OOS
"Ollie Oriented Skateboarding"

Sara Cappelletti
Rachele Margutti
Davide Tonelli
Fabio Veroli

24 aprile 2022

Indice

Capitolo 1

Analisi

L'obiettivo del gruppo è realizzare un gioco platform 2D su più livelli, cercando di ottenere il punteggio più alto possibile e completare le missioni. Il protagonista è uno skater che deve saltare degli ostacoli e raccogliere monete o power-up di cui usufruisce per ottenere abilità speciali che lo aiuteranno a sopravvivere. Con le monete potrà acquistare elementi per cambiare il proprio outfit o tentare la fortuna acquistando una Mystery Box. Man mano che il gioco avanza la difficoltà aumenta.

1.1 Requisiti

Requisiti funzionali

- Il gioco si apre con una schermata iniziale che permette di cominciare una partita.
- All'aumento della distanza percorsa, la difficoltà incrementa.
- Il salto viene effettuato tramite input preso da barra spaziatrice.
- In caso di collisione con un ostacolo, il counter delle vite diminuirà di uno.
- In caso di collisione con una piattaforma, il player portà salirci sopra.
- Quando il counter delle vite sarà pari a zero, si verificherà il termine della partita con conseguente apparizione della schermata di Game Over.
- Schermata di Game Over con possibilità di accesso allo shop o creazione di una nuova partita.

- Nella mappa saranno presenti delle monete raccoglibili che al termine della partita verranno aggiunte al totale precedentemente raccolto.
- Durante la partita compariranno 5 diversi item:
  - Lo scudo deve rendere il player immune agli ostacoli per alcuni secondi;
  - Il supersalto deve raddoppiare l'altezza raggiunta da un salto normale, sempre per un determinato lasso di tempo;
  - La vita extra deve incrementare di un'unità il counter delle vite;
  - La bomboletta spray deve far scomparire tutti gli ostacoli presenti nello schermo nel momento della sua raccolta;
  - Il funghetto deve raddoppiare il valore delle monete, sempre per un determinato lasso di tempo.
- Le monete raccolte saranno spendibili in uno shop per personalizzare il proprio personaggio o comprare la Mystery Box.
- Durante una partita sarà visualizzabile il numero delle monete raccolte, il numero delle vite e la distanza raggiunta.

Requisiti non funzionali

- Saranno presenti vari effetti sonori: musica di sottofondo, salto, raccolta di una moneta e Game Over.

1.2 Analisi e modello del dominio

Il gioco prevede un player che rimane fisso nella stessa posizione, mentre i vari ostacoli procedono verso di lui.

Bisognerà gestire la fisica del salto, la corretta gestione delle collisioni (in base all'entità che collide col player) e l'implementazione dei power-up.

Essendo la generazione degli ostacoli infinita e semi-randomica si dovrà gestire una corretta creazione delle varie entità in movimento in modo che sia possibile giocare in maniera scorrevole. Oltre a questi dovranno essere generati anche i power-up e le monete e gestire corretamente l'attivazione del loro effetto.

Dovremo ideare l'implementazione di un game loop composto da tre fasi: input processing (ad ogni iterazione del game loop si dovranno eseguire tutti i comandi digitati da tastiera dall'utente), update (aggiornamento dello stato

di gioco con tutte le dovute conseguenze come salto, raccogliementi, collisioni e game over) e rendering (visualizzare su schermo lo stato attuale di quel frame del model).

Inoltre lo shop dovrà occuparsi della creazione di due diversi oggetti: una Mystery Box che fornirà un premio in monete e la skin che permettono di cambiare il personaggio con cui si gioca. Verrano implementati i metodi di pagamento per ognuno di questi oggetti.

Tutte le statistiche e gli item acquistati dovranno essere salvati su file.

Il requisito non funzionale dei suoni dovrà essere ben coordinato alle azioni che avvengono nel gioco.

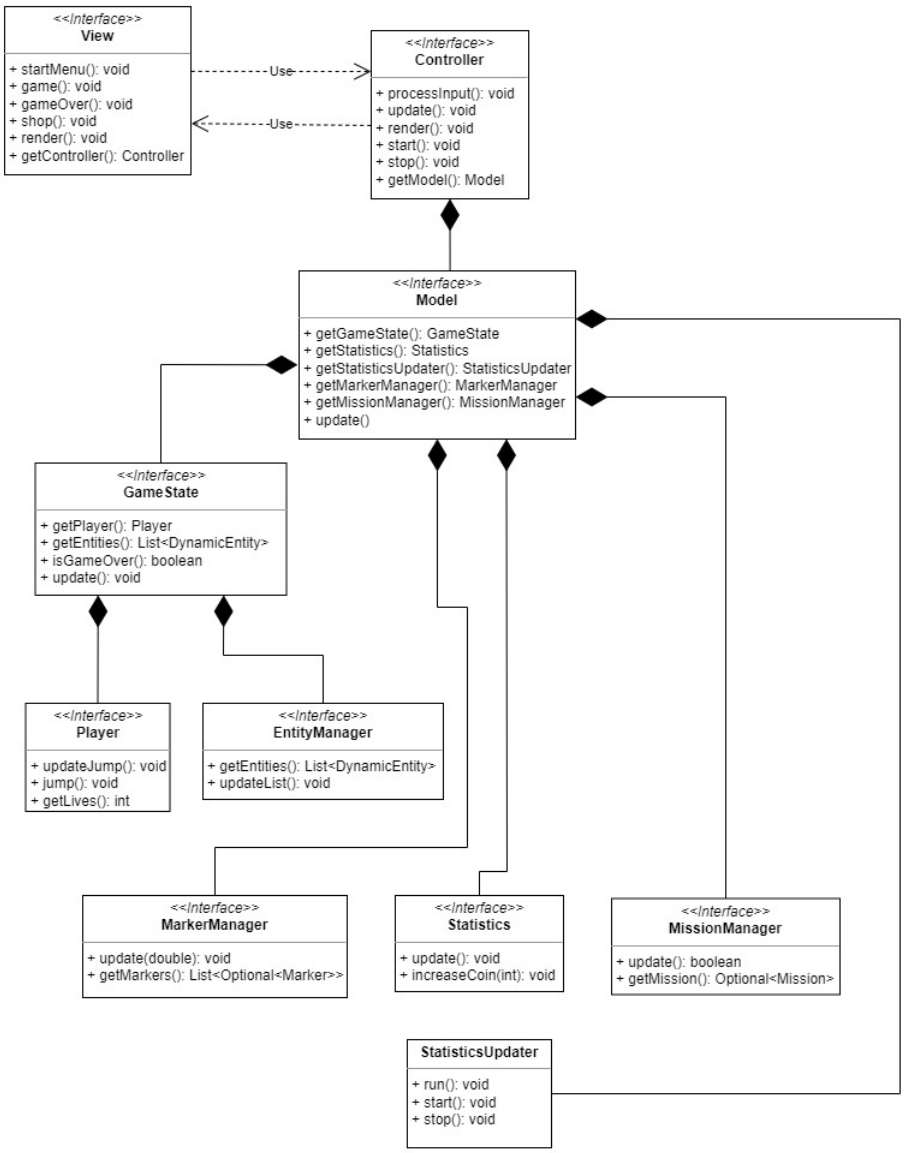

Figura 1.1: Schema UML dell'analisi del progetto, con rappresentate le entità principali ed i rapporti fra loro, fatto in vista dei cambiamenti effettuati (come descritto in seguito)

Capitolo 2

Design

2.1 Architettura

L'architettura di OOS segue il pattern architetturale MVC (vedi Figura 1.1). L'entry point del programma è nella view che, dopo aver renderizzato il menu iniziale, crea e passa il controllo al controller dell'applicazione. Ad inizio gioco il controller "ordina" al model di inizializzare tutte le entità di gioco (personaggio, piattaforme, ostacoli, ...). Da qui il controller inizia un game loop dove in prima fase si occupa di processare l'input rilevato, poi "ordina" al model di aggiornarsi e successivamente fa si che la view renderizzi i dati del model in quell'istante. Per ultimo, a fine partita, si occupa della gestione dei salvataggi. Alla partita successiva il controller crea un nuovo model, utilizzando i salvataggi precedenti.

Si è scelto di utilizzare questa architettura in quanto il controller orchestra l'aggiornarsi del model e il rendering della view lasciando però l'implementazione compito delle suddette classi. In questo modo aggiungere funzionalità diventa rapido e semplice e non richiede alcuna modifica delle classi già esistenti. Un esempio sono state le modifiche apportate successivamente per aggiungere il concetto dei Marker e delle missioni.

Anche lato view si mantiene questa modularità permettendo di cambiare in futuro viste di gioco e relativo aspetto, senza stravolgere l'architettura.

