Relazione per “Programmazione ad Oggetti”

Across The World

Dellamotta Roberto
Muffato Daniele
Torelli Alessandro
Ujkaj Gledis

A.A 2022/2023

Indice

Capitolo 1

Analisi

Il gruppo si pone come obiettivo quello di realizzare un gioco ispirato a "Crossy Road" (link: https://play.google.com/store/apps/details?id=com.yodo1.crossyroad). La nostra versione del gioco prende il nome di Across The World.

1.1 requisiti

Lo scopo del gioco è quello di tagliare il traguardo o comunque di arrivare il più lontano possibile senza morire. Il giocatore veste i panni di una mascotte e deve cercare di evitare tutti i vari tipi di ostacoli che incontra durante il percorso. Ci sono ostacoli che causano morte immediata, come per esempio fiumi, auto, treni, biciclette ecc.. e altri ostacoli che bloccano il proseguimento costringendo il giocatore a percorrere una strada differente. Compiere passi in avanti aumenterà il punteggio raggiunto del player che indicherà quanto lontano è riuscito ad andare il giocatore all'interno del livello. In giro per la mappa sono disseminate delle monete d'oro che al raggiungimento di una determinata quantità permetteranno di sbloccare nuovi personaggi (skin), oltre alle monete saranno presenti anche dei power up che daranno al giocatore delle agevolazioni temporanee. Le monete raccolte e le skin acquistate verranno salvate solamente col bottone “save and exit”, altri metodi di uscita forzata non permetteranno il salvataggio delle statistiche di gioco.

Requisiti funzionali

- Il giocatore si potrà muovere all’interno della mappa di gioco (che cambia in base alla difficoltà), senza uscire dai bordi
- La partita dovrà terminare quando il giocatore raggiunge il traguardo o collide con un’entità nemica
- Il power up raccolto (distinguibile dall’icona) avrà una durata temporanea
- Le monete raccolte verranno salvate al termine della partita
- Le varie entità nemiche avranno velocità diverse in base al loro tipo
- Alla morte del giocatore verrà visualizzato un menu con la possibilità di scelta tra: riiniziare la partita, tornare al menu o uscire dall’applicazione

Requisiti non funzionali

- Dal menu si può accedere allo shop e acquistare skin (impatto solamente estetico) con le monete raccolte
- Le informazioni relative alle monete raccolte, allo score raggiunto e ai power up attualmente attivi vengono visualizzate in alto a sinistra durante la partita per non essere troppo invasivi ma allo stesso tempo facili da leggere

1.2 Analisi e modello del dominio

Il gioco è strutturato in tre difficoltà, ogni modalità ha una mappa corrispondente dove le varie entità presenti potranno interagire con l'entità player comandata dal giocatore. Durante lo svolgimento della partita il giocatore incontrerà alcune entità che dovrà evitare per non morire cercando di avanzare quando il percorso è libero o utilizzare delle piattaforme per attraversare zone che gli causerebbero la morte, altre entità (i power up) invece conferiscono al giocatore delle agevolazioni temporanee per il completamento della mappa come l'immortalità, bonus sulla raccolta delle monete come un moltiplicatore di valore della moneta raccolta o la calamita che permette di raccogliere monete che stanno attorno al player ma non nella sua stessa posizione. Una volta raggiunto il traguardo verrà visualizzata una schermata di vittoria mentre se si colliderà con un ostacolo che causa la morte del player verrà visualizzata una schermata di game over, le monete ottenute vengono collezionate sia in caso di vittoria che in caso di sconfitta.

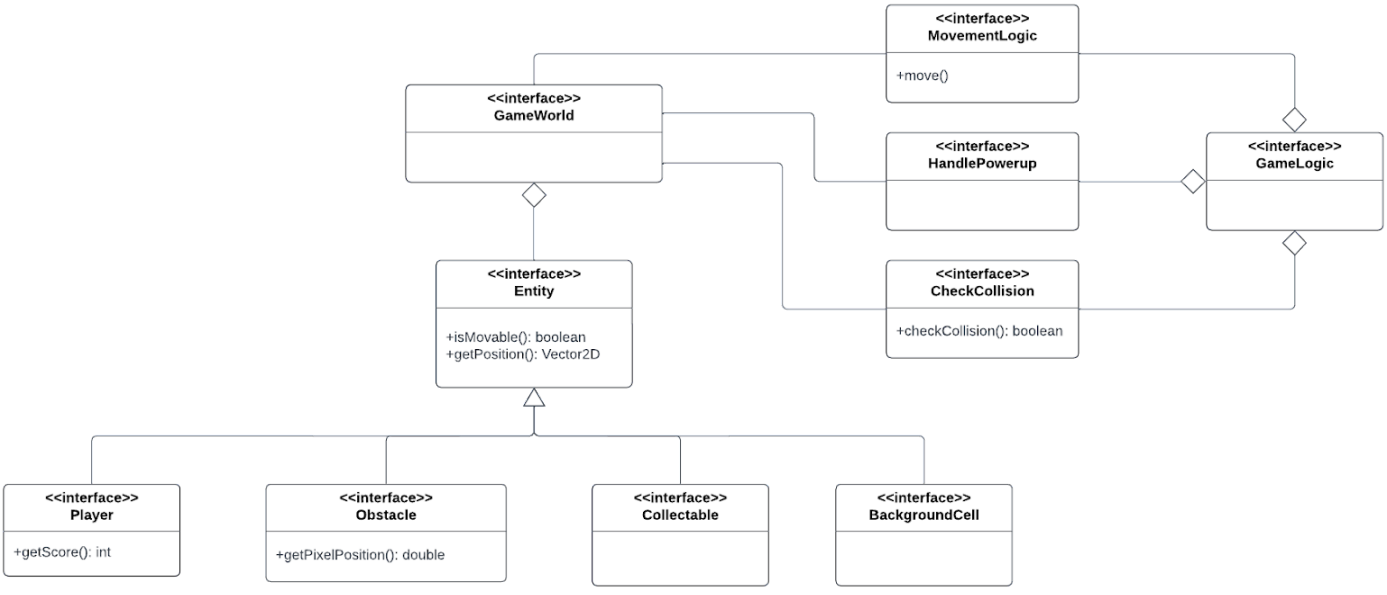

Immagine 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali e i rapporti fra loro

