Relazione Space Invaders

Riccardo Azzali
Michele Pasi
Enrico Baroni
Alberto Rossi

25/04/2021

Sommario

Questo documento `e una relazione del progetto di OOP (Object-Oriented Programming) dell'anno accademico 2020-21.

Lo scopo di questo documento è quello di descrivere i punti principali della progettazione e del funzionamento dell'applicazione. Tale descrizione sarà suddivisa in diverse sezioni e sottosezioni in modo da partizionare le varie fasi di sviluppo del sistema applicativo. Per ciascuna delle sezioni del documento sarà fornita una descrizione generale o particolare della corrispettiva fase di analisi, progettazione o implementazione.
Il modello della relazione segue il processo tradizionale di ingegneria del software fase per fase (in maniera ovviamente semplificata).

Indice

Capitolo 1

Analisi

1.1 Requisiti

Questo software tenta di replicare il famosissimo videogame arcade anni ’70 **“Space Invaders”**. Un videogioco arcade è un videogioco che si gioca in una postazione pubblica apposita a gettoni o a monete, costituita fisicamente da una macchina posta all'interno di un cabinato. Gli arcade rappresentano la prima generazione di videogiochi di largo consumo. In Space Invaders, l’utente controlla una navicella aerospaziale la quale ha lo scopo di sopravvivere ai vari nemici che si presentano in differenti livelli.

Requisiti funzionali

- L’applicazione garantisce il movimento della navicella, attraverso le frecce direzionali, all’utente nella finestra di gioco.
- Sono presenti differenti tipologie di nemici, che cercano di sconfiggere la navicella del giocatore colpendola con colpi di arma o tentando di scontrarsi con essa.
- La partita termina quando le navicelle nemiche raggiungono il fondo dello schermo decretando l'avvenuta invasione o quando si esaurisce la vita disponibile.
- Durante il corso del gioco è possibile sbloccare dei power-up aggiuntivi al giocatore.
- E’ presente la musica durante il gioco.
- E’ presente un menu grafico composito di vari sottomenu i quali guidano l’utente nell’esplorazione delle funzionalità del gioco.
- A partita terminata verrà memorizzato in una classifica il punteggio totalizzato dall’utente in base alle azioni compiute in gioco. I punteggi sono visualizzabili graficamente in una tabella dei migliori giocatori.

Requisiti non funzionali

- Il gioco potrebbe supportare varie lingue.
- Potrebbe essere aggiunta una modalità di gioco *Sopravvivenza*, ove l'utente gestisce la navicella in modo macchinoso differentemente dalla modalità di gioco principale.

1.2 Analisi e modello del dominio

Il sistema applicativo avrà varie tipologie di entità, che dovranno rispettare i parametri definiti nell'interfaccia Entity che rappresenta il fulcro base di tutte loro:

- il personaggio principale (player);
- i proiettili del personaggio principale
- le navicelle nemiche;
- i proiettili delle navicelle nemiche;
- i power-up;
- le meteore.

L'interfaccia Game è la base logica del gioco e mantiene al suo interno tutti i riferimenti alle entità prima citate. Questa interfaccia, inoltre si occuperà di controllare le collisioni, in particolare tra il player e tutte le altre entità.

Durante lo svolgimento del gioco il personaggio può utilizzare determinati power-up, per esempio: scudo protettivo, vita bonus e il freeze delle navicelle nemiche.

Il sistema prevede lo scontro tra le diverse entità:

- i proiettili del giocatore si possono scontrare con le navicelle nemiche e viceversa, ma anche contro le meteore;
- le navicelle nemiche possono collidere contro il giocatore;
- il giocatore può scontrarsi contro le meteore;
- il giocatore può ottenere power-up scontrandosi con loro.

All'interno di Game ci sarà un riferimento a Level per poterci permettere di sapere a che livello il giocatore è arrivato.

Le difficoltà che potrebbero presentarsi nello sviluppo del software consistono nell'identificazione delle collisioni fra entità, ottimizzazione nel caso di presenza di un numero elevato di entità, raccolta di power-up, creazione casuale delle meteore.

