RELAZIONE PROGETTO MORPHEUS

PROGRAMMAZIONE OGGETTI

MENGOZZI LUCA

NIGRO MATTEO

RICIPUTI JACOPO

Indice

Capitolo 1
Analisi

Il software mira alla realizzazione di un gioco del tipo endless runner. Con lo scopo di totalizzare il maggior punteggio evitando ostacoli e mostri, con la possibilità di sparare proiettili e di saltare per eludere i nemici.

1.1 Requisiti

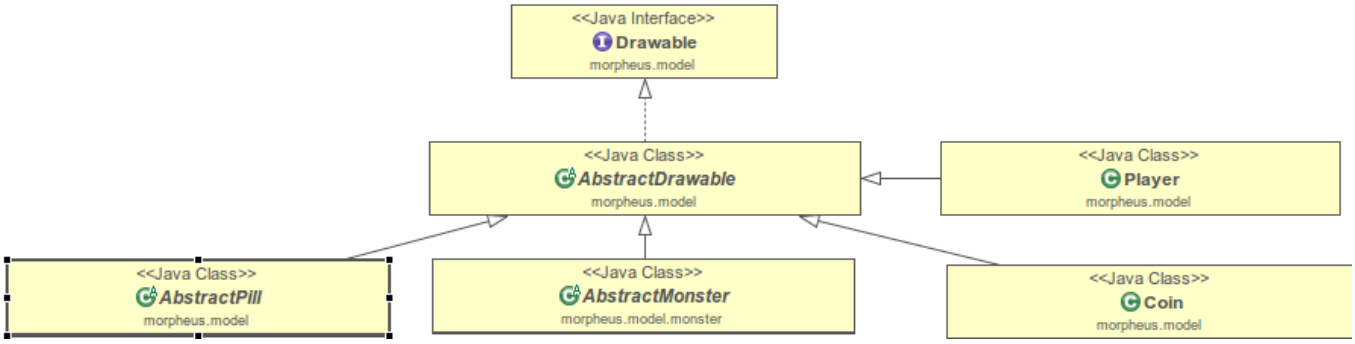
L’applicazione deve permettere la creazione di un mondo di gioco “infinito”. Con avanzamento del personaggio e della telecamera a sua volta. Con possibilità di morte del personaggio nel caso di collisioni con mostri o di caduta fuori dal mondo di gioco.

Il gioco dovrà essere capace di:

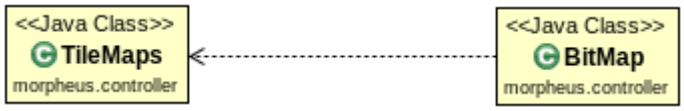
- Gestire le collisioni fra le entità che vi partecipano.
- Creare un mondo di gioco che si protenda all’infinito e che permetta una diversificazione del level design tra una partita e l’altra, in modo da ridurre la ripetitività e indurre il giocatore a giocare più volte.
- Realizzare una classifica con i migliori punteggi.

1.2 Analisi e modello del dominio

Le entità all’interno dell’applicazione devono poter essere renderizzate sullo schermo, perciò le entità a schermo derivano dall’interfaccia Drawable contente utilità per gestirle.



La generazione del mondo di gioco avviene tramite una pool di mappe preimpostate le quali vengono prelevate in maniera randomica durante la partita creando un livello “infinito” e generato proceduralmente sulla base delle suddette tile maps.



Poi si ha inoltre una gestione di cosiddetti State, la cui funzione è quella di dividere il programma in più parti per una più corretta gestione, ma di questo se ne parlerà in seguito.