Capitolo 2

Design

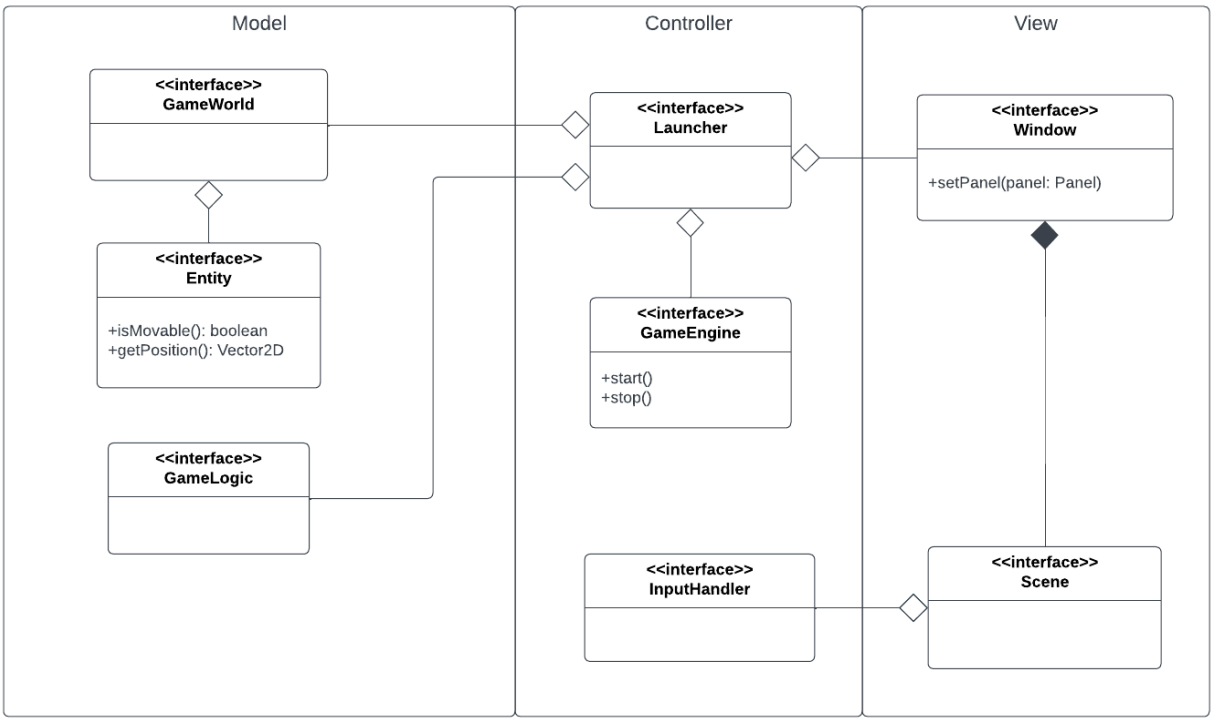

Immagine 2.1 Schema UML - Architettura MVC

2.1 Architettura

L'architettura di across the world è stata impostata utilizzando il pattern architetturale MVC (Model View Controller). Questa scelta permette di suddividere in tre macro componenti l'applicazione, dove la componente di view si occupa della rappresentazione grafica e viene svolta dalle interfacce Scene e Window, la componente di model di contenere le informazioni riguardanti l'applicazione e i relativi metodi di modellazione tramite le interfacce Entity, GameWorld, GameLogic e infine la parte di controller che mette in comunicazione le due componenti sopra citate utilizzando le interfacce Launcher, Loader e GameEngine.

2.2 Design dettagliato

2.2.1 Dellamotta Roberto

Il mio compito all'interno del progetto riguarda lo sviluppo delle seguenti interfacce:

- Entity
- Collectable
- Player
- Obstacle
- Background Cell

con i relativi entity type.

Dato che le entità condividono metodi comuni ho creato un'interfaccia generica entità, la quale verrà estesa da ogni specifico tipo di entità aggiungendo i metodi necessari.

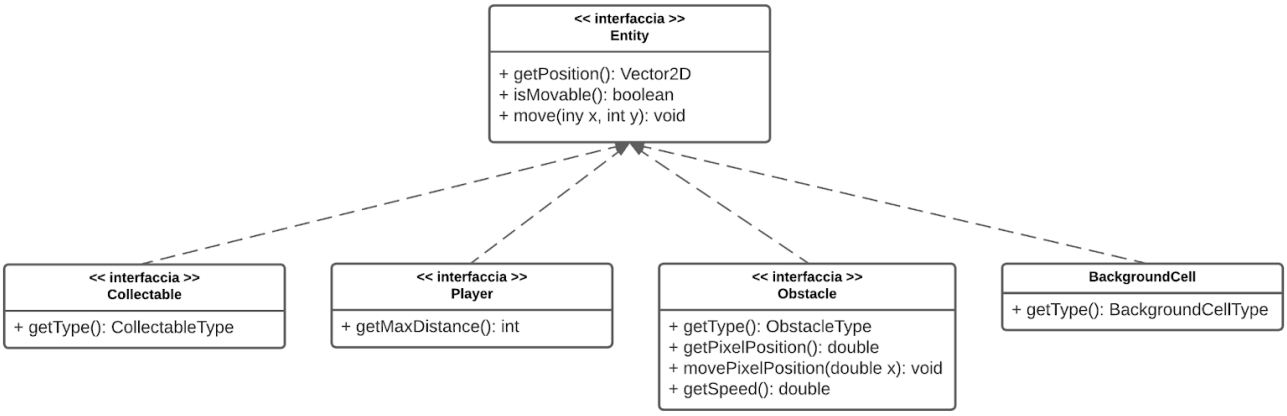

Immagine 2.2: schema uml delle interfacce entity.

Per gestire lo score del player ho creato un metodo che controlla la distanza massima raggiunta dal personaggio all'interno della mappa, ogni volta che il player esegue un movimento verrà controllata la massima posizione y raggiunta dal personaggio, se la nuova y è maggiore verrà salvata e visualizzata durante la partita in alto a sinistra incrementando lo score.
Per la gestione degli ostacoli si è creato un enum, nel quale sono descritti tutti i tipi di ostacoli statici e dinamici, i quali avranno diverse proprietà come la velocità di movimento (se sono statici il valore sarà settato su 0) o il fatto che siano camminabili (come tronchi, rocce e materassini) o meno.
Per i collectable invece si è fatta distinzione fra i power up e le monete, per organizzare la gestione del salvataggio di quest'ultime.

Mi sono occupato inoltre della creazione delle diverse mappe di gioco e delle skin di qualsiasi entità (player, sfondo, ostacoli..).
Per come abbiamo organizzato il progetto non richiede ulteriore lavoro l'aggiunta di nuovi ostacoli, per questo ho ampliato l'idea iniziale dei treni, macchine e tronchi con ulteriori ostacoli come jet, mostri, biciclette, palloni.. in base al tipo di difficoltà, in modo tale da avere diversi scenari di gioco.

Problema: come gestire il salvataggio delle mappe?
Soluzione: le mappe sono state salvate su file di testo, memorizzando per ciascuna entità, utilizzando il nome dei vari entity type, la lista delle coordinate delle celle in cui è situata tale entità; nella classe AbstractLoader verranno associati i vari file .png contenenti le immagini di gioco ai relativi entity type e all'avviamento della scena di gioco essi verranno caricati nelle relative coordinate.

Per determinare la vittoria ci sarà un BackgroundCell speciale con la bandiera del traguardo e appena attraversata comparirà la win scen.

2.2.2 Muffato Daniele

Il mio ruolo nel progetto è stato relativo all'implementazione delle seguenti classi:

- Launcher
- GameEngine
- Loader

Il Launcher è la classe principale del progetto, e si occupa di tenere traccia di tutte le classi necessarie per far funzionare l'applicazione. Viene utilizzato il pattern del singleton per semplificare le chiamate ai metodi da classi che ne dovessero aver bisogno, e per assicurare la presenza di un singolo oggetto launcher nell'applicazione. Si occupa, inoltre, di fornire un'interfaccia per la comunicazione tra le varie componenti dell'architettura.