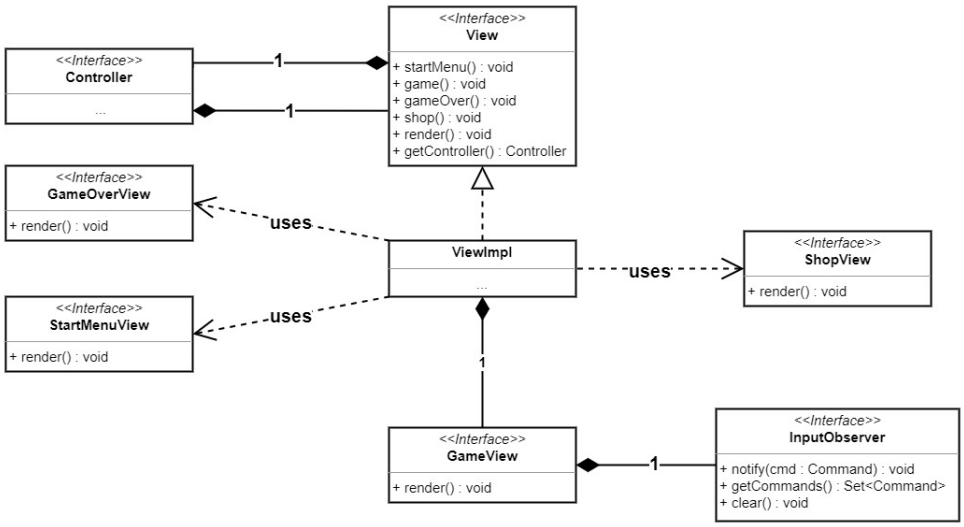

Figura 2.1: Schema UML dell'attuale gestione della view

Attualmente l'implementazione della view, chiamata ViewImpl, genera nuove sotto-view per il menu iniziale, il game over e lo shop; mentre mantiene un riferimento all'interfaccia GameView, fondamentale per rappresentare il model quando le viene ordinato di fare il render dal controller.

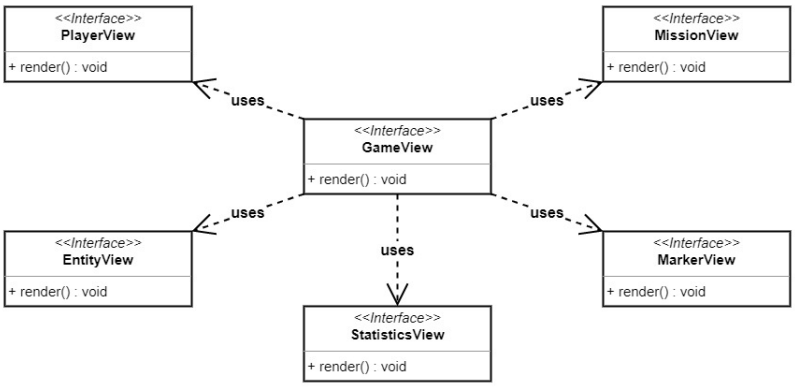

Figura 2.2: Schema UML della GameView

La GameView si occupa di raggruppare i rendering specializzati di ogni sottocomponente del model.

2.2 Design dettagliato

Sara Cappelletti

Gestione del player

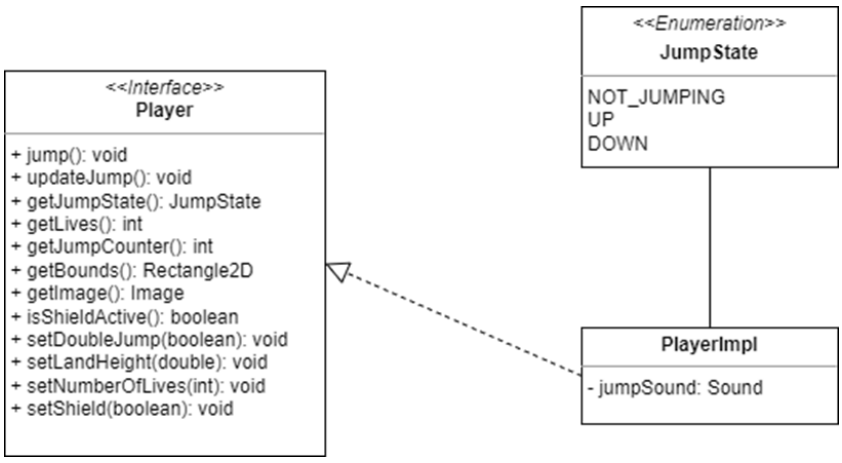

Figura 2.3: Schema UML relativo alla parte Model del Player

Problema In fase di sviluppo è stato necessario differenziare le fasi del salto del player: inizialmente erano presenti dei campi booleani per indicare

se il Player stesse saltando e se fosse in fase di discesa, ma ci siamo accorti che erano molto simili e portavano a ripetere del codice.

Soluzione Avendo a che fare con tre stati separati del player si è deciso quindi di creare una enumeration per differenziare la fase di non salto, quella di salto in salita e discesa per poterle poi riutilizzare anche in altre parti del codice (vedi fig. 2.4). Il salto inizia quando il giocatore preme spazio e la posizione del player viene aggiornata continuamente tramite un update, in modo da creare un movimento scorrevole. Il player ha inoltre un suono attivato a inizio salto.

Animazione del player

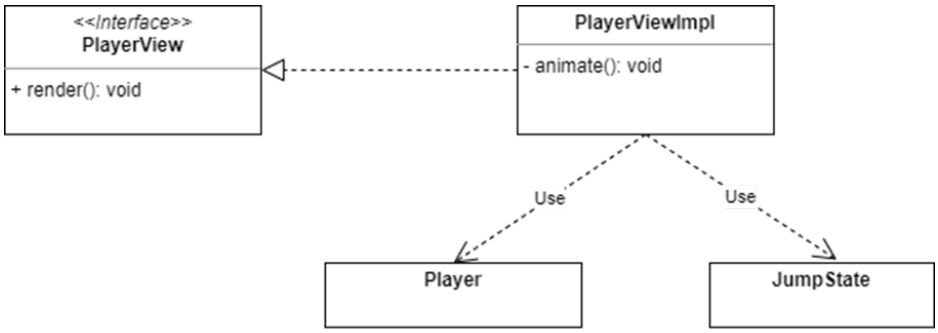

Figura 2.4: Schema UML relativo alla parte View del Player

Problema Creare un'animazione del player.

Soluzione Dato che anche l'animazione del player è divisibile in tre stati, si è deciso di riutilizzare la enumuration creata in Figura 2.3 e aggiornare continuamente l'immagine del player.

Gestione collisioni

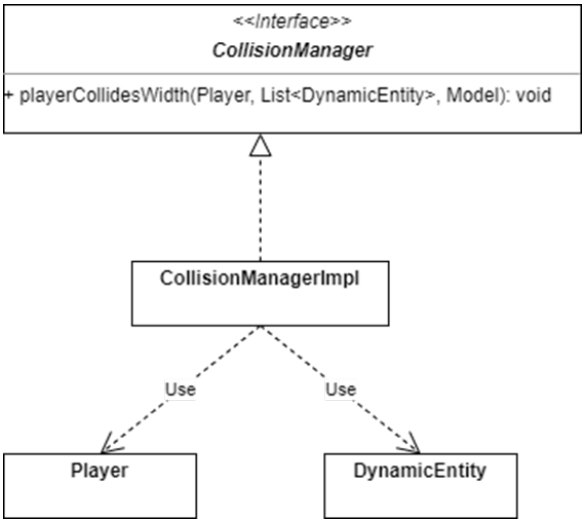

Figura 2.5: Schema UML relativo alle collisioni

Problema Gestire in modo efficace le collisioni.

Soluzione Le collisioni vengono gestite tutte insieme dentro la classe CollisionManager, che si comporta in modo differente in base ai vari tipi di Entity che collidono col Player.

Rachele Margutti

Power-up e timer

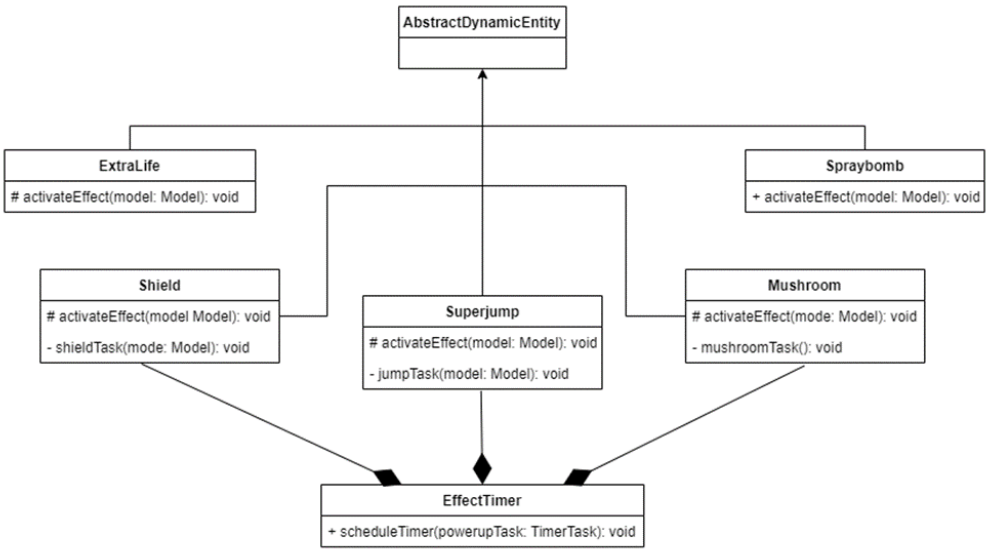

Figura 2.6: Rappresentazione UML dei diversi power-up e del relativo timer

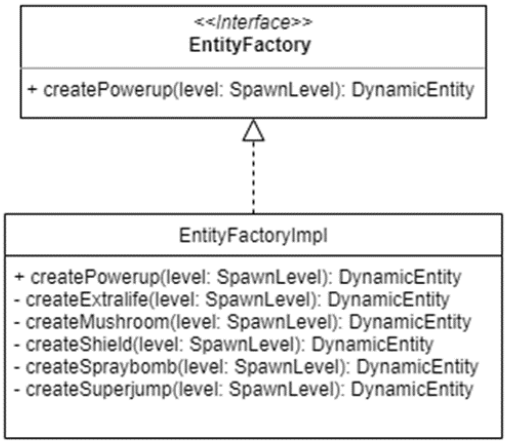

Figura 2.7: Rappresentazione UML della Factory

Problema Dopo una prima realizzazione di prova dei power-up, mi sono resa conto di correre il rischio di imbattermi in numerose ripetizioni di codice

in quanto essi hanno un comportamento molto simile alle altre entità, questo perché condividono il modo di apparizione sullo schermo e l'attivazione di un effetto.