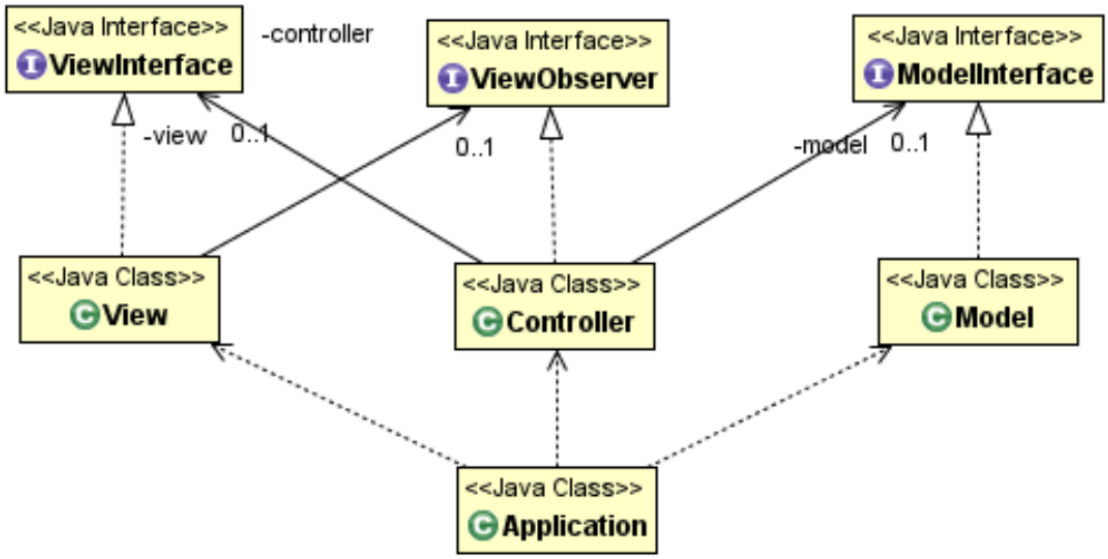
Tra le funzionalità opzionali erano state inserite la possibilità di potenziamenti vari e la diversificazione dell’ambiente di gioco che non sono state implementate nell’applicazione ma che con buona probabilità verranno risolti in futuro.

Capitolo 2

Design

2.1 Architettura

Per la realizzazione del software abbiamo optato per l'architettura MVC (Model-View-Controller).



A nostro malgrado non siamo riusciti a implementare al meglio il pattern di progettazione previsto.
Dal lato Model il pattern MVC è stato implementato tramite l'utilizzo di un'interfaccia Model e di una classe ModelImpl che permette sia a View che a Controller di prelevare i dati messi a disposizione dal Model.

La classe principae Morpheus che contiene il metodo Main ha anche il compito di inizializzare il thread principale e le tre componenti.