Il GameEngine si occupa di gestire il motore di gioco, con un metodo per farlo iniziare, e un controllo eseguito ad ogni iterazione per interrompere lo stesso qualora la scena non sia più quella di gioco.

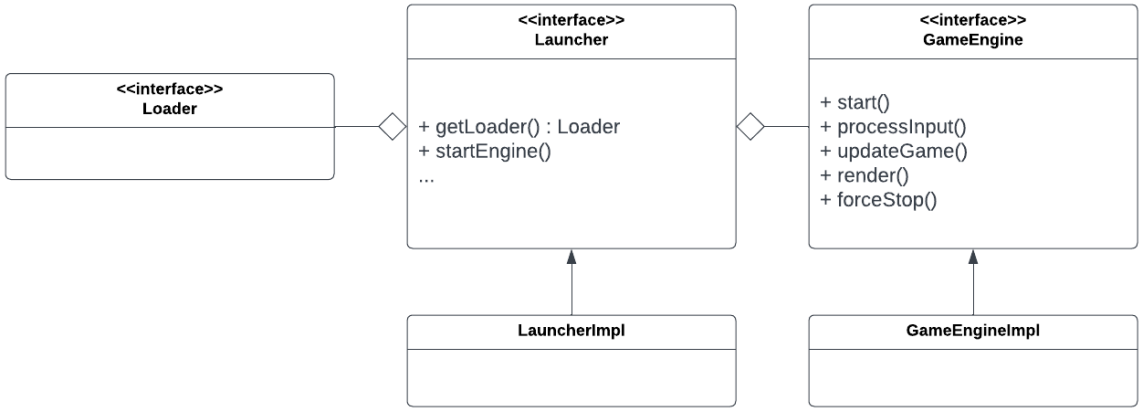

Immagine 2.3: schema uml delle classi del Loader, Launcher, GameEngine.

Il Loader gestisce il caricamento delle risorse, la loro elaborazione e conseguente conversione in oggetti, messi successivamente a disposizione alle altre classi. Le risorse da gestire sono state suddivise nelle tre categorie: immagini, mappe, statistiche. A ciascuna categoria è stata riservata una directory nella sezione delle risorse, con un file allegato contenente le istruzioni per aggiungere e/o modificare le suddette risorse.
Infine è stato necessario suddividere l'implementazione del Loader in varie classi astratte con ruoli differenti, per rispettare il Single Responsibility Principle.

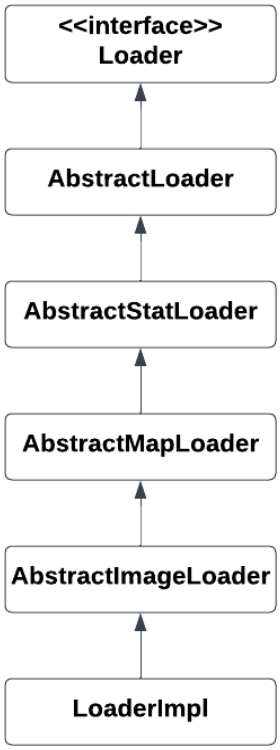

Immagine 2.4: schema uml della suddivisione del Loader in varie classi astratte.

Problema: il file delle statistiche è salvato nella home directory dell'utente, e può quindi non esserci o avere degli errori (l'utente potrebbe modificare o rimuovere il file). In tal caso l'applicazione non funzionerebbe, visto che il loader fallirebbe ad ottenere le statistiche
Soluzione: creare un file delle statistiche che viene letto solo se non è presente il file nella home directory, o se ha problemi, e nel secondo caso viene cancellato il file, resettando i progressi, ma assicurando il corretto funzionamento dell'applicazione.
Inoltre è stato aggiunta

2.2.3 Torelli Alessandro

Il mio compito all'interno del gruppo comprendeva lo svolgimento di:

- CheckCollision
- HandlePowerup
- MovementLogic
- GameStat

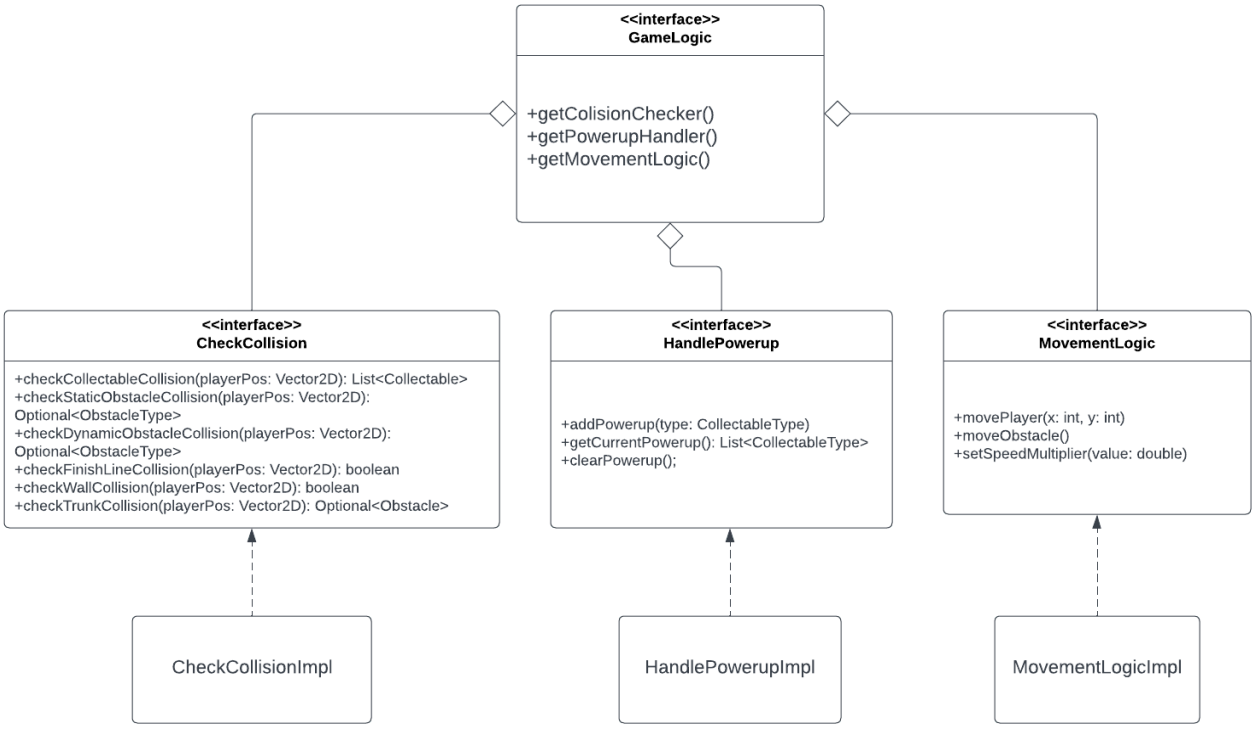

Immagine 2.5 Schema UML GameLogic e classi contenute

Per la parte di controllo delle collisioni ho creato un metodo per ogni collisione che era importante rilevare distintamente. Organizzando il gioco in celle ogni metodo va a controllare la posizione (tradotta in cella) dell'entità di cui si vuole verificare la collisione col player comparando quindi entrambe le posizioni e se appartengono alla stessa cella viene restituito il tipo, l'entità o una lista in base allo scopo del metodo.