Soluzione Il mio compagno Fabio Veroli ed io abbiamo perciò deciso di fare in modo che i power-up estendessero la classe astratta AbstractDynamicEntity, utilizzando così il pattern Template Method. Alcuni power-up sfruttano un EffectTimer in modo da rendere il loro utilizzo a tempo.

ShopModel

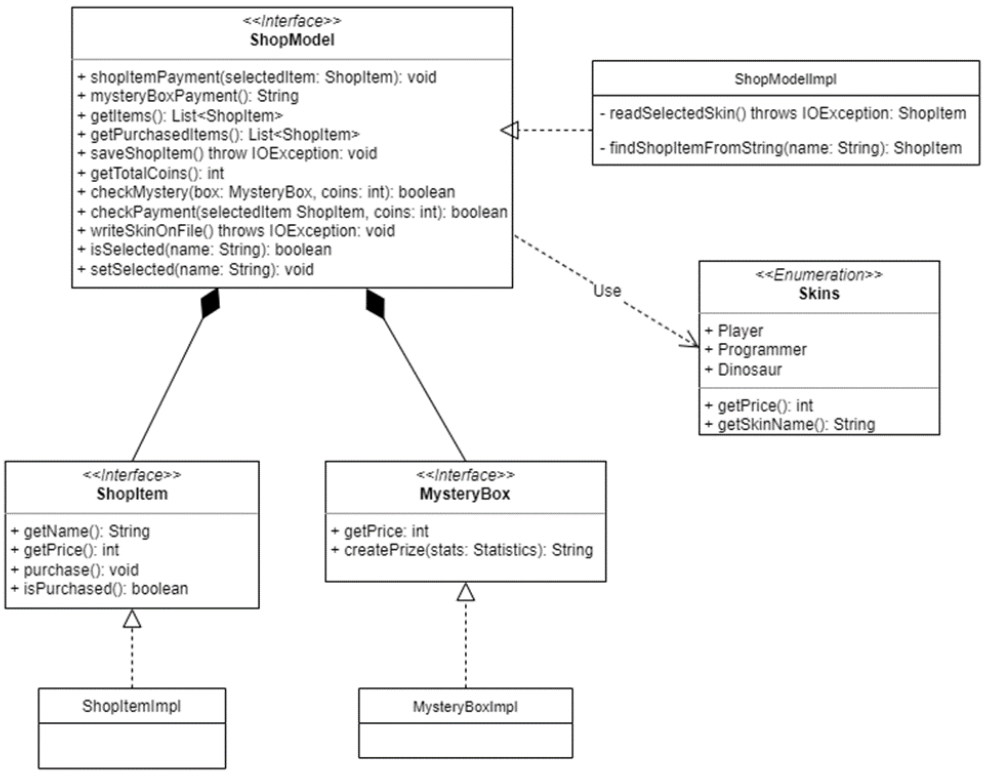

Figura 2.8: Rappresentazione UML della ShopModel

Problema Realizzazione di oggetti acquistabili nello shop.

Soluzione Gli oggetti sono stati creati tramite la realizzazione di due classi ShopItemImpl e MysteryBoxImpl: nella prima saranno disponibili le info riguardanti le skin, ossia i vari personaggi utilizzabili durante le partite e

per fare ciò ho creato una enumeration Skins; nella seconda, oltre alle info riguardanti il prezzo, verrà creato un premio per il giocatore.

Problema Gestione del pagamento.

Soluzione Per ovviare al problema del pagamento, nella classe ShopModelImpl sono stati implementati dei metodi per effettuare correttamente le transazioni per i diversi oggetti.

Per verificare il suo corretto funzionamento, mi sono assicurata che questo sia impossibile ogni volta in cui il giocatore non possieda monete sufficienti e, nel caso in cui si tratti di uno ShopItem, che la skin selezionata non sia già stata acquistata in precedenza.

Problema Gestione del salvataggio su file.

Soluzione Ogni volta che uno ShopItem verrà acquistato, sarà aggiunto ad una lista per essere salvato su file, in questo modo sarà sempre possibile essere a conoscenza dei personaggi posseduti e di conseguenza selezionarli successivamente per le partite.

Shop View e Controller

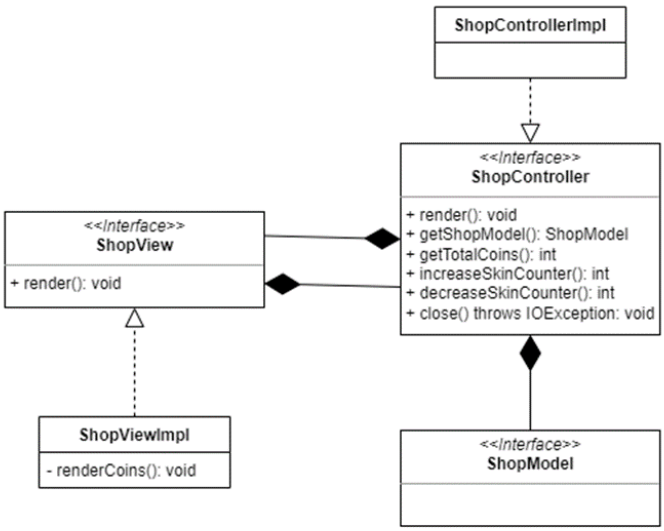

Figura 2.9: Rappresentazione UML del controller e della view dello Shop

Problema Suddivisone dello shop.

Soluzione Si è deciso di sfruttare il pattern MVC per la corretta divisione dei compiti delle diverse parti dello shop. In questo modo, la view farà solo la render dei comandi ricevuti dal controller, i quali a loro volta vengono passati dalla ShopModel.

Davide Tonelli

Controller dell'applicazione

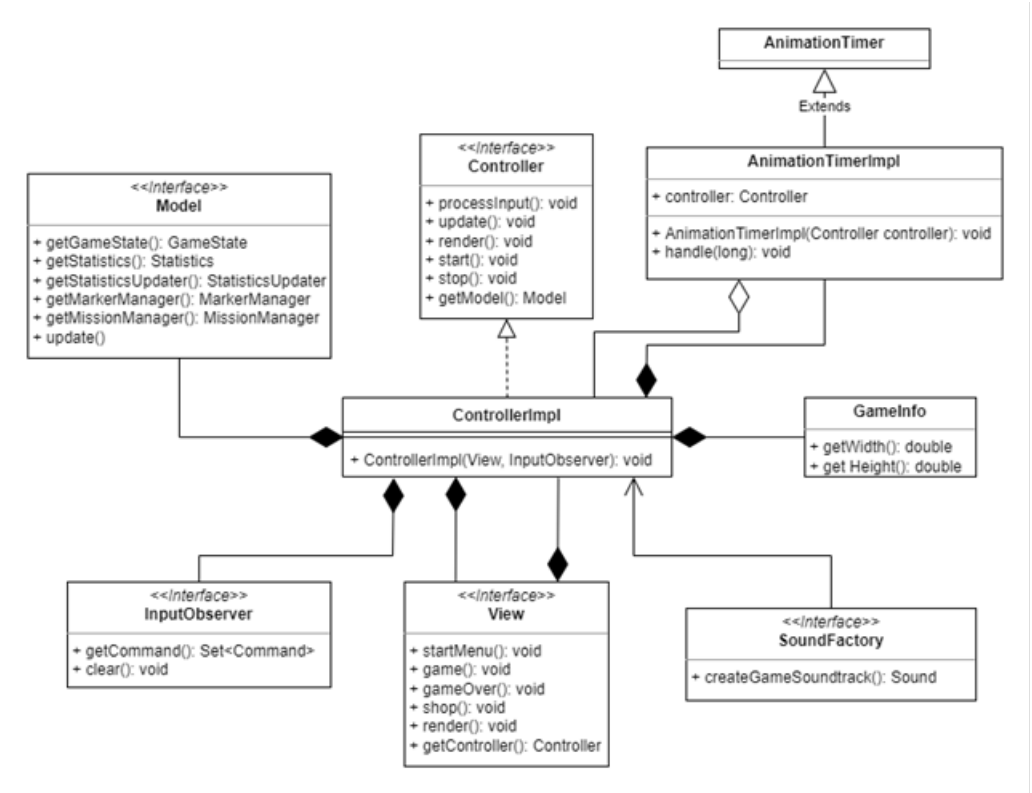

Figura 2.10: Rappresentazione UML del controller dell'applicazione

Problema Realizzare un controller secondo il pattern architetturale MVC, modulabile e non saturo di elementi.

Soluzione Si è scelto di rendere compito del Controller ogni fase del game loop, permettendogli di "ordinare" al Model quando aggiornarsi ed alla View

quando eseguire il rendering. Per ottenere un game loop si è scelto di utilizzare un AnimationTimerImpl, estensione di AnimationTimer. GameInfo permette, grazie alla sua presenza in Controller, di trasmettere le informazioni necessarie per il gioco evitando ripetizioni di codice. Dopo un'attenta analisi si è optato di rendere ControllerImpl e View reciprocamente accessibili permettendo così al controller di essere sempre responsabile di un cambio di schermata di gioco seguendo le linee guida MVC.

