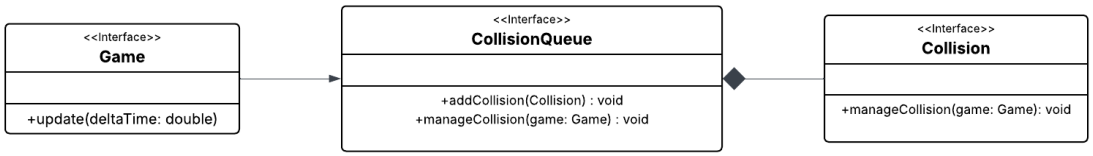

Figura 2.7: Schema UML per la gestione delle collisioni.

Gestione del Game Loop e degli Stati di gioco
Problema: Durante l'esecuzione del gioco è necessario aggiornare costantemente la logica interna (come i movimenti, le collisioni, le interazioni...) ma anche gestire i diversi stati in cui il gioco può trovarsi (come menù principale, selezione livello, partita in corso...). Il problema principale è quindi quello di coordinare l'esecuzione del Game Loop con il cambio degli stati di gioco, garantendo il corretto caricamento delle interfacce.

Soluzione: Per affrontare il problema del Game Loop è stato adottato il "Game Loop Pattern", implementato nella classe GameControllerImpl, che è quindi responsabile del costante aggiornamento del gioco. Il loop viene eseguito in un thread dedicato, in modo da non bloccare l'interfaccia grafica e consentire un aggiornamento continuo. La classe GameEngine invece gestisce gli stati globali del gioco attraverso l'enumerazione GameState. Ogni stato determina quale logica eseguire e quale schermata mostrare. Questo permette di mantenere un gioco estendibile aggiungendo semplicemente un valore nel GameState definendo poi la relativa schermata.

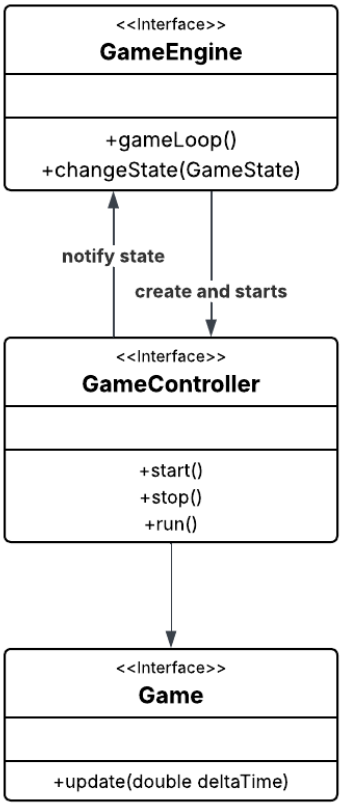

Figura 2.8: Schema UML per la gestione del Game Loop.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per il progetto è stato realizzato un insieme di test automatizzati JUnit focalizzati principalmente sul modello logico del livello. L'obiettivo è stato quello di garantire la correttezza del comportamento interno. Non sono stati realizzati test automatici sull'interfaccia grafica dato che le funzionalità della view sono state verificate direttamente durante lo sviluppo. Questa scelta è stata fatta ritenendo che fosse sufficiente testare automaticamente le parti più soggette ad errori come logica di gioco e gestione di collisioni.
Componenti testate automaticamente:

- LevelLoaderTest: che verifica il corretto caricamento dei livelli, dei personaggi e degli oggetti da file txt.
- ObjectsTest: che testa i principali comportamenti degli oggetti di gioco e si assicura che rispondano ad attivazioni e disattivazioni in modo corretto.
- PlayerMovementTest: che controlla il corretto funzionamento dei movimenti base dei personaggi (destra, sinistra e salto) verificando anche che direzione e velocità siano coerenti.
- TestCollision: che testa la logica di collisione tra player e oggetti di gioco, assicurando che gli stati del gioco vengano aggiornati correttamente.

3.2 Note di sviluppo

3.2.1 Barone Benedetta

- Lambda expressions
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/view/HomePanel.java#L72-L88
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/view/InstructionsDialog.java#L133
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/view/LevelCompleteDialog.java#L194-L199
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/view/WinDialog.java#L158-170

Codice preso da altri
Gestione movimento e salto dei personaggi, logica ispirata alla classe Player.
Permalink:
https://github.com/axelporras1010/Fireboy_and_Watergirl_Java/blob/master/src/entity/Player.java

3.2.2 Lucarelli Sebastiano

- Lambda expressions
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/view/LevelView.java#L137
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/view/LevelView.java#L80

  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/view/LevelView.java#L317
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/view/GameOverView.java#L108
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/view/GameOverView.java#L113

- Stream
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/test/java/it/unibo/sampleapp/LevelLoaderTest.java#L55

Codice preso da altri
Per la gestione e la struttura degli oggetti ho preso ispirazione da:
Permalink:
https://github.com/axelporras1010/Fireboy_and_Watergirl_Java
Le immagini sono state reperite da:
Permalink:
https://github.com/HyunaJo/FireboyAndWatergirl/tree/main/FireBoyAndWaterGirl/src/static/image

3.2.3 Pedini Emily

- Interfaccia Funzionale ed Espressioni Lambda
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/model/collision/api/Collisions.java#L9-L17
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/model/collision/impl/CollisionFactoryImpl.java#L25-L279

  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/model/game/impl/GameImpl.java#L258
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/view/LevelProcessView.java#L117-L121

- Stream
  - Permalink:
    //github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/2495c4d82c166d90a7cdbf7839d114785d534e5a/src/main/java/it/unibo/sampleapp/model/game/impl/GameImpl.java#L258
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/controller/core/impl/GameEngineImpl.java#L120-L123