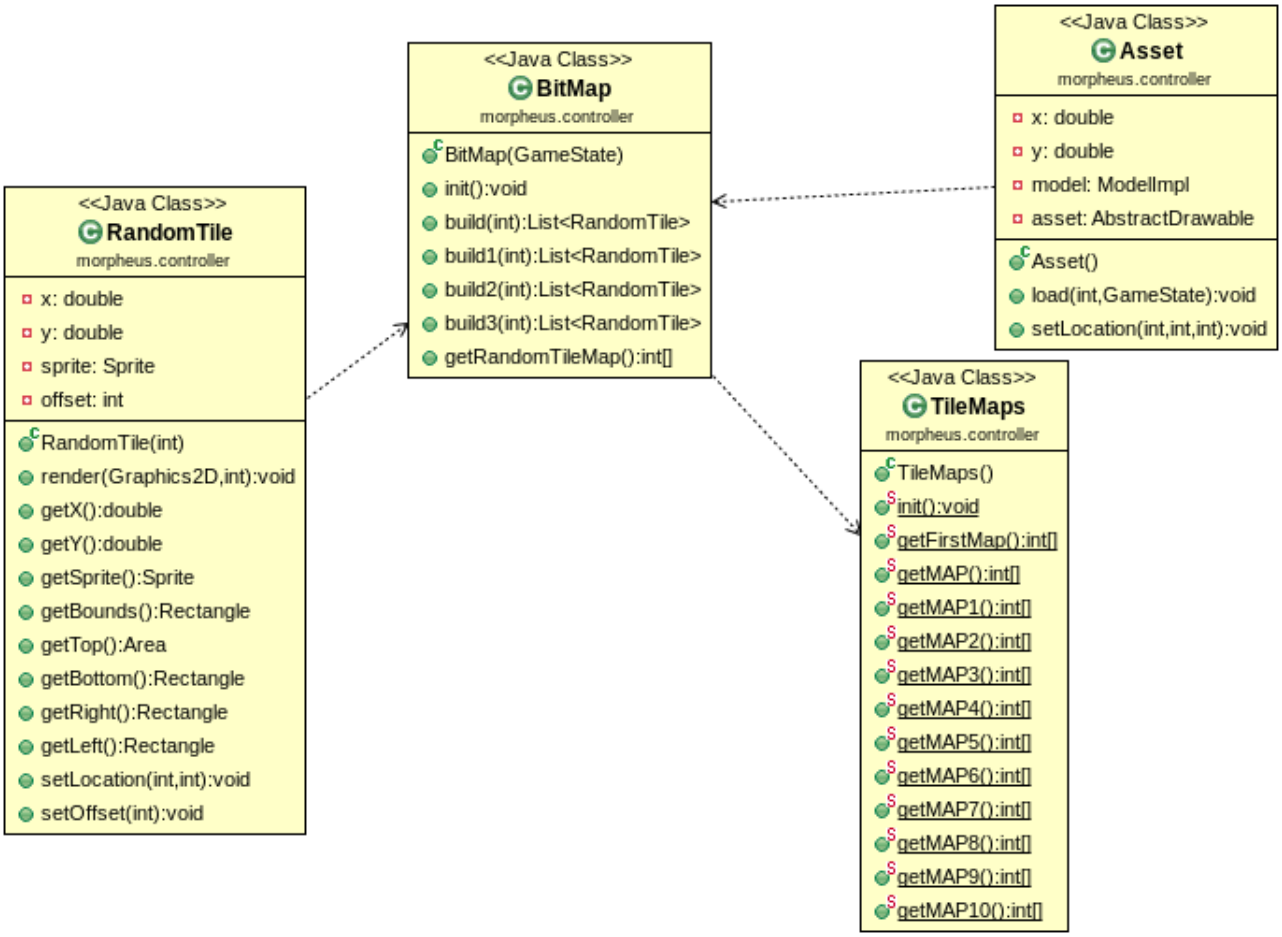
2.2 Design dettagliato

Le classi che contengono i sorgenti sono state suddivise in tre package principali:
- morpheus.model
- morpheus.view
- morpheus.controller

Di cui il primo package contiene a sua volta altri tre package (morpheus.model.exception, morpheus.model.monster, morpheus.model.test) ed il secondo suddiviso in un sottopackage (morpheus.view.state).

Nigro - Controller

Il Controller si occupa di tutte quelle operazioni logiche che l'applicazione sarà in grado di svolgere.
La parte più densa di contenuti è sicuramente il sistema di generazione procedurale ed "infinito" del mondo di gioco.



L’algoritmo di gerazione procedurale proposto si avvale di una lista di mappe predefinite dichiarate ed inizializzate nella classe TileMaps la quale tramite dei getter le fornisce alla classe BitMap, quest’ultima tramite calcoli che portano la viewport del gioco ad essere divisa in blocchi astratti di 64x64 (dandogli una struttura a griglia) permettono il posizionamento di assets all’interno del mondo di gioco creando a tutti gli effetti un level design tutto sommato accurato.
La funzione delle classi Asset e RandomTile consiste nel fornire elementi da aggiungere al gioco provenienti dal Model tramite un comodo costrutto switch che differenzia i vari tipi di assets possibili. Il sistema di aggiunta di elementi alla viewport è stato pensato per essere il più intuitivo possibile, questo obiettivo viene raggiungo tramite le TileMaps stesse le quali consistono in array monodimensionali con valori varianti in base allo switch case sopracitato presente nelle classi RandomTile e Asset; tale sistema permette di aggiungere assets al mondo di gioco semplicemente modificando i valori all’interno dell’array.