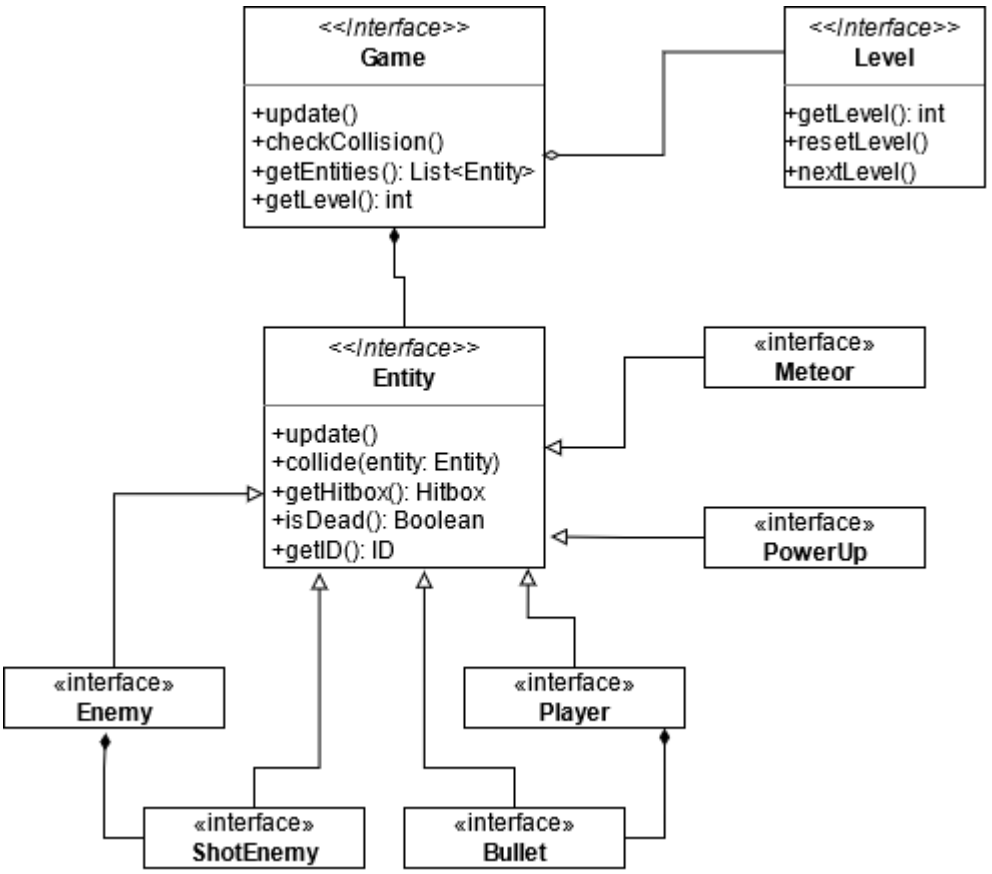


Figura 1: Schema UML con le principali entità.

Capitolo 2

Design

2.1 Architettura

L'architettura della nostra applicazione segue il pattern MVC (Model-View-Controller).

Il main del gioco crea la view e il controller.

Il controller gestisce l'applicazione, comunica sia con la view, a cui invia periodicamente l'ordine di aggiornarsi e ridisegnare le entità, sia con il game da cui ottiene informazioni sullo stato del gioco. In particolare è vitale che il controller possa ottenere informazioni riguardo alle entità presenti in quel momento, al punteggio e al livello, per poterle passare alla view e quindi aggiornarla.

Abbiamo utilizzato il controller come observer, la view infatti non può comunicare direttamente con il controller, quello che succede è che la view notifica al controller l'input ricevuto ed è poi lui a decidere cosa fare.

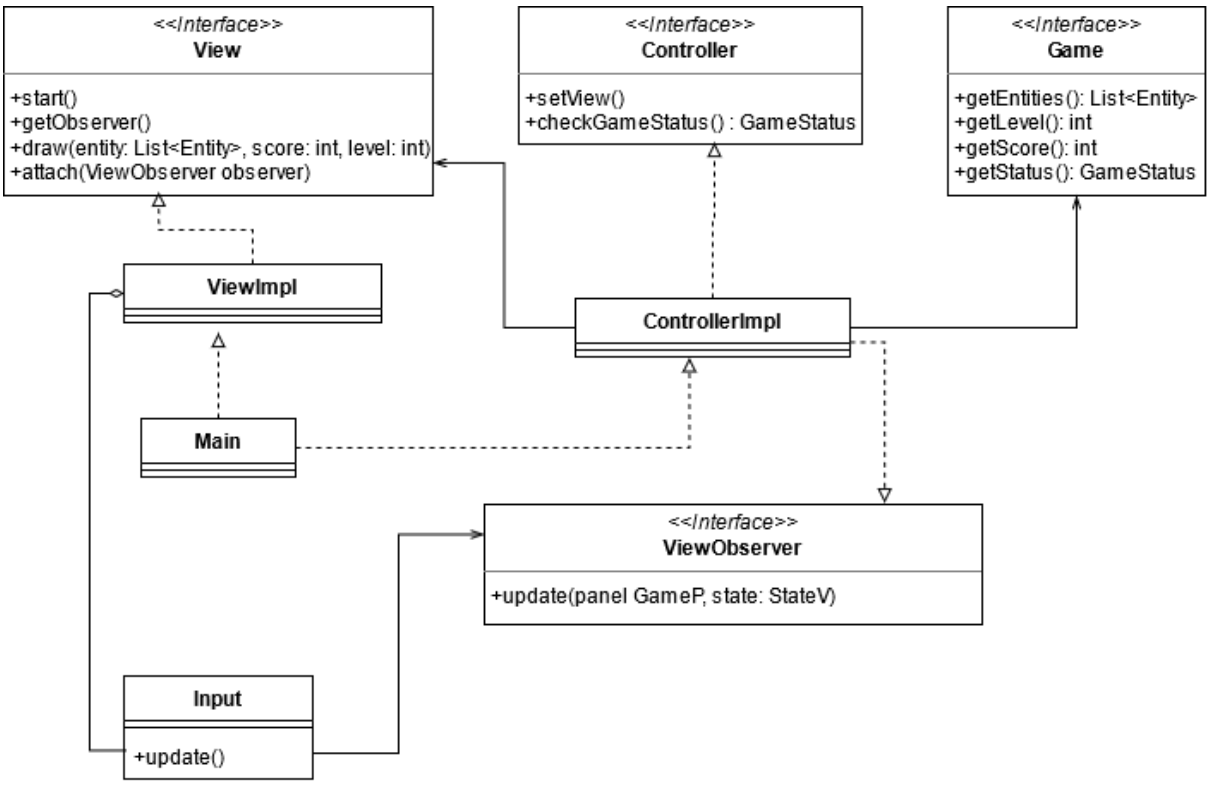


Figura 2: Schema UML dell'architettura.