Input da tastiera

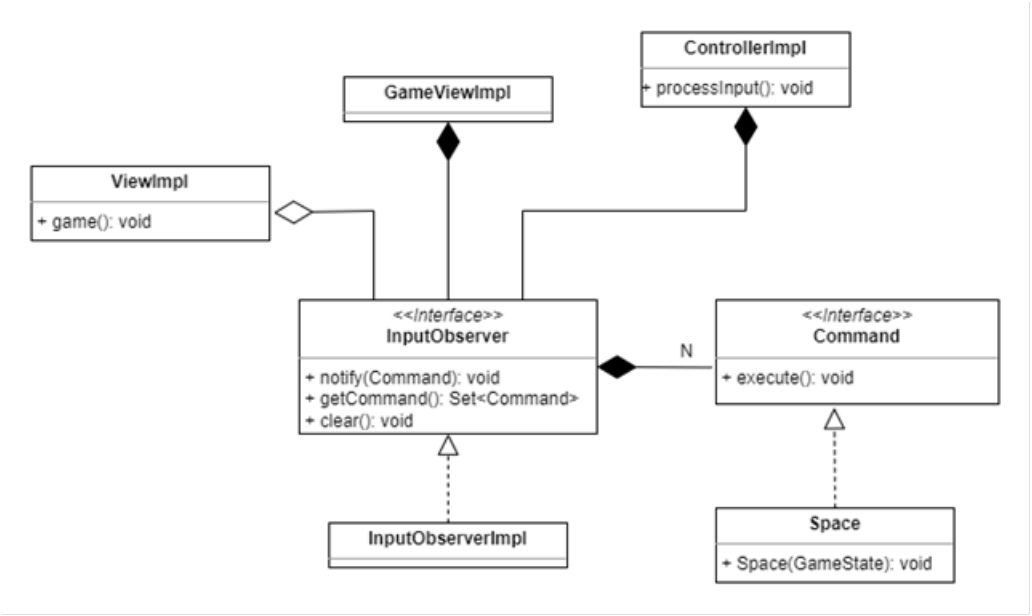

Figura 2.11: Rappresentazione UML della gestione dell'input da tastiera

Problema Implementare un sistema di input flessibile e modulabile.

Soluzione Per ottenere il sistema di input ricercato si è presa ispirazione dal pattern Observer con osservatore InputObserver ed osservato GameViewImpl. ControllerImpl in fase di input processing utilizzerà tutto l'input raccolto da InputObserver. L'interfaccia Command permette di creare infinite tipologie di comandi rendendo tale sistema modulabile.

Suoni di gioco

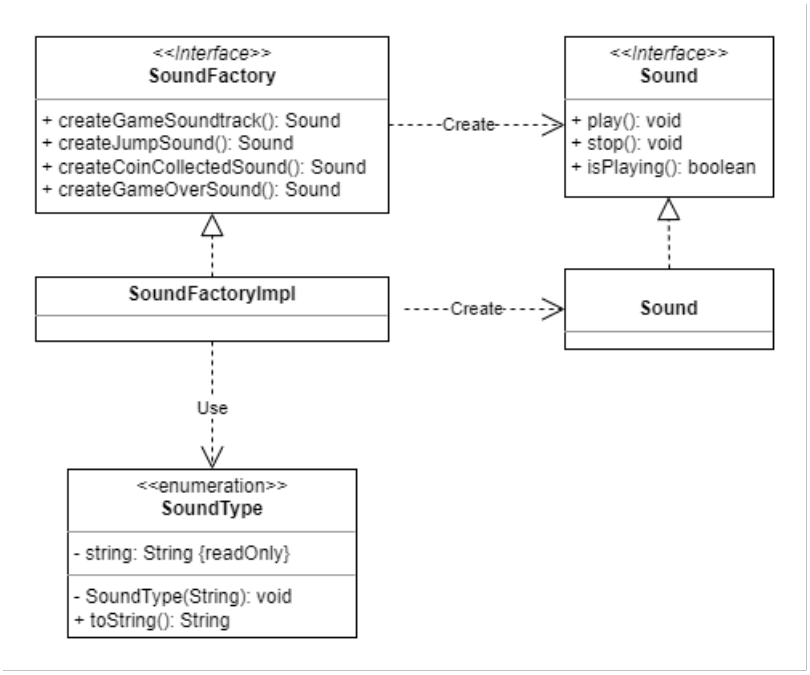

Figura 2.12: Rappresentazione UML della creazione e gestione dei suoni di gioco

Problema Implementare un sistema di suoni funzionale.

Soluzione Per la creazione di Sound si è usato il pattern Factory Method con i metodi forniti dall'interfaccia SoundFactory. L'implementazione della factory SoundFactoryImpl crea con i suoi metodi una classe anonima che implementa Sound, in modo tale da permettere varianti di suoni in futuro (es. Sound che cambia dinamicamente il proprio volume in base ad alcuni avvenimenti di gioco). Viene infine utilizzata una enumerazione per ottenere il nome dei suoni di gioco.

Marcatori della distanza di gioco

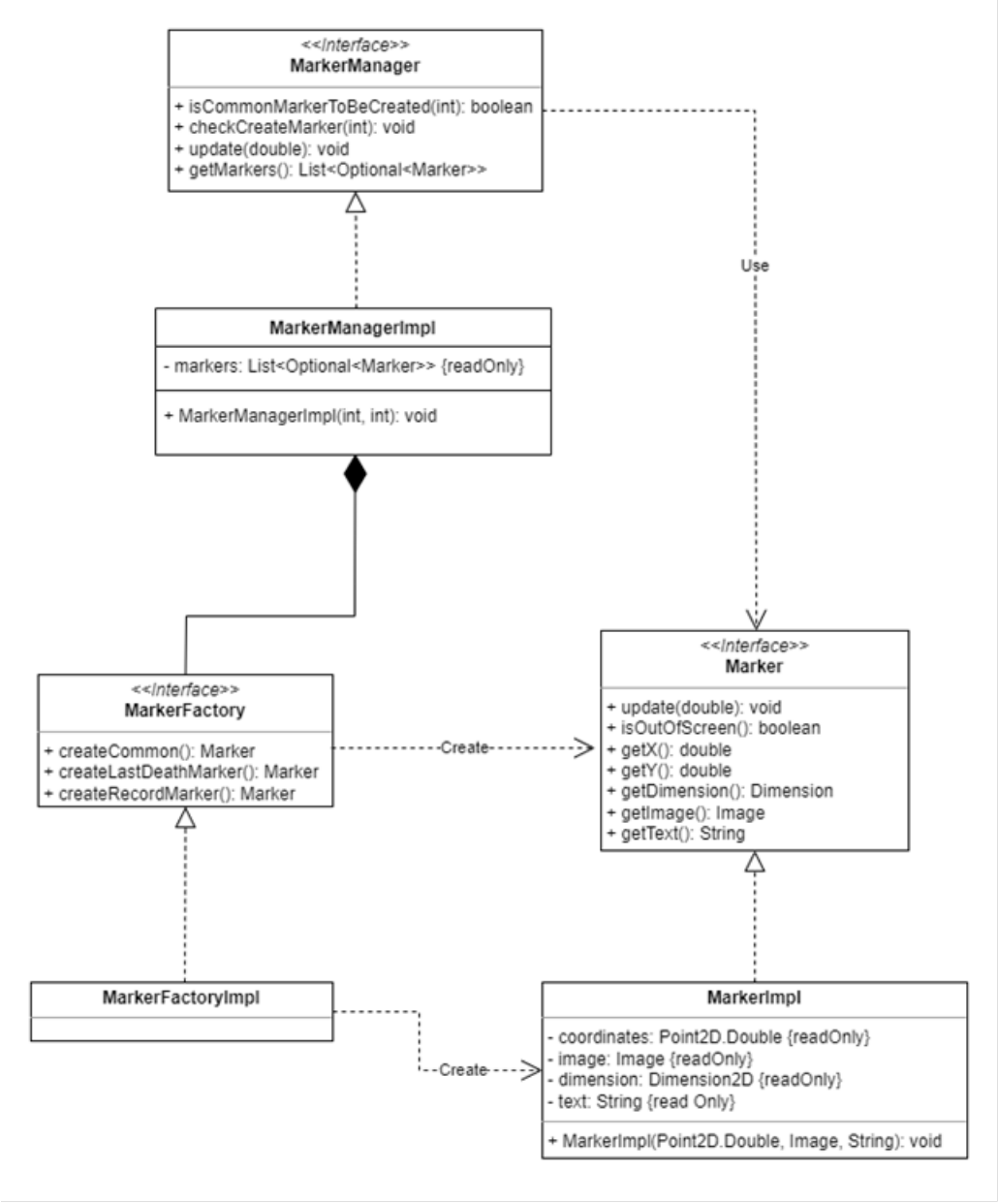

Figura 2.13: Rappresentazione UML della creazione e gestione dei marcatori di distanza

Problema Data la velocità di gioco incrementale, il giocatore percorre sempre più distanza ad ogni secondo, rendendo impossibile prevedere il punteg-

gio totalizzato a fine partita. Inoltre si è notato come l'assenza di elementi di gioco che notificassero il superamento del record personale (o del punteggio totalizzato nell'ultima partita) rendesse l'esperienza dell'utente meno accattivante.