Nella parte riguardante i power up bisognava trovare un modo per gestire in maniera temporizzata ogni singolo powerup e quando richiesto fornire una lista di tutti i tipi di potenziamenti attivi al momento. La gestione temporizzata è stata sviluppata utilizzando un “timer” che dopo il tempo prestabilito rimuove dalla lista dei power up quello in testa che risulterà essere il primo inserito.
Problema: durante lo sviluppo dell’applicazione ci siamo poi accorti che utilizzando la funzione di restart se si aveva ancora attivo un powerup dalla partita precedente e nella partita successiva si riusciva a prendere un nuovo power up prima che fosse passato il tempo di scadenza del “vecchio” power up, quello “nuovo” veniva rimosso allo scadere del timer di quello “vecchio” (quindi sarebbe durato meno tempo rispetto al tempo normale di durata del power up).
Soluzione: per risolvere questo problema è stato messo un identificativo che riguardasse la partita corrente in modo tale da non far togliere al thread del timer del power up della partita precedente quello della partita corrente

All’interno della MovementLogic viene controllata l’effettiva possibilità del player di poter essere mosso nella direzione in cui vorrebbe muoversi. In caso di collisione con entità viene riconosciuto l’altro oggetto restituito dalla collisione e viene gestita la relativa azione, mentre per quanto riguarda il movimento degli ostacoli viene controllata dopo l’effettivo spostamento se la loro posizione coincide con quella del player e in caso terminare il gioco poiché sta a significare che il player è stato colpito da un ostacolo, senza quest’ultimo controllo si sarebbe verificato che il player stando fermo avrebbe avuto la possibilità di essere immune alle collisioni (sarebbero state controllate solo quando decideva di muoversi).

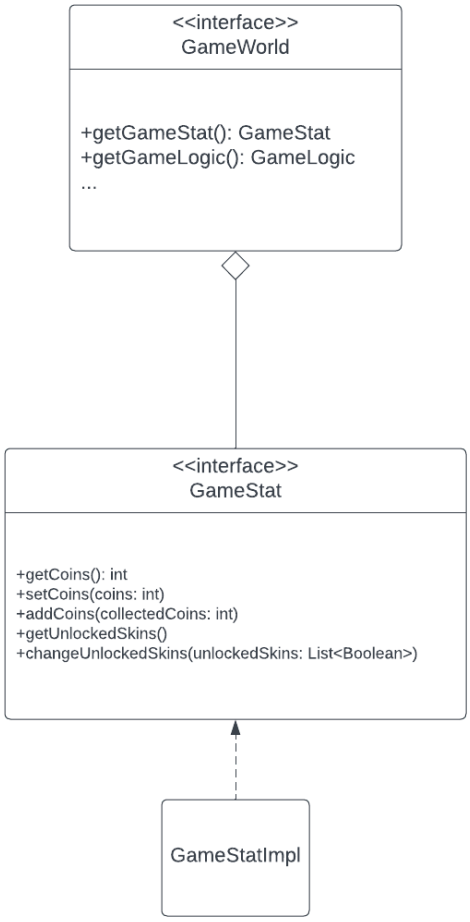

Immagine 2.6 Schema UML GameStat

Per quanto riguarda le statistiche del gioco ho pensato alle possibili informazioni che potevano servirci riguardanti le skin e le monete collezionate dal player. E' stata quindi creata una classe che potesse contenere e gestire tutte queste informazioni, accessibili tramite GameWorld poiché ha senso avere e gestire queste informazioni solo una volta che è stata creato il "mondo di gioco" con tutte le entità del gioco.

2.2.4 Ujkaj Gledis

All'interno del gruppo, il mio ruolo prevedeva lo sviluppo delle seguenti attività:

- Scene di gioco;
- Input Handler delle scene.

Le scene sviluppate sono MenuScene, ShopScene, VictoryScene e GameOverScene con i relativi input.

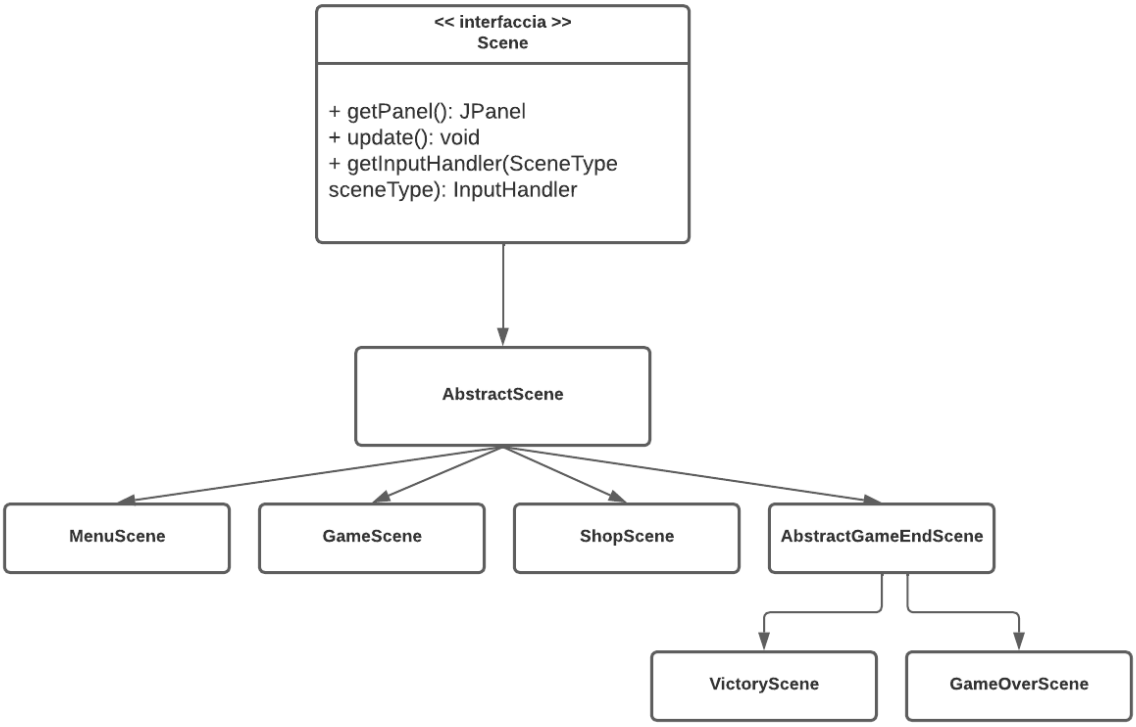

Immagine 2.7: Schema UML di Scene, rappresenta il rapporto con le varie scene sviluppate.

La MenuScene si occupa di creare e gestire la scena del menu del gioco, fornendo un'interfaccia che permette all'utente di avviare il gioco, accedere al negozio, eliminare il progresso e uscire dall'applicazione. Inoltre, fornisce la possibilità di selezionare la difficoltà di gioco tramite una casella combinata.

La ShopScene configura gli elementi grafici e definisce metodi per gestire l'acquisto delle skin. Viene controllato se il giocatore ha abbastanza monete per acquistare una skin e viene mostrato un messaggio appropriato se non ci sono abbastanza monete o se tutte le skin sono state già acquistate.