2.2 Design Dettagliato

Riccardo Azzali

Nel progetto mi sono occupato delle classi PlayerImpl, Input, e GameLoop con rispettive interfacce. Le classi e le interfacce GameLoop e Input si trovano all'interno del package Controller mentre la classe PlayerImpl e la sua interfaccia nel package del model.

La classe PlayerImpl estende la classe EntityImpl di cui usa tutte le funzioni, ereditando così tutte le caratteristiche di un'entità normale come le coordinate e le modalità per accedervi e modificarle, la possibilità di morte, i controlli sulle collisioni e l'update, mentre implementa i suoi metodi, cioè quelli che gli permettono di interagire con i power up e la barra della salute, dall'interfaccia omonima.

Sarà poi gestito nella classe di implementazione del game (la classe GameImpl) che invocherà i metodi collide e update quando necessario, mentre si muoverà grazie alla classe Input.

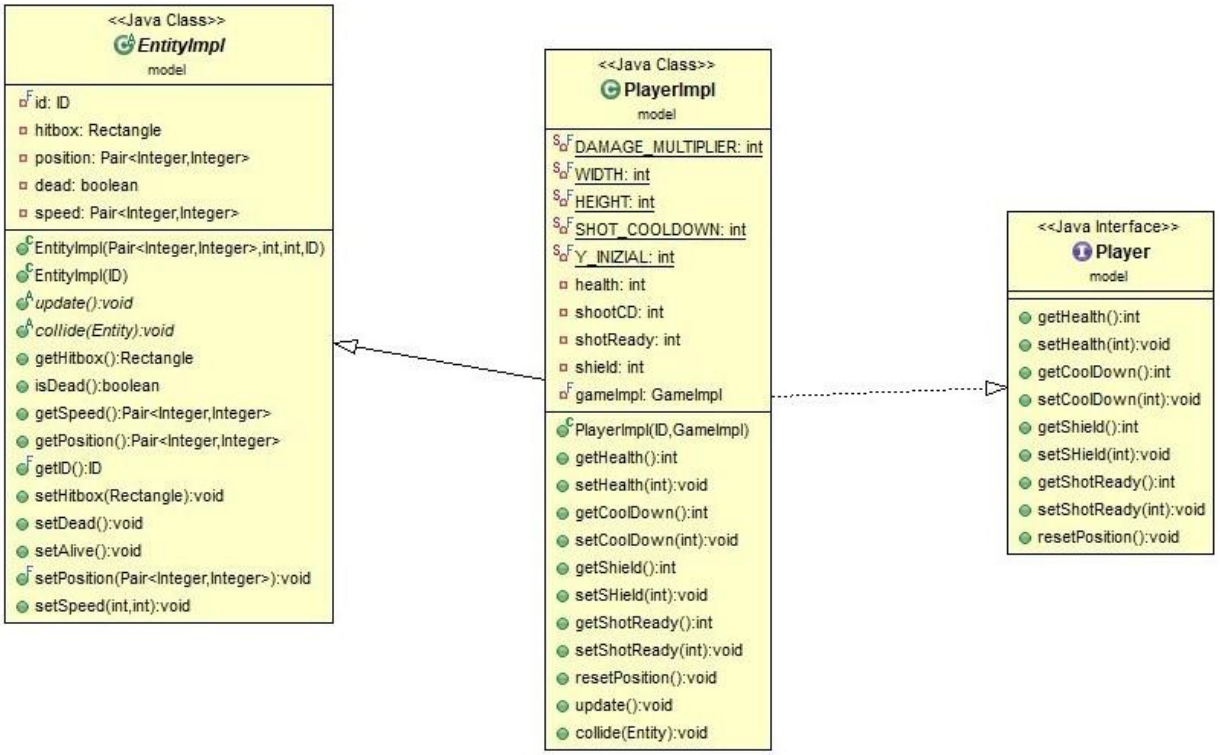


Figura 2.1: Schema UML Player Implementation

La classe Input si occupa di gestire l'input del Player durante la partita. Quando La classe viene costruita ha bisogno del Game per accedere al Player e del viewObserver per conoscere lo stato di gioco(ad esempio se il gioco è in pausa). E' un estensione della classe preesistente in java KeyAdapter che ci permette di svolgere diverse operazioni al verificarsi di alcuni keyEvent. Vediamo due metodi keyPressed e keyReleased che vanno a memorizzare lo stato dei tasti disponibili per i Player negli array keyDownPlayerX. Verrà poi verificato nel metodo update lo stato dei vari tasti e l'aggiornamento della velocità del player di conseguenza. La classe gestisce inoltre l'aggiunta di Bullet al game, verifica che il Player abbia un Bullet a disposizione e se il tasto è premuto ed il Bullet è pronto, lo aggiunge all' apposita lista. Da notare che il KeyInput lavori in maniera molto simile ad un vero e proprio observer che favorisca la comunicazione tra player e il keyListener.