Soluzione Per poter aggiungere questa funzionalità di gioco si è ideato un manager di marker che, mantenendo una lista di Marker, gestisce quando un Marker dovrà essere creato e con che caratteristiche. Il manager di marker si occupa anche dell'aggiornamento di ogni marcatore. Si è scelto di utilizzare un manager di marcatori per la facile modularità che propone, in questo modo non è presente un limite architetturale per quanto riguarda il numero e le caratteristiche di ogni Marker. Per la creazione di Marker si è utilizzato il pattern Factory Method con i metodi forniti dall'interfaccia MarkerFactory.

Missioni di gioco

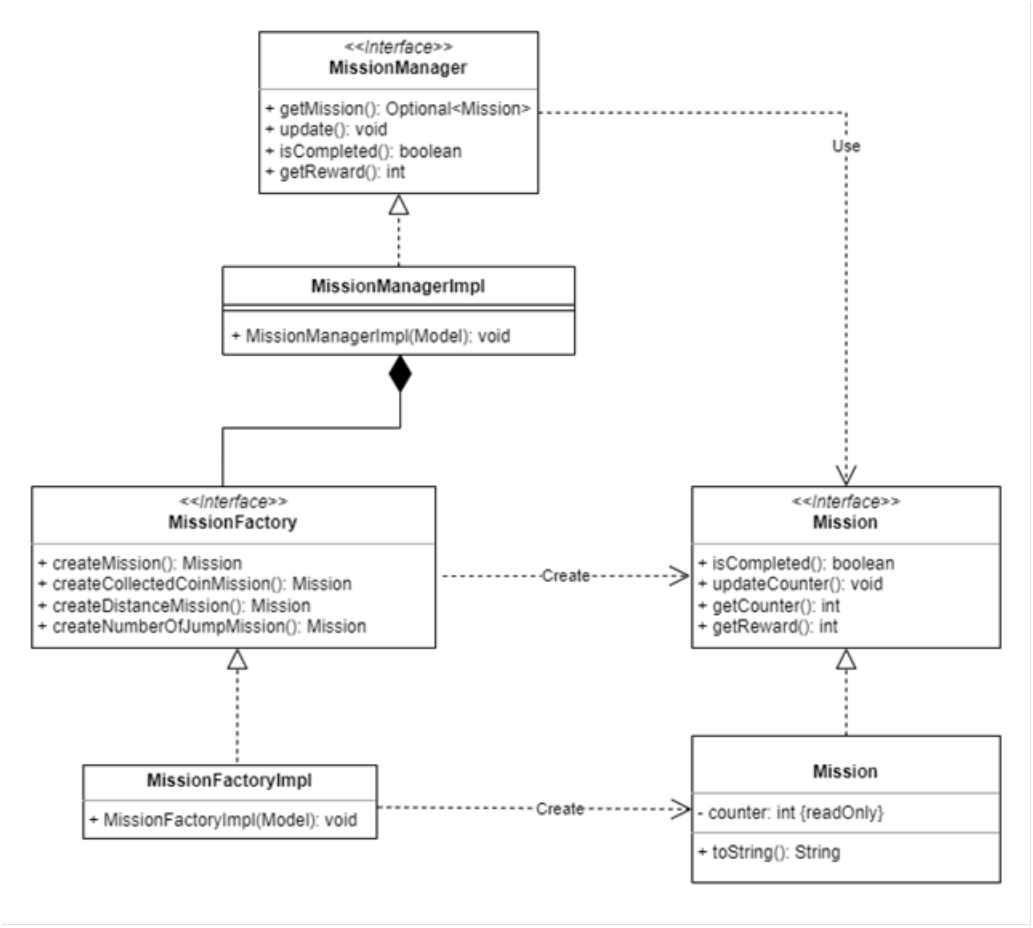

Figura 2.14: Rappresentazione UML della gestione e creazione di missioni di gioco

Problema Realizzazione di sfide di gioco facoltative, casuali ed affrontabili in grado di rendere più coinvolgente l'esperienza di gioco.

Soluzione Si è ideato un manager di missioni con il compito di gestirne creazione ed aggiornamento. Attualmente si è scelto di avere una sola missione per partita in quanto si è reputata sufficiente ed adeguata per un corretto grado di sfida, ma questo design permette di non avere limiti architetturali riguardo al numero e tipologia di missioni. La creazione di Mission è stata realizzata utilizzando il pattern Factory Method con i metodi forniti nell'interfaccia MissionFactory. Data la desiderata modularità del tipo di missioni,

l'implementazione della factory MissionFactoryImpl crea con i suoi metodi una classe anonima che implementa Mission, in modo tale da permettere infinite varianti.

Aggiornamento delle statistiche di gioco

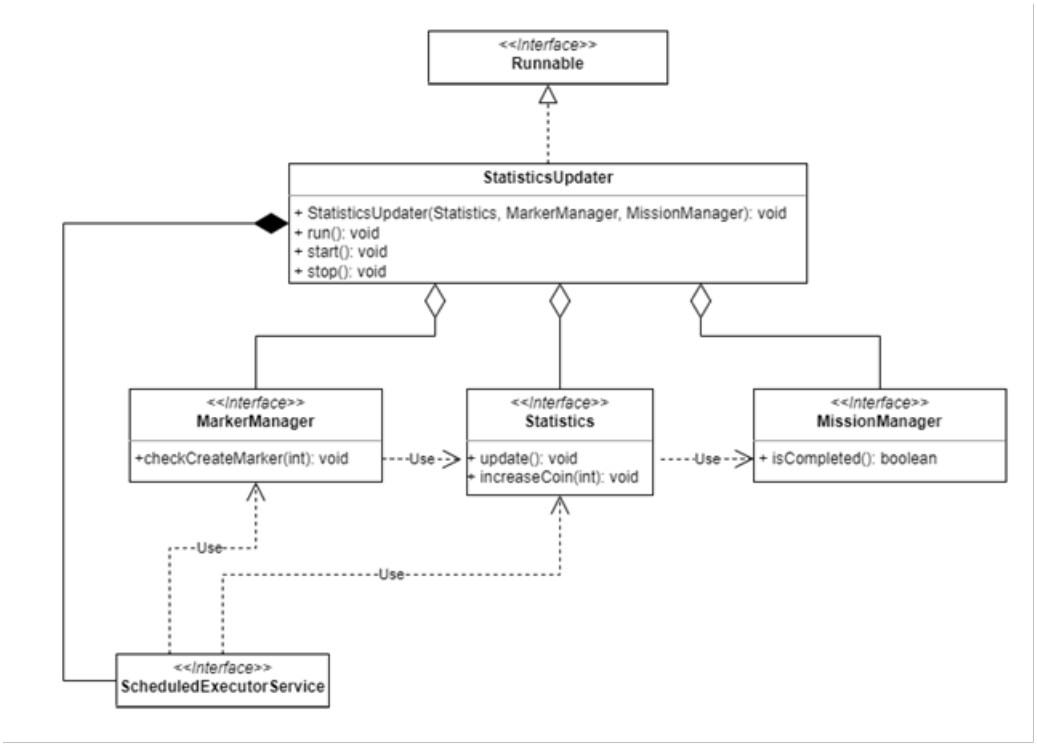

Figura 2.15: Rappresentazione UML della gestione dell'aggiornamento delle statistiche di gioco

Problema Ottenimento di un realistico ed incrementale grado di difficoltà di gioco, ponendo particolare attenzione sul compromesso tra giocabilità e difficoltà.

Soluzione Per ottenere un incremento della difficoltà di gioco realistico si è utilizzato ScheduledExecutorService che permette di eseguire periodicamente un aggiornamento delle statistiche di gioco (tra cui la difficoltà). L'unità di tempo per la frequenza di aggiornamenti è cambiabile permettendo in futuro l'inserimento di varianti di gioco. Siccome l'aggiornamento delle statistiche viene effettuato separatamente dagli aggiornamenti del game loop, si è scelto di includere anche il controllo sulla creazione di Marker effettuato dal

MarkerManager e il controllo riguardante il completamento delle missioni da parte del MissionManager.