Problema: La VictoryScene e la GameOverScene erano identiche e gradlew check dava errore per la ripetizione di codice.
Soluzione: Utilizzo della AbstractGameEndScene, classe astratta che può essere estesa per creare scene di fine gioco specifiche, personalizzando il titolo e le azioni dei pulsanti.

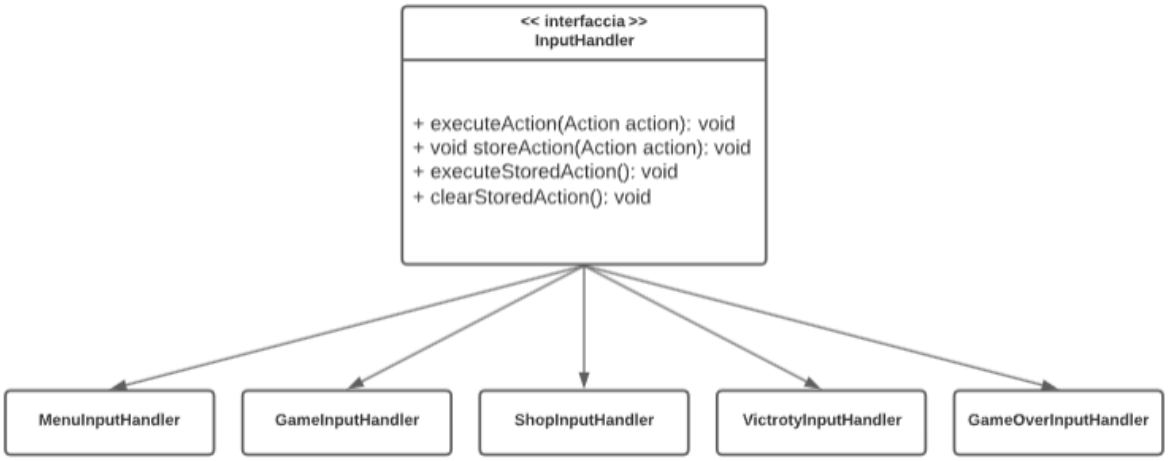

Immagine 2.8: Schema UML di InputHandler, rappresenta il rapporto con gli input delle scene.

Problema: Molti Input Handler avevano in comune gli input necessari.
Soluzione: Inserire tutte le azioni in SharedInputHandler e richiamando il metodo executeAction() viene eseguita l'effettiva azione richiesta.
Esempio: Voglio uscire cliccando 'Save and exit' dal menù principale. Di conseguenza l'azione sarà 'EXIT_APP', dal singleton verrà chiamato il metodo 'saveAndCloseWindow()' che salverà i dati e chiuderà l'applicazione.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

I test che sono stati effettuati sono:

- LauncherTest: controlla che i get non ritornino valori nulli dopo che sono stati inizializzati tramite il set e che i vari metodi funzionino correttamente.
- LoaderTest: classe che controlla il corretto caricamento delle risorse di gioco, che tutti i metodi “get” non ritornino un valore nullo e che tutti i progressi vengano salvati correttamente.
- GameEngineTest: classe che controlla che l’engine sia in grado di essere eseguito senza provocare crash.
- CheckCollisionTest: questa classe controlla che tutti i tipi di collisioni avvengano correttamente, in base al diverso tipo di ostacolo o oggetti con cui il player collide, come ad esempio le monete, ostacoli statici e dinamici.
- EntityTest: questa classe viene utilizzata per controllare che le entità statiche non si possano effettivamente muovere e quelle dinamiche si.

- PlayerTest: questa classe controlla che il player si muova correttamente nella mappa di gioco e che restituisca la posizione massima raggiunta dal player nella mappa durante la partita.
- InputHandlerTest: verifica che non ci siano errori durante il cambiamento di scena da parte dell'applicazione.
- SceneTest: controlla che i get non ritornino valori nulli e che la window sia in grado di settare la scena.
- WindowTest: classe che verifica il corretto "set" della scena senza provocare crash.

3.2 Metodologie di lavoro

3.2.1 Dellamotta Roberto

Il mio lavoro è orientato alla parte di model in MVC dove ho implementato le varie entità di gioco e i relativi type, inoltre mi sono occupato delle mappe di gioco e delle texture di gioco.
Riassumendo mi sono occupato di:

- Implementazione di Player (package it.unibo.project.game.model.impl)
- Implementazione di BackgroundCell (package it.unibo.project.game.model.impl)
- Implementazione di Entity (package it.unibo.project.game.model.impl)
- Implementazione di Obstacle (package it.unibo.project.game.model.impl)
- Implementazione dei vari EntityType (package it.unibo.project.game.model.impl)
- Creazione delle texture (package it.unibo.project.sprite)
- Creazione delle mappe (package it.unibo.project.maps)

3.2.2 Muffato Daniele

Il mio lavoro è stato in gran parte relativo alla parte di controller in MVC, in particolare mi sono occupato di:

- implementazione di RandomizeLine (package it.unibo.project.utility)
- implementazione di LauncherImpl (package it.unibo.project.controller.core.impl)
- implementazione di AbstractStatLoader (package it.unibo.project.controller.core.impl)
- implementazione di AbstractImageLoader (package it.unibo.project.controller.core.impl)
- implementazione di AbstractMapLoader (package it.unibo.project.controller.core.impl)
- implementazione di AbstractLoader (package it.unibo.project.controller.core.impl)
- implementazione di LoaderImpl (package it.unibo.project.controller.core.impl)
- implementazione di GameEngineImpl (package it.unibo.project.controller.engine.impl)

3.2.3 Torelli Alessandro

La maggior parte del mio lavoro si è svolta nella parte di model in MVC cercando di svolgere le mie parti rendendole più riutilizzabili possibile, in modo da non dover cambiare la parte di model in caso di aggiunta di nuovi tipi di skin o ostacoli. Ho creato a inizio progetto le classi di utility Vector2D e Pair per evitare ripetizioni di codice e fornire ai miei compagni le classi da poter usare nel loro lavoro. Mi sono occupato quindi di:

- Implementazione di CheckCollision (package it.unibo.project.game.logic.impl)
- Implementazione di HandlePowerup (package it.unibo.project.game.logic.impl)
- Implementazione di MovementLogic (package it.unibo.project.game.logic.impl)
- Implementazione di GameStat (package it.unibo.project.game.model.impl)
- Implementazione di Vector2D (package it.unibo.project.utility)
- Implementazione di Pair (package it.unibo.project.utility)
- Creazione mappa relativa alla difficoltà facile (package it.unibo.project.maps)

3.2.4 Ujkaj Gledis

Mi sono occupato di:

- Implementazione di MenuScene (package it.unibo.project.view.impl);
- Implementazione di AbstractGameEndScene (package it.unibo.project.view.impl);
- Implementazione di GameOverScene (package it.unibo.project.view.impl);
- Implementazione di VictoryScene (package it.unibo.project.view.impl);
- Implementazione di ShopScene (package it.unibo.project.view.impl);
- Implementazione di tutti gli input delle scene (gestite unicamente poi da SharedInputHandler) (package it.unibo.project.input.impl).