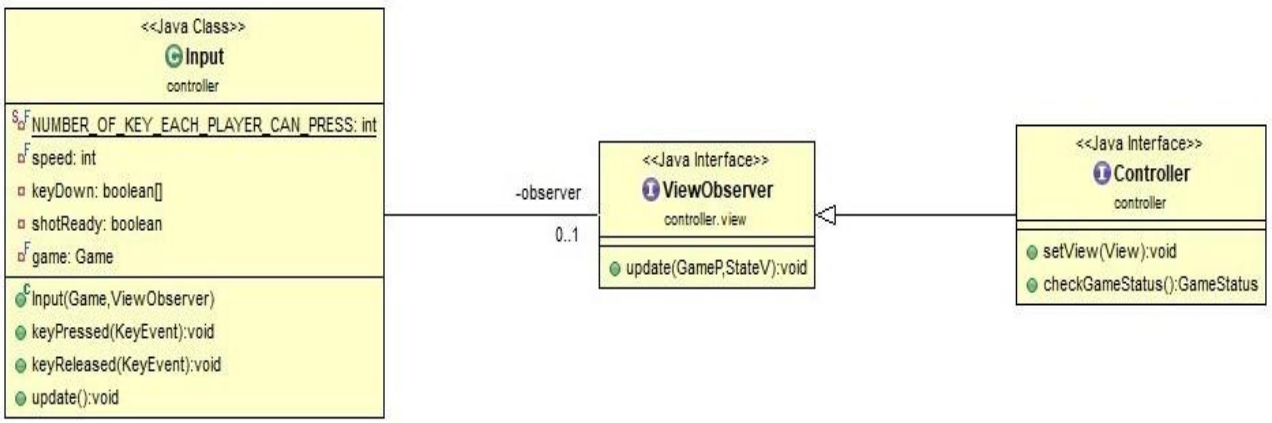


Figura 2.3: Schema UML dell' Input

Il gameLoop è il cuore del gioco e garantisce che lo stato del game e quello della view sia aggiornato. La sua struttura di base è standard, tanto che alcuni lo considerano un pattern di progettazione per lo sviluppo di giochi. E' parte del controller ed è l'unica classe attiva, fatta eccezione per la view. Anche se non efficiente per giochi più complessi, ho usato un solo thread, quello del gameLoop, per eseguire tutte le azioni necessarie per aggiornare il game essendo sufficiente per gestire fino a venti/trenta entità contemporaneamente.