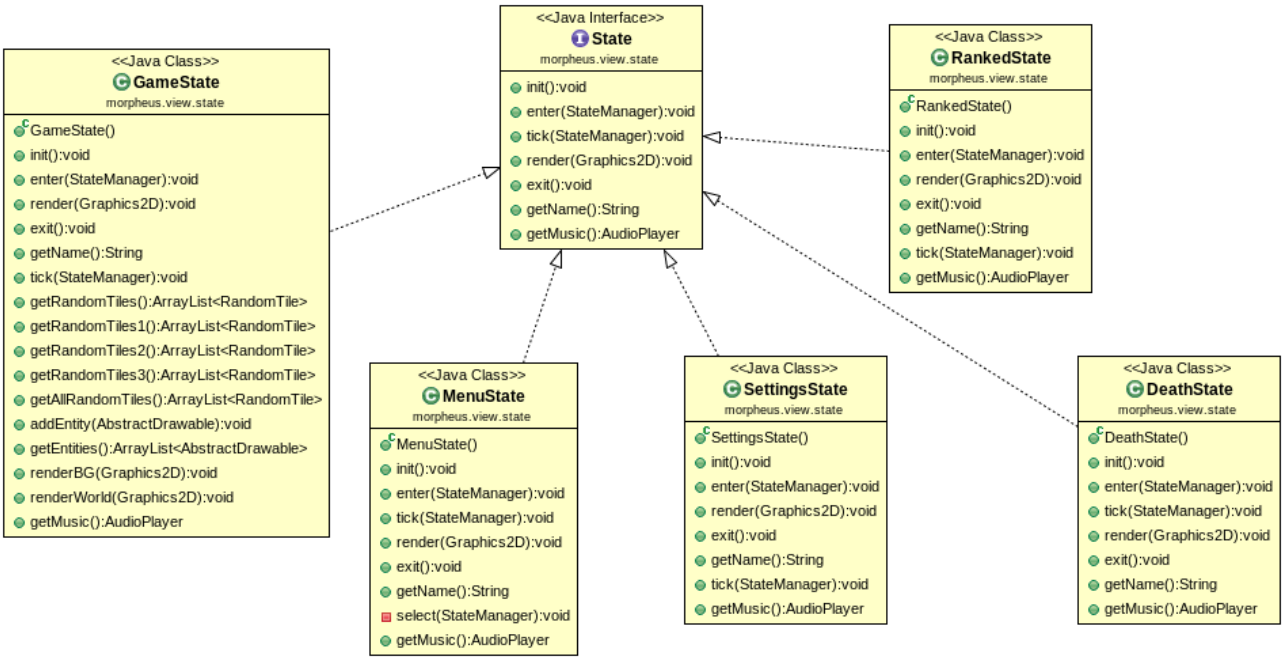
La reale potenzialità del sistema sopracitato è visibile nella classe GameState in cui l’istanza di BitMap, tramite i metodi build, genererà una delle mappe della classe TileMaps selezionata randomicamente. Il mondo di gioco verrà reso infinito tramite un sistema di 4 mappe renderizzate consecutivamente una all’altra le quali con l’avanzare del gioco verrano debitamente spostate offscreen per dare appunto l’illusione di “infinito”.

Il controller si occupa anche della gestione di tutte le collisioni del gioco. Precisamente la classe Collision ha questo compito la quale preleva dati dal Model relativamente alla posizione dei rettangoli utilizzati per identificare le collisioni. La classe si occupa di gestire le collisioni del player con le tile, i nemici, i proiettili dei nemici e i bonus presenti nel gioco.

Infine il controller ha cura oltre della gestione delle camera di gioco la quale è realizzata utlizzando il metodo translate della classe Graphics2D permettendo una sorta di spostamento di “inquadratura”, anche della mappatura degli input da tastiera e mouse implementata attraverso metodi statici booleani e la gestione del sistema del comparto sonoro composto da musica di background dei vari State ed effetti sonori.

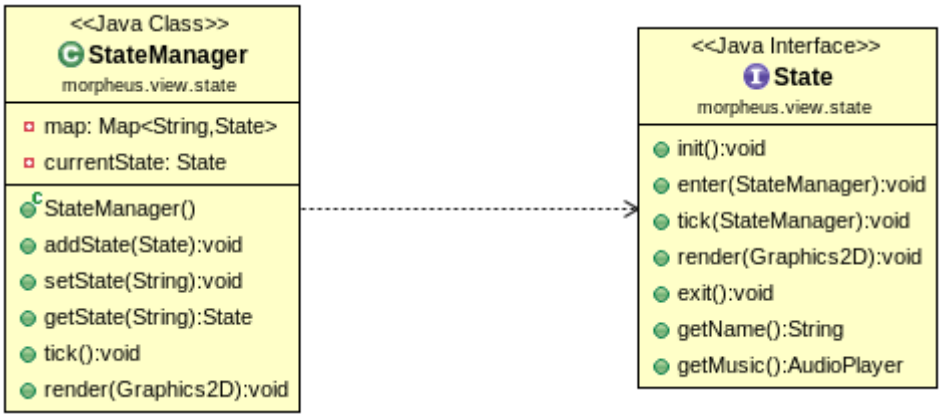
Mengozzi – View

La View si occupa di gestire l'interazione con l'utente e di mostrare a video il comportamento dell'applicazione. Per realizzarla è stato scelto di non utilizzare alcuna libreria grafica esterna.
L'aspetto più interessante della sua implementazione è sicuramente la sua divisione in State, in questo caso GameState, MenuState, RankedState, SettingState ed DeathState.