- Gestione dei thread
  - Permalink:
    https://github.com/SebastianoLucarelli/OOP24-OppositeElements/blob/master/src/main/java/it/unibo/sampleapp/controller/impl/GameControllerImpl.java

Codice preso da altri
Per la gestione delle collisioni ho preso ispirazione da
Permalink:
https://github.com/emilia-pace/OOP22-D-n-A/blob/master/src/main/java/it/unibo/dna/model/events/impl/EventQueue.java
Permalink:
https://github.com/emilia-pace/OOP22-D-n-A/blob/master/src/main/java/it/unibo/dna/model/events/api/Event.java
Per la realizzazione delle classe Pair
Permalink:
https://bitbucket.org/mviroli/oop2024-esami/src/master/a01a/e1/Pair.java

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Barone Benedetta

All'inizio questo progetto mi incuteva un po' di timore, soprattutto perché non sapevo bene da dove partire e come organizzare il lavoro. Tuttavia, grazie al confronto continuo e alla disponibilità reciproca all'interno del gruppo, siamo riusciti a trovare la giusta direzione.
La collaborazione è stata molto positiva: ci siamo supportati a vicenda, abbiamo condiviso idee e affrontato insieme le difficoltà che si sono presentate. Dopo questa fase iniziale, ho iniziato a sentirmi sempre più a mio agio con il progetto.
Lavorare con i miei compagni è stato davvero stimolante, poiché ognuno ha contribuito con idee e competenze diverse.
Sono molto soddisfatta del risultato finale. Ritengo che questa esperienza mi abbia arricchita, insegnandomi non solo a gestire meglio il mio tempo, ma anche ad ascoltare e confrontarmi in modo costruttivo con gli altri. In generale, la capacità di comunicare e di sostenersi reciprocamente è stato uno dei maggiori punti di forza del gruppo.

4.1.2 Lucarelli Sebastiano

La realizzazione di questo progetto è stata senza dubbio la sfida più grande e complessa che mi sono trovato ad affrontare durante il mio percorso universitario. L'idea di sostenerlo mi ha intimorito fin dalla sua presentazione durante le lezioni e forse è anche questo che mi ha spinto a rimandarlo di così tanto. Nel momento in cui abbiamo iniziato questo lavoro devo però ammettere che mi sono trovato anche affascinato dal pensiero di realizzare

qualcosa di grande, nonostante le difficoltà iniziali legate soprattutto al cercare di capire come impostare il lavoro.
Durante lo sviluppo del progetto mi sono trovato molto bene a lavorare con il mio gruppo, siamo riusciti a dividerci i compiti in maniera efficace rispetto ai tempi e a supportarci a vicenda nei momenti di difficolta. Nonostante io non sia un amante dei progetti di gruppo devo dire che l'esperienza è stata più che positiva portandomi in parte a rivalutare la mia idea su questo tipo di lavori.
Ad oggi mi considero soddisfatto del lavoro svolto in quanto mi ha permesso di approfondire concetti più elevati di programmazione ad oggetti e di poterli mettere in pratica, consentendomi anche di andare a toccare con mano il risultato di tutto quello che ho studiato in questi anni.

4.1.3 Pedini Emily

Questo progetto è stato uno dei più grandi e complessi del corso, infatti inizialmente avevo un pò di timore nell'affrontarlo, sia per la mole di lavoro che per le responsabilità che comportava. Con il tempo però ho iniziato a prendere confidenza anche con i compiti che dovevo svolgere.
Lavorare con i miei compagni di progetto è stata una vera fortuna, infatti, abbiamo mantenuto un costante scambio di idee, organizzando incontri regolari per fare il punto della situazione e confrontarci sulle difficoltà da affrontare. Devo ammettere anche, che se un progetto è di gruppo, mi sento più motivata,produttiva e coinvolta.
Ho avuto la possibilità di dare il mio contributo in maniera significativa e, secondo me, sono riuscita a fare la mia parte al pari degli altri membri del gruppo.
Durante il lavoro ho cercato di dare il meglio di me, penso infatti di aver mostrato determinazione e voglia di imparare. Mi sento,infatti, molto soddisfatta del risultato finale e più sicura nel lavorare in team e credo di aver imparato a coordinarmi meglio con gli altri, valorizzando le idee comuni senza perdere di vista il mio contributo.

4.2 Difficoltà incontrate e commenti per i docenti

Durante il progetto abbiamo incontrato principalmente due difficoltà: la gestione di Git e dei vari branch, che inizialmente ha richiesto un po' di pratica per essere utilizzata correttamente, e la comprensione del template PDF relativo alla relazione, che all'inizio non era del tutto chiaro.

Appendice A

Guida utente

All'avvio del software viene visualizzata la schermata Home, dove è possibile leggere le istruzioni di gioco. Per iniziare a giocare è necessario cliccare il pulsante Start che porta alla schermata di Selezione Livello.
Ogni livello è identificato da una gemma; se la gemma ha bordo colorato, il livello è sbloccato e può essere avviato cliccandola. Durante il gioco, è possibile mettere il livello in pausa, scegliendo se continuare, ricominciare o tornare alla Home per uscire dal gioco. Superato un livello viene mostrata la schermata con gli obiettivi raggiunti insieme a un bottone che riporta alla schermata di Selezione Livello. Da qui è possibile sia proseguire con i livelli successivi, sia visualizzare i progressi già effettuati.
In caso di superamento di tutti i livelli si può uscire dal gioco o tornare alla home per rigiocare i livelli già completati.