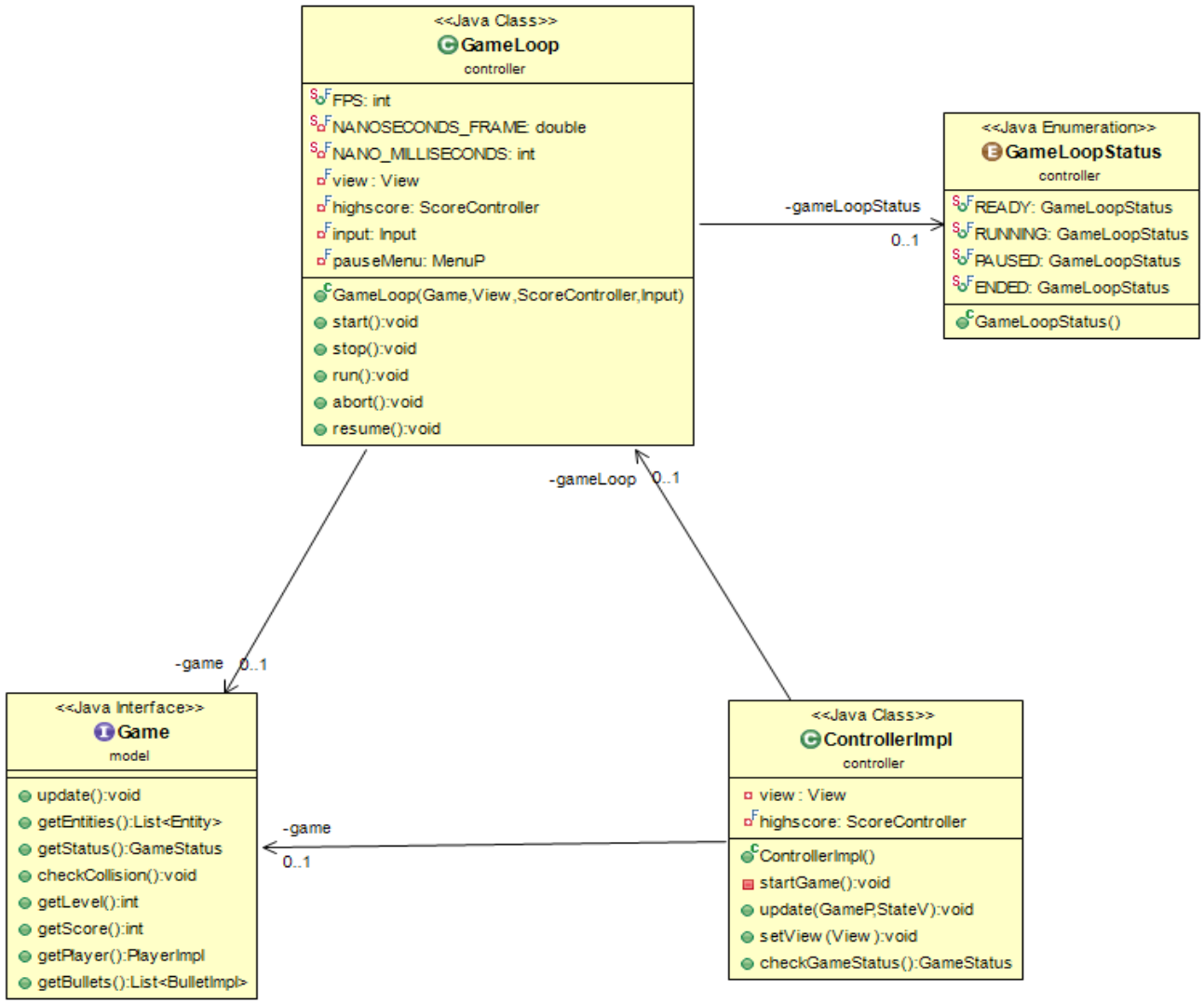


Figura 2.4: Schema UML del GameLoop

Michele Pasi

In questo progetto mi sono occupato principalmente della gestione dei nemici

La gestione delle navicelle nemiche corrisponde all'unico e vero avversario del giocatore.

Le navicelle nemiche sono di due tipologie:

- Navicella Base, si tratta del nemico più frequente e lo ritroviamo in tutti i livelli. La loro peculiarità è che sparano esclusivamente in basso
- Navicella Boss, si tratta di un nemico più potente e lo ritroviamo ogni 5 livelli. La sua peculiarità è che spara in tre direzioni: In basso, in basso a destra e in basso a sinistra

Ogni navicella base, boss e sparo estendono sempre la classe Entity ed i suoi metodi collide e update che gestiscono rispettivamente: collisione tra i vari oggetti e il comportamento di ogni singolo oggetto.

Per riuscire a gestire al meglio entrambi le classi nemiche è stato creato un' sistema gerarchico che vede l'interfaccia principale Enemy in cima alla scala, in secondo piano vediamo la classe astratta, AbstractEnemy, che implementa l'interfaccia ed infine tutte le classi che estendono la classe astratta.

Di seguito il diagramma delle classi esplicativo della scala gerarchica creata per lo sviluppo dei nemici