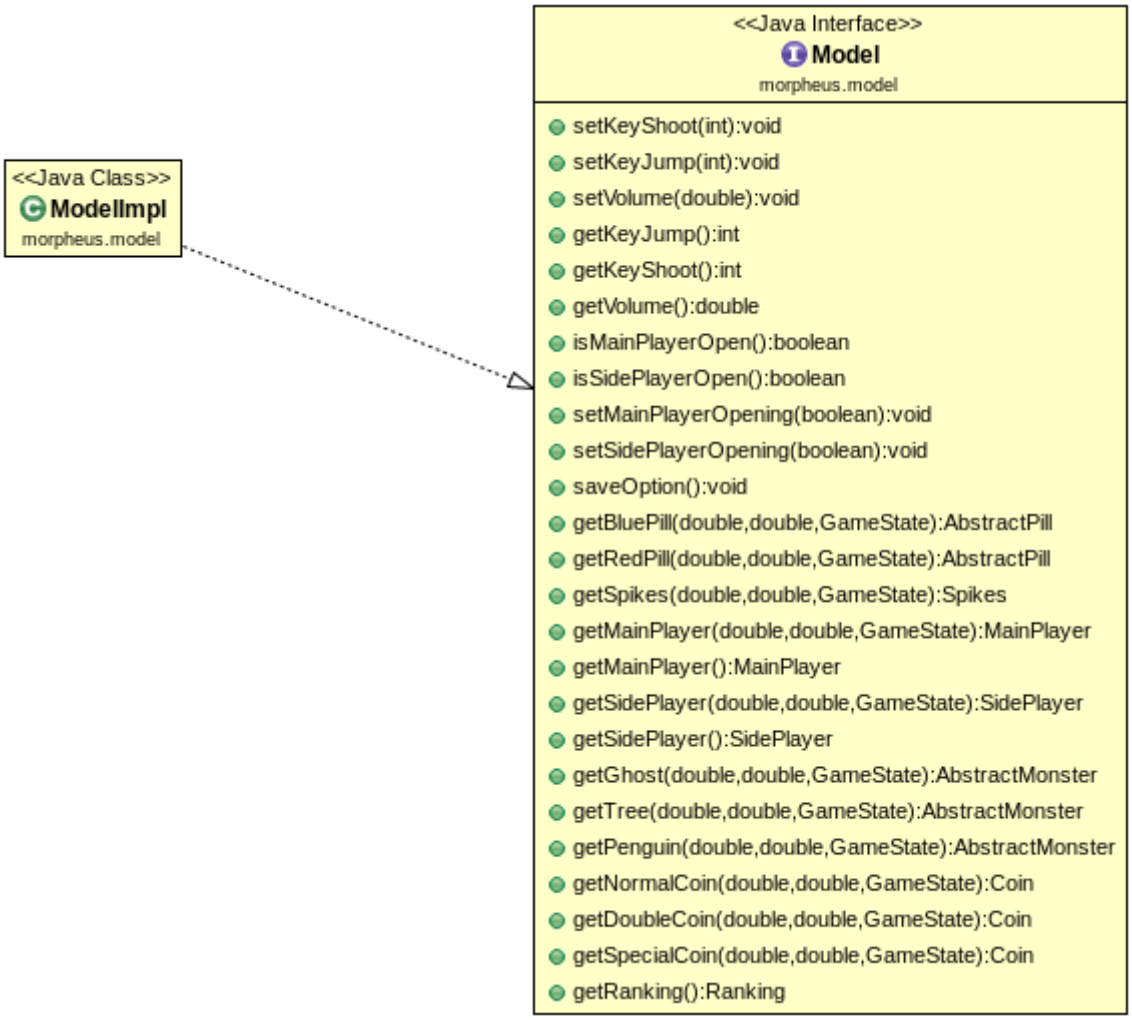
Questi State appunto implementano tutti la stessa interfaccia che ne definisce la struttura, ogni State dell'applicazione avrà come si nota dallo schema UML il metodo init (verrà chiamato una ed una sola volta all'avvio dell'applicazione), il metodo enter (ogni volta che si entra), i metodi tick e render (chiamati continuamente che calcolano la parte logica per il tick e renderizzano la parte grafica per il render), il metodo exit (chiamato ogni volta che si esce dallo state), getName che ne resistuisce il nome ed il metodo getMusic pensato inizialmente pet restituire la musica di un determinato State.