Fabio Veroli

Riuso del codice delle entità

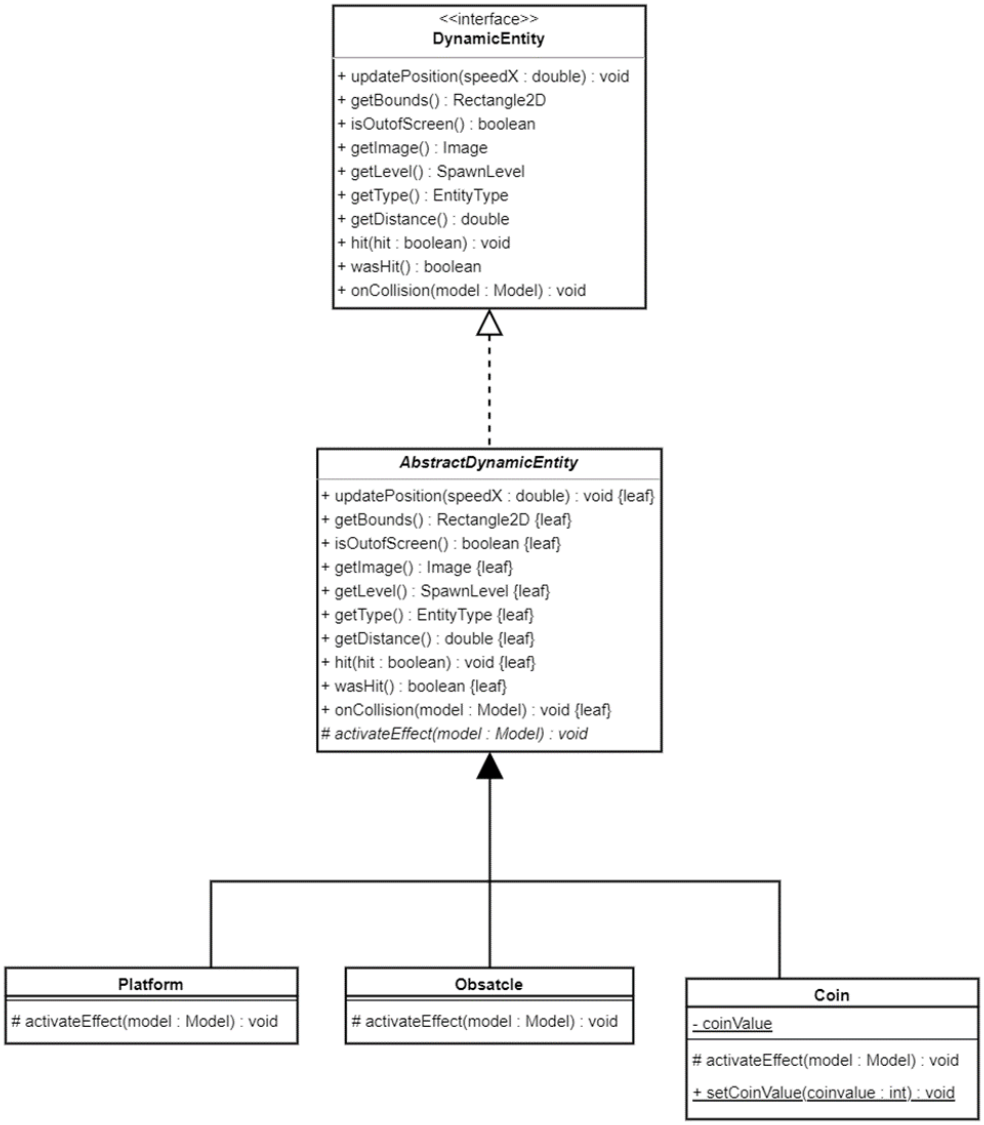

Figura 2.16: Rappresentazione UML dell'utilizzo di Template Method per la creazione delle entità

Problema Il gioco si compone di diverse entità che si muovono in direzione del personaggio, e in caso di collisione con quest'ultimo devono svolgere un'azione diversa in base al loro tipo. In fase di sviluppo abbiamo notato che le entità condividevano gran parte del comportamento e questo avrebbe portato ad una duplicazione del codice e a classi molto simili tra loro.

Soluzione Per risolvere il problema è stata creata un'interfaccia comune a tutte le entità (DynamicEntity), una classe astratta che implementa l'interfaccia definendo i comportamenti comuni a tutte le entità (AbstractDynamicEntity). Per massimizzare il riuso, è stato utilizzato il pattern template method in cui, come da Figura 2.16, il metodo template è il metodo non astratto onCollision(), che chiama il metodo astratto e protetto activateEffect(), la cui implementazione è fornita dalle sottoclassi della classe astratta. In questo modo tutte la classi che estendono AbstractDynamicEntity definiscono un comportamento diverso in seguito ad una collisione con il personaggio.

Astrarre la creazione di entità dalla loro classe

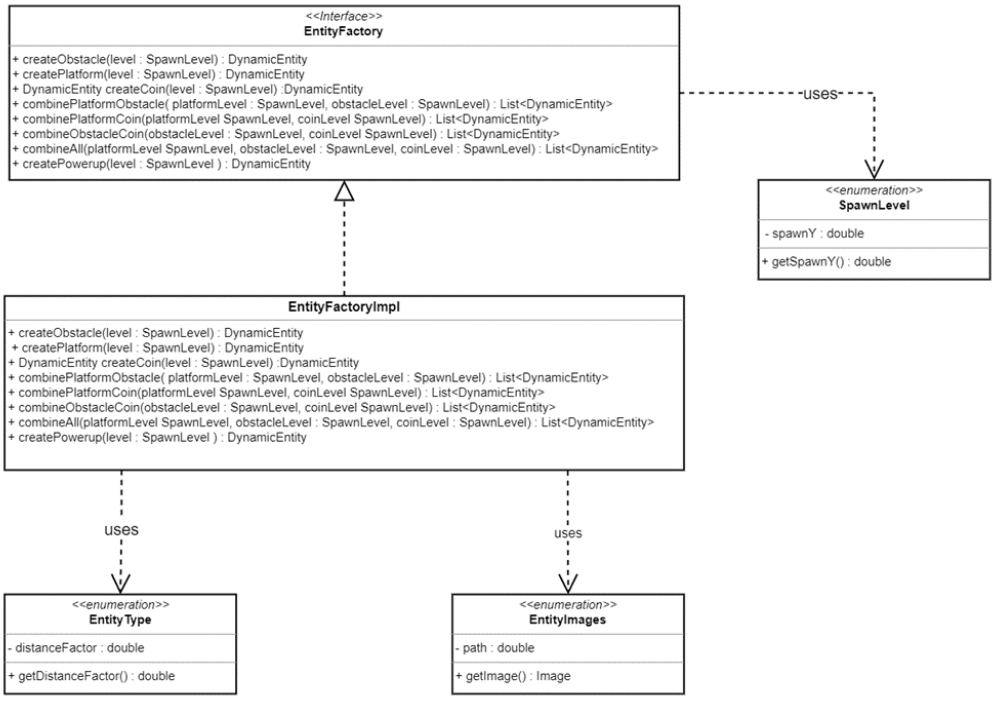

Figura 2.17: Rappresentazione UML del pattern Factory Method usato per creare le entità

Problema Creare entità di diverso tipo (o collezioni di entità) in modo che la logica di creazione sia esterna alla classe che le identifica e allo stesso tempo esterna ad eventuali classi che ne vogliano usare delle istanze (vedi il manager di entità).

Soluzione Uso il pattern Factory Method in cui, come da Figura 2.17, l'interfaccia EntityFactory fornisce i metodi factory per creare e ritornare l'entità (o la collezione di entità), mentre la classe EntityFactoryImpl incapsula la logica per la creazione delle entità. Per il calcolo delle loro proprietà vengo anche usate varie enumerazioni che contengono informazioni relative all'immagine, al livello e alla distanza fra entità.

Gestire diversi tipi di entità correlate tra loro

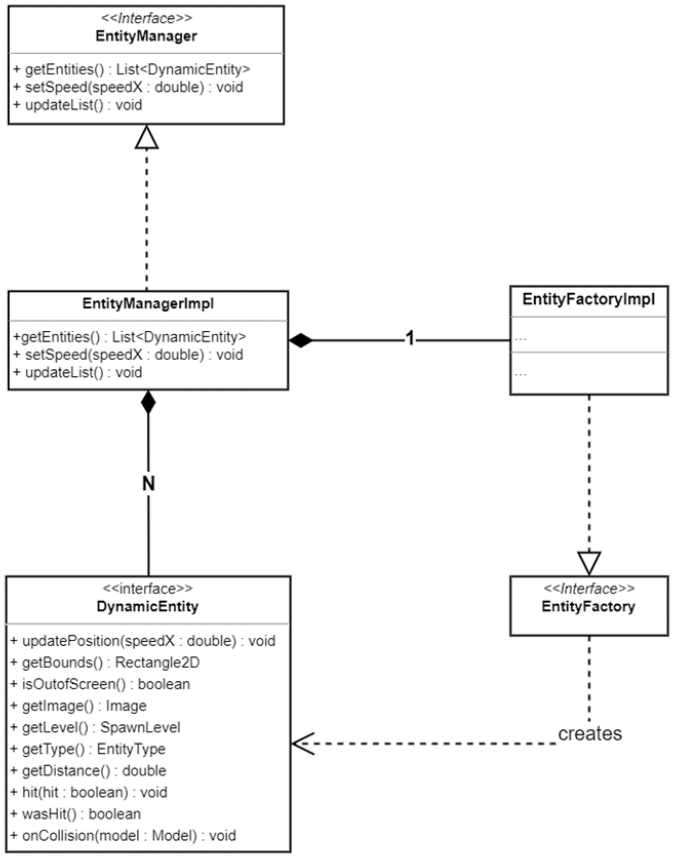

Figura 2.18: Rappresentazione UML del funzionamento di un EntityManager

Problema Gestire le diverse entità (piattaforme, monete, ostacoli e potenziamenti) durante il gioco in modo che, seppur compaiano in modo semi-randomico, rendano scorrevole e giocabile per lungo tempo la partita.

Soluzione Optiamo per la creazione di un manager di entità, che se pur di diverso tipo implementano tutte la stessa interfaccia DynamicEntity. In questo modo tutte le entità possono essere gestite tramite un'unica lista ordinata. Gli elementi vengono creati tramite una EntityFactory, che come descritto in precedenza si occupa della logica di creazione delle entità; una volta creati saranno poi aggiunti alla lista direttamente dal manager. Sulla lista verranno poi effettuate operazioni di rimozione ed aggiunta di elementi secondo il verificarsi di certe condizioni. Le diverse configurazioni di entità saranno generate in modo semi-randomico.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Ognuno di noi ha creato vari test in JUnit 5 per testare le funzionalità delle proprie classi:

- **PlayerTest:** testa l'inizio del salto, l'aggiornamento del salto, il contatore delle vite, il contatore dei salti e che i Bound del Player siano corretti.
- **CollisionTest:** testa le collisioni del Player con ostacoli, piattaforme e monete.
- **PowerupTest:** creazione dei diversi Powerup e corretto funzionamento dei rispettivi effetti.
- **CommandTest:** esecuzione di un comando.
- **InputObserverTest:** corretto stato iniziale, aggiunta e rimozione di comandi.
- **MarkerTest:** corretto stato iniziale, corretto funzionamento marker comune, last death marker, record marker.
- **MissionTest:** creazione randomica di una missione, missione basata sulla distanza raggiunta, missione basata sul numero di monete raccolte, missione basata sul numero di salti effettuati.
- **StatisticsTest:** aggiornamento delle statistiche, difficoltà massima raggiungibile, incremento delle monete.