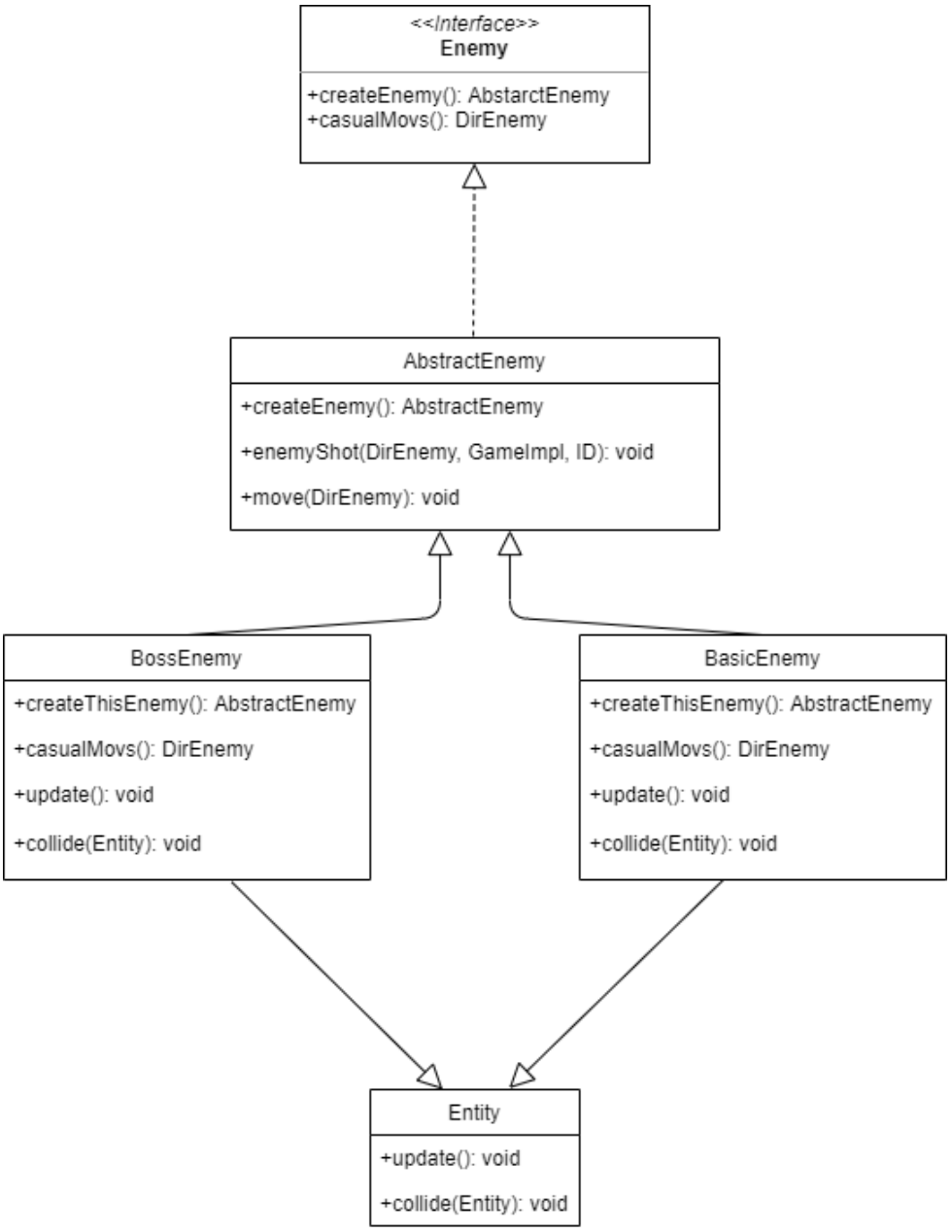


Figura 2.5: Schema UML che mostra il sistema gerarchico per la gestione e lo sviluppo dei nemici

Nella figura 2.5, inoltre, si noti che all'interno della classe astratta (AbstractEnemy) si trova un metodo per la creazione di oggetti tipo enemyShot, quest'ultima è l'unica in grado classe di assolvere a questo compito.

Lo Sparo delle navicelle nemiche, quindi, è gestito dalla classe ShotEnemyImpl che implementa l'interfaccia principale ShotEnemy e che, come detto precedentemente, estende la classe EntityImpl. Ogni navicella nemica sarà quindi in grado di sparare esclusivamente verso il basso mentre si muove in tutte le sue direzione, se il colpo andrà a buon fine, il giocatore perderà della vita, se arriverà invece in fondo alla schermata di gioco svanirà.

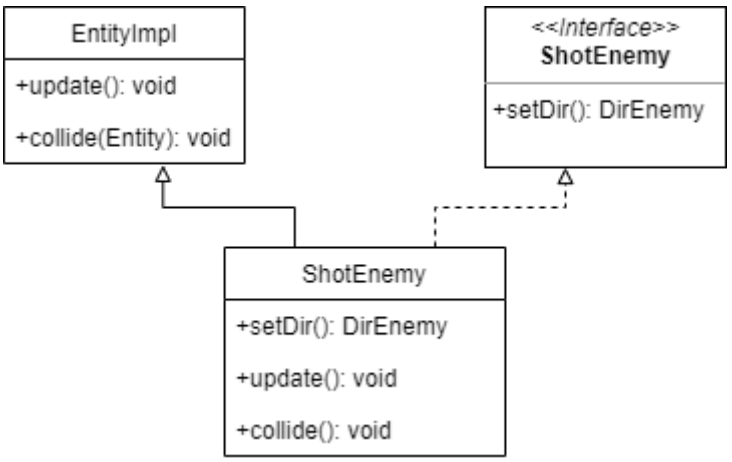


Figura 2.6: schema UML che mostra la gestione e lo sviluppo dello sparo delle navicelle nemiche

Alberto Rossi

In questo progetto mi sono occupato dell'implementazione dei PowerUp e dell'interfaccia del menù di gioco.

Power Up

Per la creazione dei PowerUp ho pensato ad una gerarchia fatta di interfaccia PowerUp che viene implementata dalla classe astratta PowerUpImpl che a sua volta vengono estese da PPowerUp e GPowerUp.

In PPowerUp potenzia il player con vari bonus, invece il GPowerUp contiene un bonus che influisce sull'intero gioco.

In questo modo ho potuto riconoscere che i PowerUp hanno in comuni vari aspetti non replicate nelle varie classi grazie a questa architettura.

Le classi due classi sono due:

- la prima gestisce l'aggiornamento dei PowerUp in base al tipo di quest'ultimo
- il secondo ha il compito di decidere se è necessario applicare il powerUp, visto che potrebbe essere già attivo.

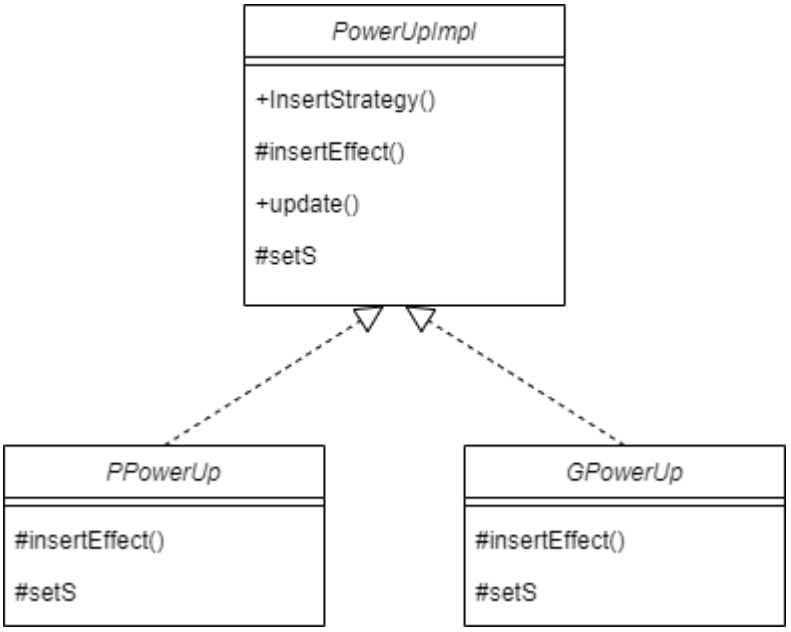


Figura 2.7: UML Method applicato all'aggiornamento e all'effetto dei powerUp.