La gestione di questi State avviene grazie alla classe StateManager:



Lo StateManager contiene al suo interno una mappa contenente a sua volta ogni singolo State ed un campo privato di tipo State in cui viene memorizzato lo State corrente.
Lo StateManager ha a disposizione diversi metodi per gestire la mappa, quali addState (lo aggiunge alla mappa), setState (lo imposta come corrente), getState (ritorna lo State corrente), ed infine si hanno i metodi tick e render che chiamano i rispettivi dello State selezionato.

Il Model si compone di una interfaccia Model implementata da una classe ModelImpl la quale permette a View ed a Controller di accedere ai dati del Model stesso, in quanto contiene al suo interno un oggetto di tipo Option e di tipo Ranking. L'interfaccia Model rispetta in maniera tuttosommato consona il pattern MVC. Essa permette di modificare le impostazioni, salvare la classifica dei migliori risultati e modificare i dati relativi al gioco ed al player.



Capitolo 3
Sviluppo

3.1 Testing automatizzato

Per la realizzazione del progetto sono stati creati test automatizzati per le classi Ranking.java e Option.java che vanno ad effettuare salvataggio su file rispettivamente della classifica e delle opzioni di gioco. Per effettuare questi test ci si è forniti del supporto di JUnit.
Non sono stati effettuati altri test automatici in quanto realizzando un gioco ci si è concentrati di più sul testing manuale e visivo degli elementi a schermo.

3.2 Metodologia di lavoro

Il lavoro è stato suddiviso secondo il pattern MVC, la parte relativa al Model è stata sviluppata da Riciputi, la parte di View da Mengozzi, mentre il Controller è opera di Nigro.
Inoltre, per favorire la sincronia tra i membri del gruppo ci sserviti del repository online BitBucket ed abbiamo utilizzato Mercurial come DVCS.

3.3 Note di sviluppo

In fase di preparazione del progetto di è scelto di non utilizzare librerie grafiche esterne per grafica o quant'altro, all'interno del codice quindi non vi è alcuna presenza di librerie importate.
Vi è però presenza di algoritmi esterni:

- All'interno della classe vi è un algoritmo per il casting di una Image ad una BufferedImage preso dal sito http://stackoverflow.com/.

Infine, per il caricamento delle grafiche del gioco, abbiamo si è optato per l'utilizzo di una bufferstrategy tramite l'apposita classe BufferStrategy la quale permette l'utilizzo di un buffer per facilitare il caricamento degli asset grafici e migliorare le prestazioni generali dell'applicazione.

Capitolo 4

Commenti finali

4.1

Autovalutazione Model:
Complessivamente soddisfatto riguardo allo sviluppo del model, la fase di debug delle entità non ha comportato grossi cambiamenti in quanto si ritiene che la gerarchia dei Drawable costituisca una buona strategia per quello che doveva essere implementato. Come model si è anche cercato di rendere alle altre parti del progetto (View e Controller) i propri oggetti interni al meglio e il più accessibile possibile. Per un discorso di poca conoscenza in alcuni ambiti, certi elementi sono col tempo migliorabili, anche per aumentare la mia confidenza con essi. Tali ambiti sono le collisioni, non sempre affidabili, ma si è cercato di ridurre al minimo gli errori e la gestione della dinamicità del personaggio, in particolare nel salto e nella caduta.