- **EntityTest:** coordinate dell'entità, funzionamento dello spostamento e attivazione degli effetti.
- **ManagerTest:** aggiunta e rimozione di elementi dalla lista, aggiornamento della posizione delle entità e aggiornamento della lista.

  Nota: in alcune classi di test sono state usate righe di codice prese da StackOverflow per risolvere il seguente errore "java.lang.RuntimeException: Internal graphics not initialized yet". L'errore si presentava in quanto alcune classi del nostro progetto, seppur appartenenti al model, sfruttano la classe Image per il calcolo di alcune informazioni; nelle classi di test quindi si presentavano problemi in quanto lo schermo risultava essere nullo e un'immagine deve essere creata e modificata dentro il JavaFX Appilication Thread. Abbiamo deciso di non sfruttare API come TestFX, ma di inizializzare lo schermo ed evitare il presentarsi dell'errore come riportato nel codice delle classi di test.

3.2 Metodologia di lavoro

Descrizione utilizzo DVCS

Per ogni feature è stato creato un branch (es. "feature.input") di cui si è poi fatto il merge una volta raggiunto un adeguato funzionamento della feature stessa.

Descrizione metodologia

Come gruppo, fin dalle prime fasi di progettazione lo scopo è stato quello di creare uno scheletro di interfacce in modo tale da permettere ad ognuno di lavorare individualmente. Durante l'analisi e l'implementazione dei vari compiti assegnati, l'approccio è stato quello di creare bozze di design su carta. Una volta comprese sufficientemente le relazioni fra i vari oggetti in questione si procedeva a scrivere le varie parti di codice. L'approccio all'implementazione è stato quello di scrivere classi "grezze" e una volta testato il loro funzionamento cercare modi per rendere il codice più pulito, sicuro ed eliminando le parti non necessarie. La coordinazione del gruppo è stata fondamentale in quanto ci si è sempre organizzati sulla priorità delle principali mansioni da svolgere di ognuno, seguendo una sorta di piano a lungo termine.

Porzioni di progetto sviluppate in gruppo

- Implementazione GameStateImpl;
- implementazione GameViewImpl;
- implementazione ModelImpl.

Le classi ModelImpl e GameStateImpl, dopo averne definito insieme le interfacce, sono "cresciute" parallelamente all'adempimento dei compiti del gruppo.

Sara Cappelletti

- Implementazione del player;
- gestione del salto;
- view con animazione del player;
- gestione delle collisioni (con piattaforme, ostacoli, power-up e monete);
- menù iniziale.

Rachele Margutti

- Gestione dei power-up;
- gestione di Model, View e Controller dello Shop;
- gestione dei vari metodi di pagamento;
- salvataggio e lettura su file per tutti i dati relativi allo shop.

  Per quanto riguarda i power-up è stato creato un package omonimo contenente le cinque classi distinte per ciascun oggetto ed una classe EffectTimer. Quest'ultima si occupa di creare un timer di 10 secondi che verrà sfruttato da tre diversi power-up: Shield, Mushroom e Superjump.

Davide Tonelli

- Creazione e gestione missioni (package mission);
- creazione e gestione marker (package marker);
- implementazione lato controller (classe ControllerImpl);
- creazione di un game loop (classe AnimationTimerImpl);
- lettura e scrittura da file;
- creazione e gestione Statistiche (package statistic);
- implementazione incremento difficoltà (classe StatisticsUpdater);
- gestione input da tastiera (package input);
- creazione e gestione suoni (package sound);
- implementazione delle MissionView, MarkerView;
- implementazione ViewImpl;
- implementazione GameOverViewImpl;

  Inizialmente tra i miei compiti scritti nella proposta di progetto veniva menzionato anche "gestione della mappa infinita", in fase di analisi ci si è resi conto che fosse una funzionalità più facilmente fondibile con la generazione di ostacoli assegnata a Fabio Veroli. Da qui la generazione della mappa scorrevole del nostro videogioco è diventata: il Player è fermo e può solo saltare, mentre ostacoli, piattaforme ed oggetti gli si muovono "contro", rendendo la quantità di model assegnata a me insufficiente. Di seguito ho contattato il Professore Viroli chiedendo direttive, egli mi ha consigliato di aggiungere nuove funzionalità. Ho aggiunto ed implementato il concetto di Mission e Marker nel videogioco, sfruttando la modularità del nostro design. Essendo queste funzionalità aggiunte in seguito, in fase di analisi non è presente nessun riferimento a Mission e Marker.

Fabio Veroli

- Gestione del movimento delle entità e della restante logica;
- gestione della generazione semi-randomica delle entità;

- creazione di Platform;
- creazione di Obstacle e relativo effetto;
- creazione di Coin e relativo effetto;
- view delle entità;
- view delle statistiche.

  Ho implementato tutte le classi/interfacce del package entity (ad esclusione del sotto-package factory e powerup) in autonomia, in particolare dopo aver deciso come gestire la posizione di una DynamicEntity ed aver creato la classe astratta AbstarctDynamicEntity, sono passato allo sviluppo di tutte le entità specifiche. Inizialmente non tutte le entità estendevano dalla stessa classe astratta, ma in fase di sviluppo notando la similitudine tra tutte le entità, io e la mia compagna Rachele Margutti abbiamo deciso che anche i PowerUp (package powerup) sarebbero diventati DynamicEntity. Abbiamo quindi implementato una factory comune usata per creare sia Coin, Platform, Obstacle e loro combinazioni (Fabio Veroli) e PowerUp (Rachele Margutti).

  Decisa la modalità di creazione delle entità mi sono occupato della gestione della loro comparsa a schermo, attraverso l'EntityManager. Sebbene la creazione di entità che implementano tutte la stessa interfaccia, permetta di aggiungerle ad una lista comune anche se di tipi diversi, pone alcune limitazioni nelle possibili configurazioni di gioco. Nonostante abbia deciso la strada più semplice che mi consentisse di generare una lista con diversi tipi di entità, nell'ottica di un progetto futuro sarebbe più opportuno creare diverse liste, in cui almeno le Platform e le altre entità siano separate. In questo modo non avverrebbe più una generazione per "colonne" come nel progetto attuale, ma sarebbe possibile gestire le entità in base alla posizione delle Platform.

3.3 Note di sviluppo

- **Sara Cappelletti**

  Le feature di interesse sono:

  - Uso di lambda expression
  - Uso di librerie di terze parti: JavaFX, per la gestione delle immagini

- **Rachele Margutti**

  Le feature di interesse sono:

  - Uso di Stream
  - Uso di lambda expression
  - Uso di librerie di terze parti: JavaFX, per la gestione delle immagini

- **Davide Tonelli**

  Le feature di interesse sono:

  - Uso di lambda expression
  - Uso di Stream
  - Uso di Optional
  - Uso di Predicate in MissionFactoryImpl
  - Uso di Supplier in MissionFactoryImpl
  - Uso di librerie di terze parti: JavaFX, utilizzata per audio, uso di immagini ed infine per usufruire di AnimationTimer come un game loop.

    Lettura e scrittura da file sono frutto di una ricerca su Internet che mi ha portato ad usare BufferedWriter e BufferedReader.

- **Fabio Veroli**

  Le feature di interesse sono:

  - Uso di lambda expression
  - Uso di Stream e Stream.Builder in EntityManager e EntityFactory
  - Uso di Predicate in EntityManager
  - Uso di librerie di terze parti: JavaFX, per la gestione delle immagini.
  - Creazione del progetto utilizzando Gradle (build.gradle.kts fornito in classe): il progetto è stato convertito da un JavaProject a un GradleProject, in quanto nonostante un iniziale scelta di non usare gradle, abbiamo risontrato problemi nell'aggiungere tutti i JAR di libreria direttamente nel progetto.

    Nota: prima di effettuare la conversione ad un progetto Gradle ho erroneamente caricato nella repository remota su GitHub il contenuto della cartella bin/ di JavaFX, per questo nelle mie statistiche di GitHub il numero di Additions-Deletions è falsato

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Sara Cappelletti

Sono molto contenta di essere riuscita a lavorare quasi sempre insieme ai miei compagni in presenza, sostenersi e aiutarsi nel campi in cui ognuno è un po' più carente si è rivelato molto utile. Essendoci divisi tutte le parti importanti è stato fondamentale collaborare affinché il lavoro procedesse regolarmente e il gioco potesse funzionare correttamente. Seguire lo sviluppo fino in fondo è stato molto soddisfacente anche se faticoso e mi sono resa conto meglio delle mie capacità. Sono soddisfatta della mia parte perchè la ritengo ordinata e ben funzionante. Considerando che ho avuto molte difficoltà iniziali durante il corso non mi sarei aspettata di riuscire a fare in questo modo un progetto che considero complesso.