Per quanto riguarda l'efficacia dei powerUp nei vari livelli del gioco ho utilizzando il Pattern Strategy che permette di aumentare l'efficacia in base al livello raggiunto.

In questo modo rendiamo più efficaci i powerUp in diversi momenti di gioco, dal primo all'ennesimo livello

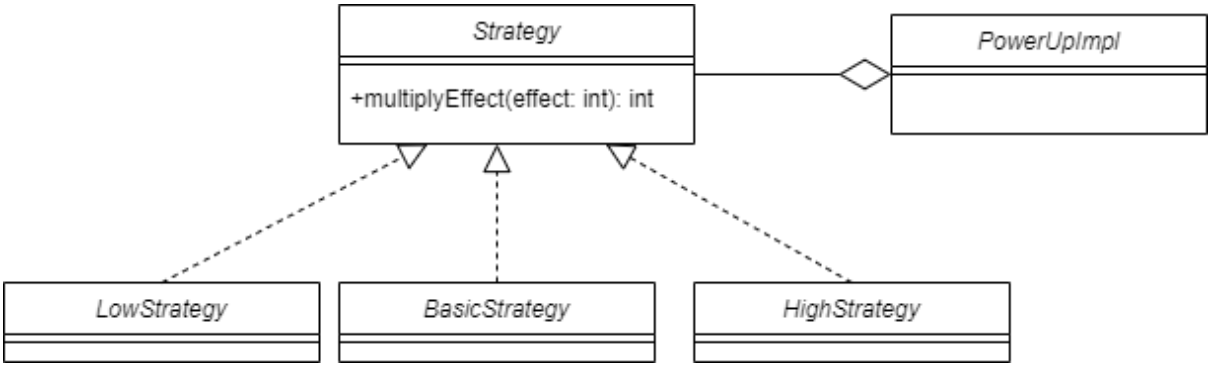


Figura 2.8: UML del pattern Strategy applicato all'aggiornamento e all'effetto dei powerUp.

Mi sono occupato anche della gestione delle varie view di gioco tra le quali il menù e i vari sotto menù.

Per poter poter rispettare l'architettura MVC ho deciso di utlizzare il patter Observer.

il controller è l'Observer e può essere attaccato ad un observabile, la view. In concomitanza di eventi particolari la view notifica il controller che modifica in maniera rapida e senza problemi.

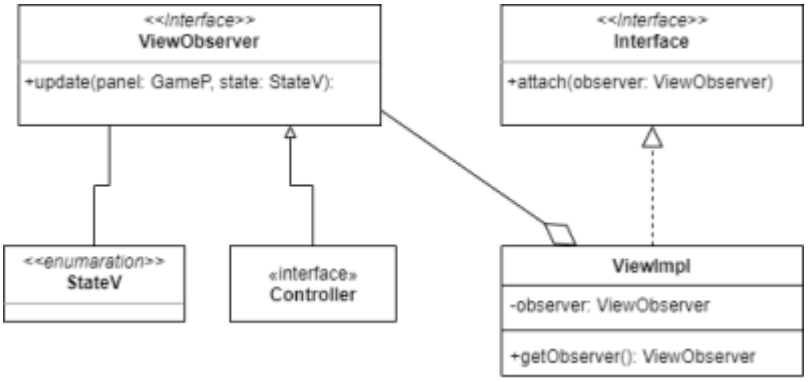


Figura 2.9: Pattern Observer tra view e controller

Per la gestione delle immagini ho creato una classe chiamata ImageLoader, che permette il caricamento di tutte le immagini all'apertura del gioco, cercando di richiedere meno risorse possibili durante l'esecuzione del game play.

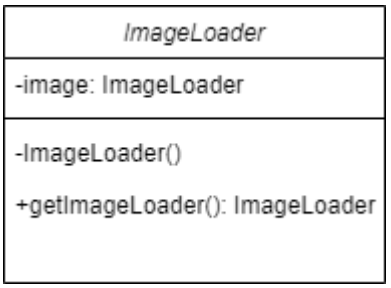


Figura 2.10: Uml classe Image Loader

Enrico Baroni

In questo progetto mi sono occupato della classe delle meteore e della gestione degli highscore.

Per quanto riguarda la gestione delle meteore ho deciso di modellare un movimento dall'alto verso il player, con un danno proporzionale al relativo livello raggiunto dallo stesso player.

La meteora estende la classe Entity ed i suoi metodi collide e update che gestiscono rispettivamente: collisione tra i vari oggetti e il comportamento di ogni singolo oggetto.

La gestione della meteora è gerarchica per non creare classi troppo complesse, così facendo può essere relativamente semplice aggiungere nuove tipologie di meteore con caratteristiche di comportamento diverse.