3.3 Note di sviluppo

3.3.1 Dellamotta Roberto

- Lambda Expression
- Stream
- Swing

3.3.2 Muffato Daniele

- Lambda Expression
- Stream
- Swing

3.3.3 Torelli Alessandro

- Lambda Expression
- Stream
- Swing

3.3.4 Ujkaj Gledis

- Lambda Expression;
- Stream
- Swing

Capitolo 4

4.1 Autovalutazione e lavori futuri

4.1.1 Dellamotta Roberto

Penso che un progetto come questo mi abbia aiutato nell'ambito della collaborazione insieme ad altre persone, ritengo inoltre che la scelta libera del tema da realizzare sia stato un incentivo in più per lo sviluppo.
Durante il progetto ho contribuito con mie idee e opinioni riguardo lo sviluppo, confrontandomi con i miei compagni per ottenere un risultato migliore.
Man mano che il progetto avanzava ero soddisfatto del risultato che stavamo ottenendo e a parer mio ora, ad applicazione terminata sono molto contento anche perchè non avevo mai sviluppato una cosa simile.

4.1.2 Muffato Daniele

Sono abbastanza soddisfatto per come siamo riusciti a sviluppare il progetto, considerando che nessuno di noi avesse priori esperienza. Noto però anche che, con il senno di poi, ci sarebbero state molte migliorie possibili per rendere il progetto più facilmente estendibile. Inoltre lavorare in gruppo ha rappresentato un'ulteriore difficoltà, per quanto abbia permesso di velocizzare lo svolgimento del progetto.

4.1.3 Torelli Alessandro

Questo progetto mi ha entusiasmato molto, fin da quando è stata accettata l'idea della tipologia di gioco dai miei compagni, abbiamo impiegato diversi incontri per valutare e decidere bene come suddividere il progetto per renderlo più semplice e scrivere poi codice il più riutilizzabile possibile. Ho contribuito alla creazione del progetto con idee e/o soluzioni a problemi e vedere che giorno dopo giorno il progetto prendeva sempre più forma mi ha dato molte soddisfazioni anche perchè non avevo mai sviluppato prima d'ora un'applicazione del genere e per di più in gruppo. Devo ammettere che non ero entusiasta prima di iniziare il progetto di dover lavorare in gruppo perchè solitamente preferisco lavorare da solo ma mi sono ricreduto e questo mi porterà a essere meno prevenuto su lavori di gruppo futuri.

4.1.4 Ujkaj Gledis

Ritengo il progetto complessivamente ben fatto, riuscendo a raggiungere gli obiettivi prefissati. Tuttavia, in totale onestà il mio impegno personale non è stato esaustivo come avrei voluto causa impegni lavorativi, personali e accademici. Ho contribuito nella fase iniziale del progetto fornendo i miei punti di vista, le mie idee e i miei suggerimenti. Ho lavorato nelle mie parti ottenendo un discreto risultato ma con un certo impegno

durante la stesura del codice. Col senno di poi affronterei diversamente un progetto del genere, sono consapevole dei punti in cui avrei potuto fare di più e sicuramente mi impegnerò nel migliorare la gestione del tempo e le responsabilità in progetti futuri.

Appendice A

Guida utente

All'inizio dell'applicazione verrà visualizzato il menù principale, che conterrà diversi bottoni per rappresentare le seguenti opzioni:

- START: avvia la partita nella difficoltà di gioco selezionata nel bottone sottostante, che prevede le modalità "easy, normal, hard";
- SHOP: accede al negozio dove si possono acquistare le skin;
- RESET: chiude l'applicazione, cancellando tutti i salvataggi di gioco (monete e skin);
- SAVE & EXIT: salva i progressi prima di chiudere l'applicazione.

I comandi principali dell'applicazione sono:

- ESC (Escape): chiude l'applicazione durante la partita salvando i progressi di gioco (come il tasto nel menu principale);
- W / ↑ (Up Arrow): movimento verso l'alto;
- S / ↓ (Down arrow): movimento verso il basso;
- A / ← (Left arrow): movimento verso sinistra;
- D / → (Right arrow): movimento verso destra.

Nella pagine successive sono riportate alcune viste dell'applicazione.