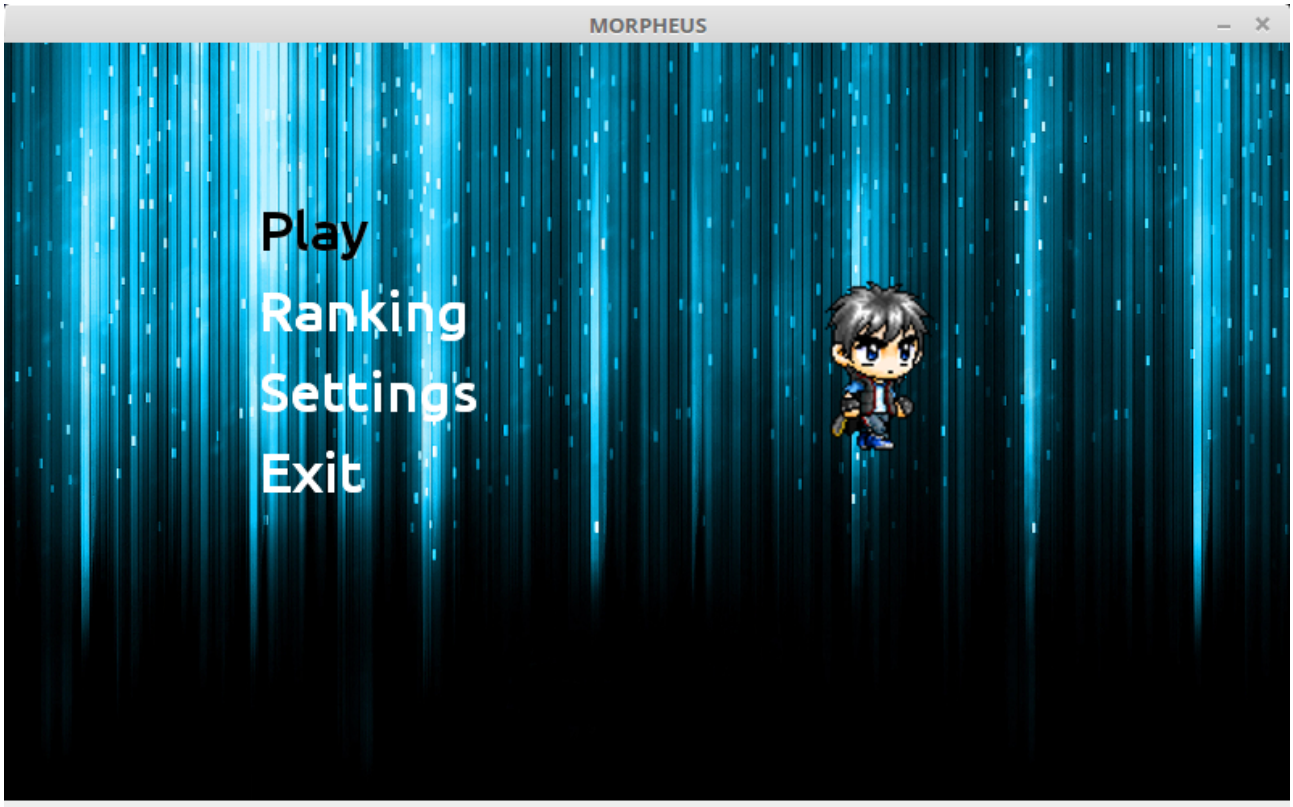
Autovalutazione Controller:
Personalmente mi trovo tuttosommato soddisfatto relativamente al lavoro svolto considerando le difficoltà incontrate durante il percorso di sviluppo di alcune funzioni come la creazione del sistema procedurale. A livello di mero contenuto mi considero soddisfatto, un po meno per alcune implementazioni riguardanti le collisioni le quali potevano essere calcolate in maniera più accurata.
Il difetto maggiore risiede sicuramente nell'implementazione del pattern MVC il quale, dal lato Controller, non è stato applicato in maniera corretta.

Autovalutazione View:
Mi ritengo abbastanza soddisfatto del lavoro svolto, in quanto le funzionalità previste in fase di progettazione sono state rispettate. Sono soddisfatto soprattutto di come ho suddiviso il mio lavoro e di come ho impostato la view a livello di organizzazione del codice. La suddivisione dell'applicazione in State è motivo di grande orgoglio per me in quanto la sua gestione non è stata per nulla semplice in quanto primo progetto nella