Rachele Margutti

Questo progetto è stato senza dubbio l'occasione per farmi prendere un po' più di sicurezza e responsabilità per quanto riguarda la scrittura di codice. Non nego di aver avuto un po' di timore all'inizio, ma lavorando spesso in gruppo coi miei compagni e acquisendo un po' di fiducia in me stessa, sono riuscita a completare tutti i miei compiti ritenendomi soddisfatta del lavoro svolto, e ho anche colmato alcune lacune su alcuni elementi di Java. Essendo poi io un'appassionata di videogiochi, grazie allo sviluppo di questa applicazione ho capito che sarebbe un campo che mi piacerebbe approfondire, sono quindi molto contenta di aver potuto svolgere questo progetto in questo punto della mia carriera.

Davide Tonelli

Mi ritengo ragionevolmente soddisfatto del lavoro da me svolto, in quanto ho sentito durante tutto il percorso di progettazione un miglioramento continuo delle mia capacità sia a livello di progettista che di programmatore java. Il mio punto di forza è stato l'approccio utilizzato, scrivere su carta ed una volta compreso bene il funzionamento tradurre la logica in codice, mi ha portato ad aggiungere efficientemente nuove funzionalità al programma mantenendo spesso intatta l'architettura precedentemente ideata. Valutando a posteriori il mio lavoro riconosco di aver avuto inizialmente timore nell'approfondire JavaFX in quanto a me una fonte totalmente sconosciuta, fatto che mi ha portato a prendere decisioni che poi mi si sono rivoltate contro. In futuro eviterò di farmi intimorire da un problema all'inizio apparentemente insormontabile dedicando il tempo necessario per comprendere gli strumenti da utilizzare. Inizialmente il mio ruolo, avendo una grande fetta di controller, è stato quello di creare bozze di design architetturali da sottoporre successivamente ai compagni di progetto, discutendo insieme la scelte da compiere. Una volta completato ciò ho sempre cercato, come i miei compagni, di portare ordine gestendo una "tabella di marcia" dei compiti che ognuno dovesse svolgere.

Fabio Veroli

Mi ritengo soddisfatto del lavoro eseguito per quanto concerne la mia parte di progetto, anche se tornando indietro adotterei delle soluzioni diverse ad alcuni problemi che si sono presentati durante lo sviluppo. Il progetto mi ha aiutato a chiarire alcuni aspetti del corso che mi erano poco chiari. Valuto positivamente la mia esperienza in gruppo, nonostante solitamente sia diffidente al lavorare in gruppo, penso sia stata fondamentale la comunicazione e discussione che si è instaurata durante lo sviluppo. Cimentarmi nelle basi del game developing ha confermato lo scarso interesse che nutro per questa branca dell'informatica, spero quindi in futuro di partecipare allo sviluppo di progetti di altra natura.

4.2 Difficoltà incontrate e commenti per i docenti

Il gruppo inizialmente si è trovato davanti a una grande difficoltà, in quanto tutti eravamo al primo progetto di programmazione. Ci siamo trovati con

molte lacune nelle conoscenze necessarie per il progetto, tra cui un'approccio iniziale alla programmazione di un videogioco.

Appendice A

Guida utente

Figura A.1: Schermata iniziale del gioco

Quando il gioco viene avviato compare la schermata inziale.
Il pulsante Start permette di avviare la partita.
Il pulsante Exit permette di chiudere il gioco.

Figura A.2: Schermata di gioco

In alto a sinistra compaiono le statistiche delle monete raccolte in questa partita e le vite rimaste.

In alto a destra è presente una missione, se completata aggiunge 20 monete al totale già raccolto.

Durante il gioco compariranno le prossime figure:

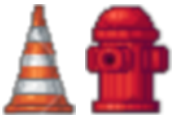

Figura A.3: Ostacoli cono stradale e idrante

Questi sono i due ostacoli da evitare saltando

Figura A.4: Piattaforma

Questa è la piattaforma su cui il Player può saltare, può comparire su due diversi livelli

Figura A.5: Moneta

Questa è una moneta raccoglibile, può comparire su tre diversi livelli

Figura A.6: Marker Swag Cat e palloncino

Questi Marker indicano la distanza raggiunta, non provocano collisioni

Figura A.7: Tomba

Questa figura rappresenta il punto dell'ultima morte, non crea collisioni

Figura A.8: Bandierina

Questa figura rappresenta il record massimo raggiunto

Figura A.9: Power-up vita extra

Questo power-up incrementa di uno le vite del giocatore

Figura A.10: Power-up funghetto

Questo power-up raddoppia il valore delle monete raccolte

Figura A.11: Power-up scudo

Questo power-up rende immuni alle collisioni con ostacoli

Figura A.12: Power-up bomboletta spray

Questo power-up elimina tutte le entità presenti nello schermo

Figura A.13: Power-up supersalto

Questo power-up moltiplica per 1.5 l'altezza del salto

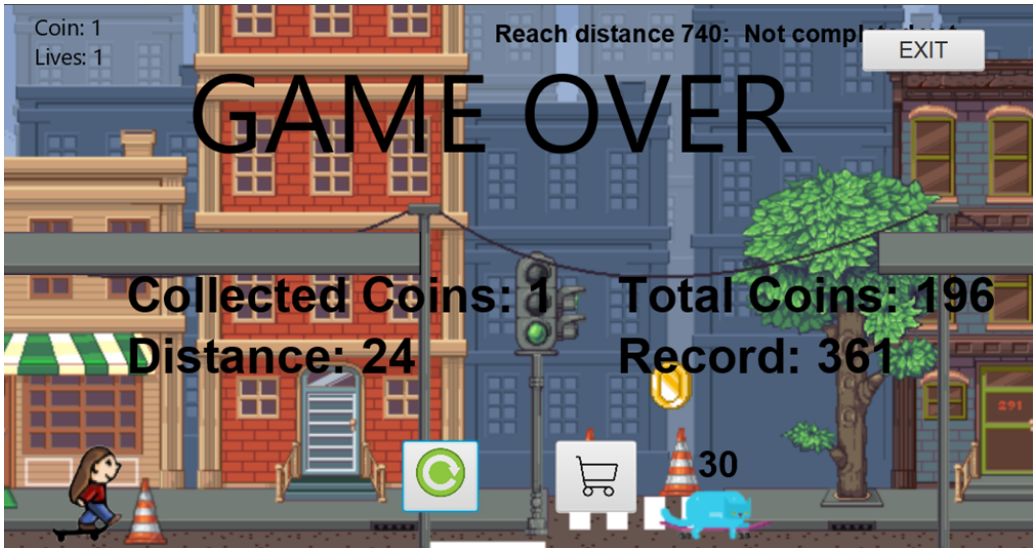

Figura A.14: Schermata di Game Over

Quando si muore compare questa schermata.

Le scritte centrali sono le statistiche di gioco: le monete raccolte in questa partita e le totali, la distanza raggiunta in questa partita e il record

Il tasto in altro a destra permette di uscire.

Il tasto centrale sinistro permette di iniziare una nuova partita.

Il tasto centrale dentro porta allo shop.

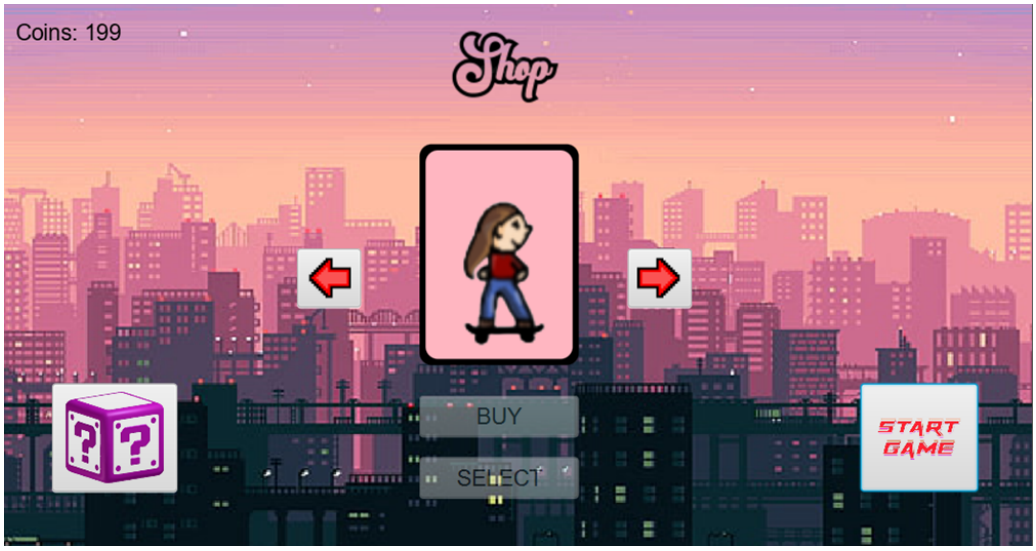

Figura A.15: Schermata dello Shop

In alto a sinistra sono riportate le monete totali.

Nel centro sono presenti le skin acquistabili e già acquistate, le frecce servono per scorrerle.

Il tasto Buy permette di comprare la skin, è attivo se il giocatore possiede abbastanza monete (1000 per il Programmatore e 2000 per il Dinosauro).

Il tasto Select permette di selezionare la skin con cui si vuole giocare, sarà attivo una volta acquistata la skin.

Il tasto in basso a sinistra è una Mystery Box (costa 200 monete) e permette di vincere uno fra tre premi: 300, 150 o 0 monete.

Il tasto in basso a destra permette di iniziare una nuova partita.

Appendice B

Esercitazioni di laboratorio

B.0.1 Sara Cappelletti

- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p136429
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p137743
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136320
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p138721
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138893
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p140234

B.0.2 Rachele Margutti

- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p136434

B.0.3 Davide Tonelli

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136874

- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138829
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139999

B.0.4 Fabio Veroli

- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p141384
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p141247
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p138752