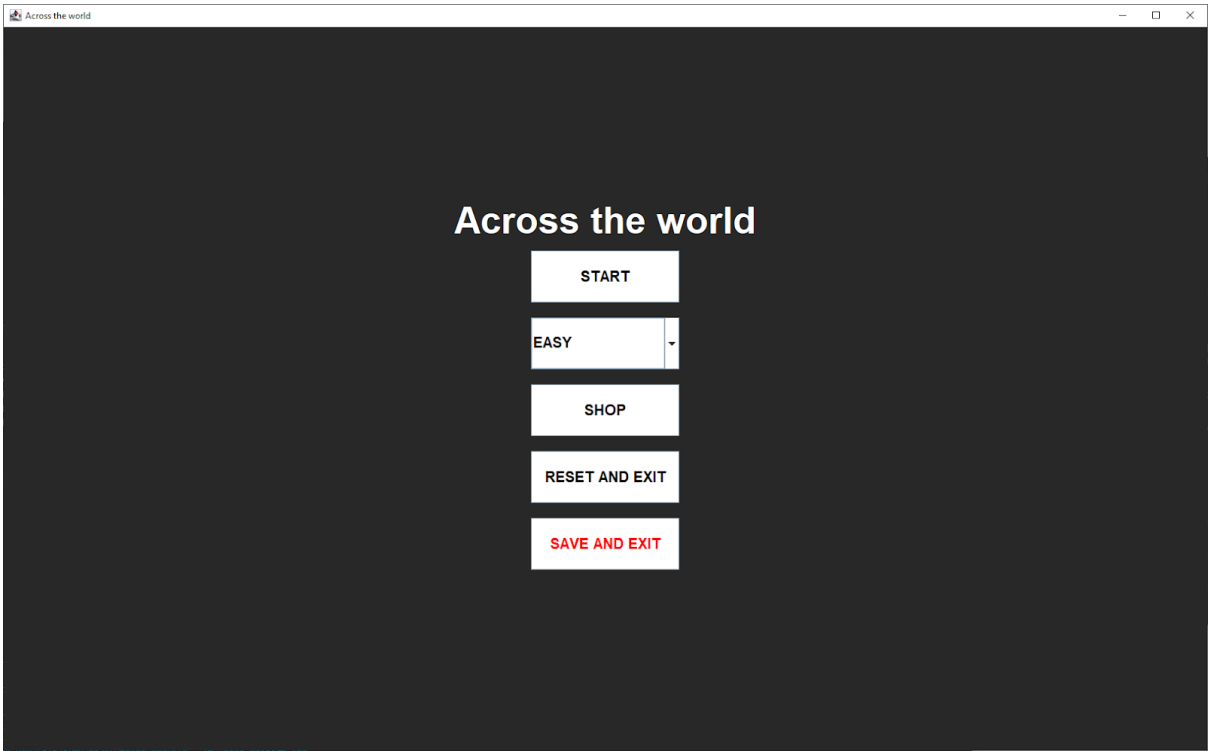

Immagine A.1: Menu principale

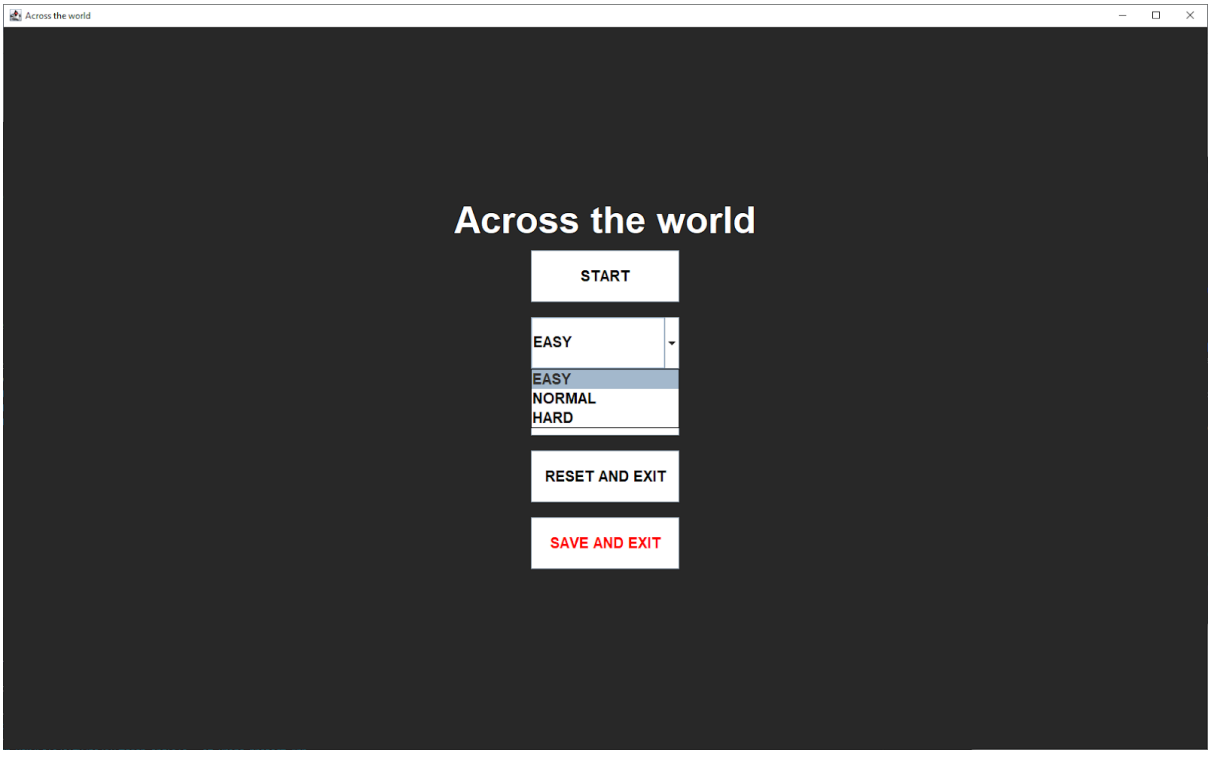

Immagine A.2: Selettore delle modalità

Immagine A.3: Shop

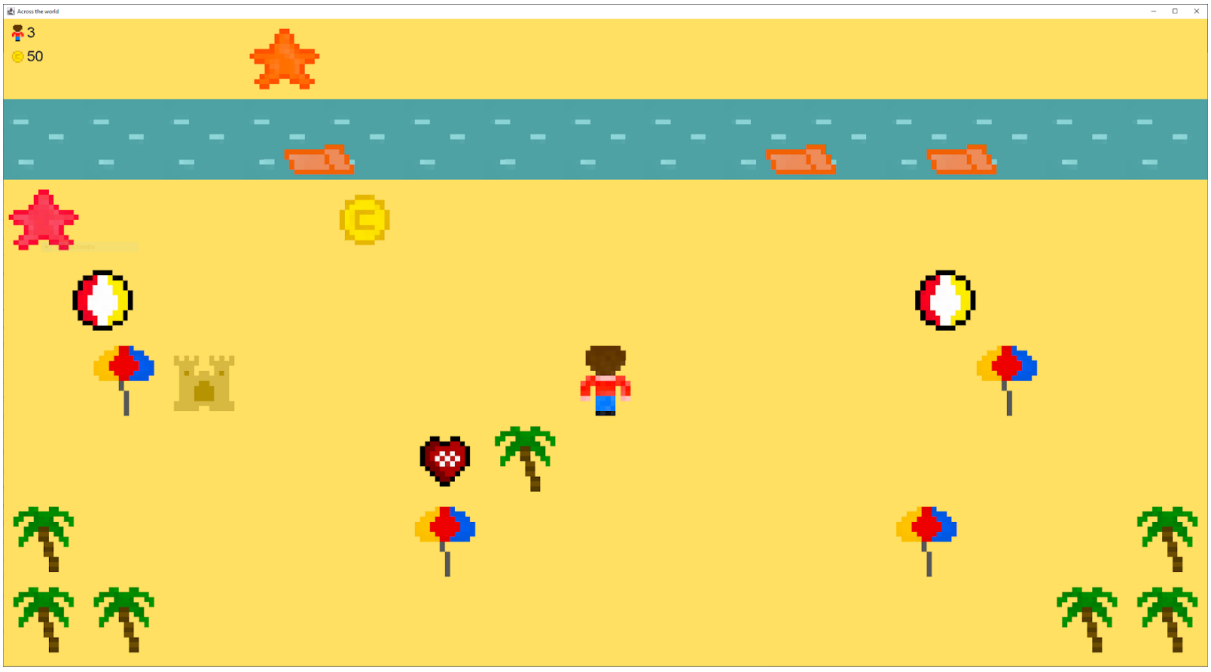

Immagine A.4: Mappa easy