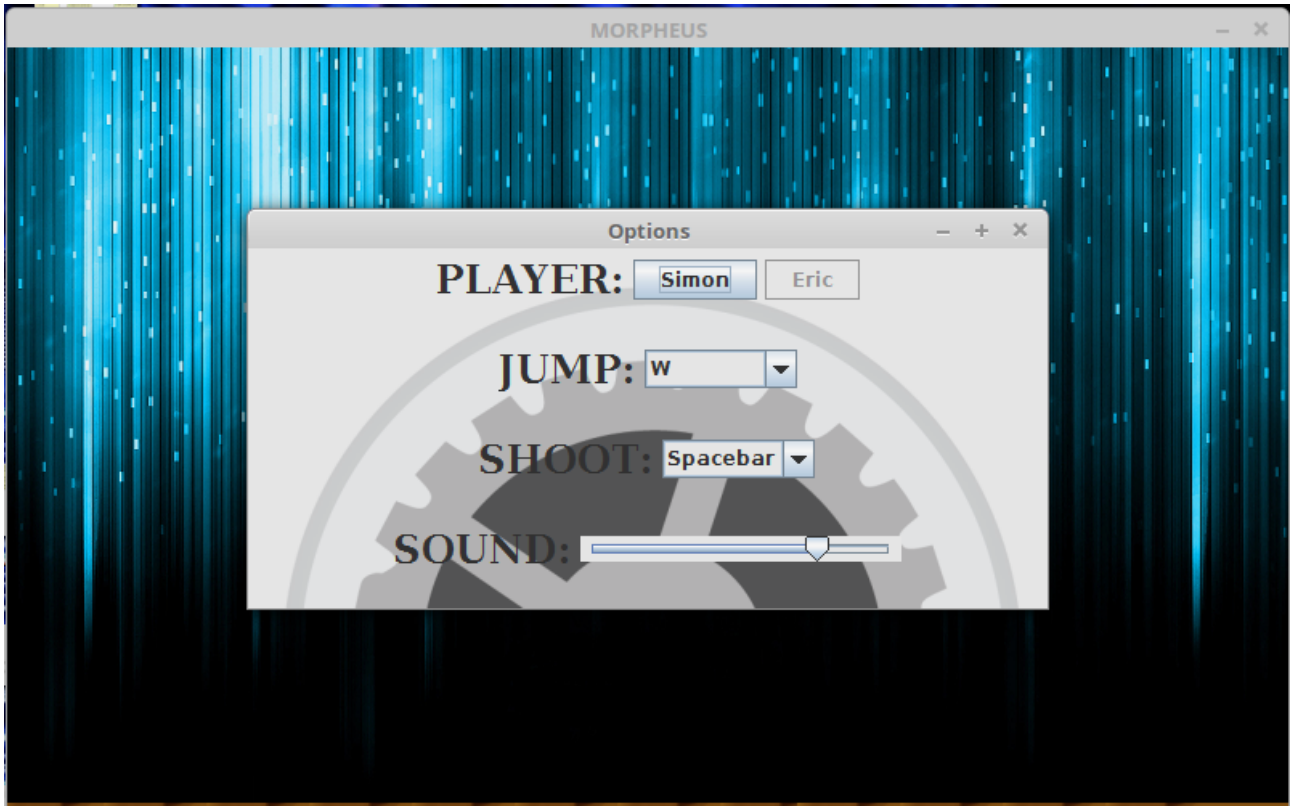
mia vita. Però c’è un grande rammarico, ovvero non aver rispettato appieno l’MVC dal lato view.
In futuro ritengo di poter lavorare ancora a questo progetto fornendo nuove funzionalità come l’inserimento della pausa durante il gioco, una maggior selezione di personaggi e di ambientazione di gioco.

Guida Utente

L’utilizzo di questo software è molto semplice, tuttavia mostriamo qui di seguito una guida per l’utente. All’avvio dell’applicazione ci si ritroverà di fronte ad un menù dal quale si potrà scegliere se giocare, guardare la classifica dei migliori risultati, modificare le impostazioni o uscire.



Dalle impostazioni si potranno modificare il personaggio, la configurazione dei tasti ed il volume.



Se invece si sceglierà di giocare ci si ritroverà di fronte a questa schermata con la quale si potrà giocare con le impostazioni scelte. In alto a destra sono rappresentati il numero di vite ed i proiettili a disposizione con i quali si potranno sconfiggere i nemici, mentre in alto a sinistra è rappresentato il punteggio il quale potrà essere incrementato con l'avanzamento del gioco e con la raccolta delle monete. Inoltre si hanno altri bonus (pillola rossa e pillola blu) i quali forniranno rispettivamente una vita e il pieno carico di proiettili.