Immagine A.5: Mappa normal

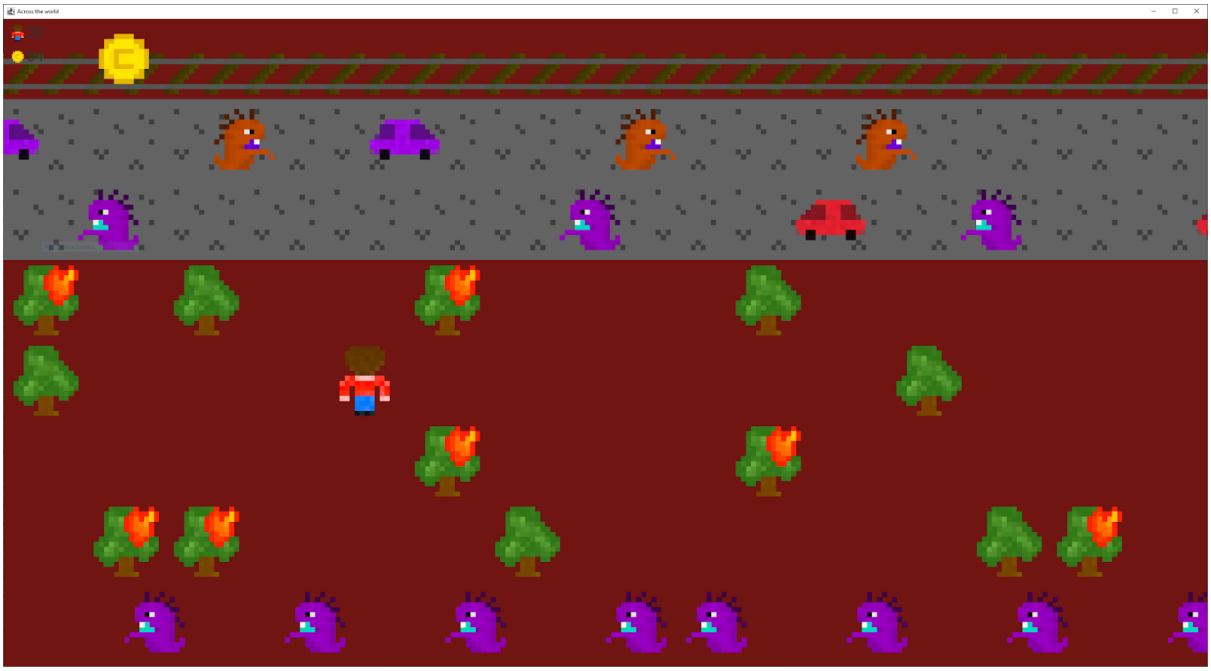

Immagine A.6: Mappa hard

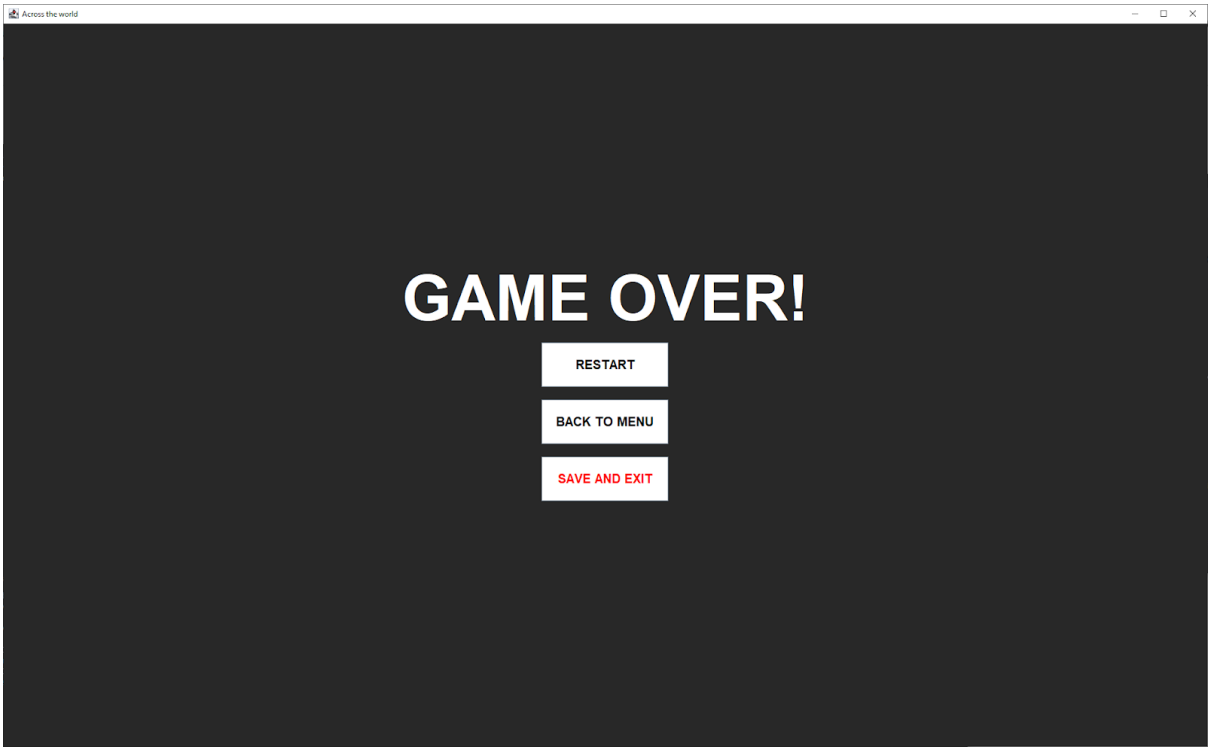

Immagine A.7: Game over scene

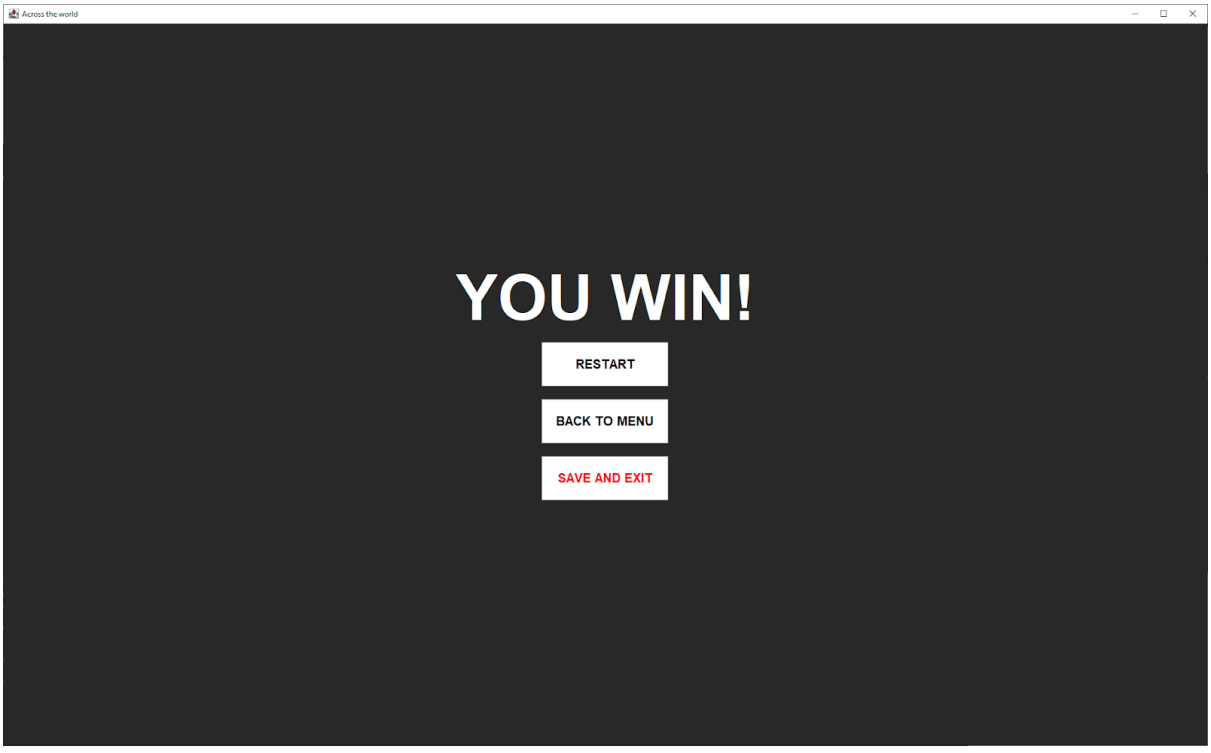

Immagine A.8: Win scene