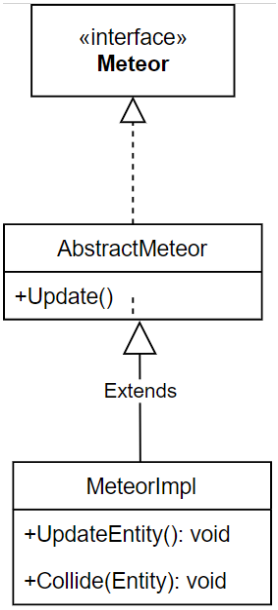


Figura 2.11: Uml model delle meteore

Infine per la creazione delle meteore ho creato una classe nella parte di controller chiamata MeteorController che gestisce la creazione appunto delle meteore impostando con un random una posizione casuale nell'arena di gioco e una velocità e una grandezza standard per la meteora.

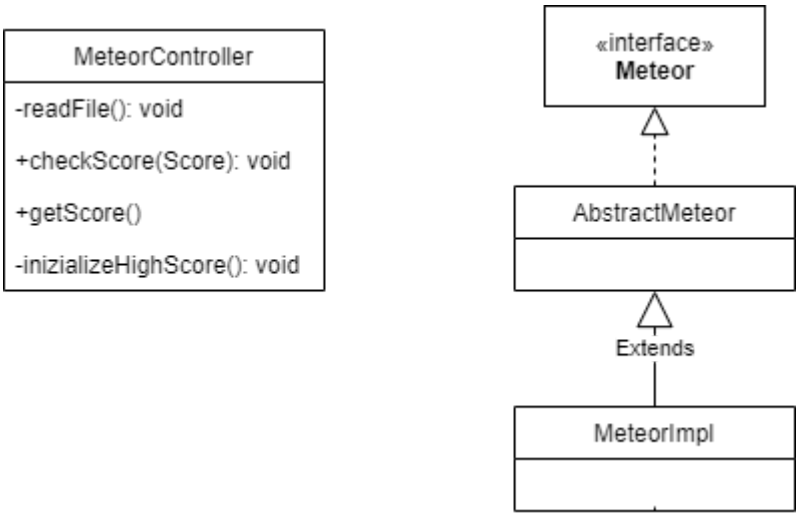


Figura 2.12: Uml controller delle meteore

Così facendo è più facile la creazione delle meteore dalla classe Level sviluppata dal mio collega.

Invece per lo sviluppo del salvataggio su file dei migliori score. Alla fine di ogni partita viene richiamato il metodo checkScore della classe ScoreController che controlla tutti i

migliori score delle scorse partite e se abbiamo un nuovo record viene salvato sul file Score.txt.

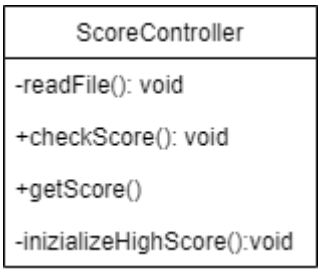


Figura 2.13: Uml Controller del salvataggio su file

Il file salverà e ordinerà i migliori 10 punteggi del player per poi renderli visibili nel menu principale sotto la voce Score.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In comune accordo abbiamo scelto tutti e quattro di utilizzare come sistema operativo Windows e necessitando di verificare il corretto funzionamento del gioco su un altro sistema operativo (Linux), abbiamo provveduto a portare il Jar su macchina virtuale e testarlo manualmente.

Il sistema operativo utilizzato per il test: Ubuntu 16.04.02

Il test su macchina virtuale è stato condotto da Alberto Rossi

Inoltre è stata sfruttata JUnit per testare funzionalità specifiche del software. In particolare:

Alberto Rossi: Test dei PowerUp (Health, Shild, Generale) e dell'ImageLoader.

Enrico Baroni: Test delle meteore e la collisione con i proiettili

Michele Pasi: Test dei nemici, sia Basic che Boss (movimento ed eliminazione) e dello shot sempre del nemico (movimento ed eliminazione)

Riccardo Azzali: Test del Player (creazione, movimento) e del Bullet (creazione e movimento)

I test inerenti alla view sono stati fatti manualmente per verificarne il corretto funzionamento.

3.2 Metodologia di lavoro

Molte parti sono state sviluppate con la collaborazione di tutti i membri, nonostante non sia mai stato possibile vedersi di persona causa COVID19. Inoltre ci siamo aiutati vicendevolmente per la risoluzione di bug e problemi non banali come crash o freeze della view del gioco.

Durante la fase di analisi abbiamo stabilito alcune interfacce base (quelle principali, cioè le entità in gioco) in modo tale da facilitare il processo di integrazione. A quel punto abbiamo programmato in maniera individuale ma rimanendo sempre in contatto durante le sessioni di coding attraverso la piattaforma Discord per permetterci rapidi chiarimenti riguardo alle interazioni tra le varie classi. Quando poi abbiamo iniziato ad unire i nostri lavori ci siamo accorti che alcune cose non erano fatte in maniera corretta

e abbiamo dovuto modificare varie classi. Questi cambiamenti ci sono costati molto tempo. Per risolvere i suddetti problemi è stato necessario discutere tutti assieme su come risolverli definitivamente e il più rapidamente possibile.

Siamo stati in grado di lavorare a distanza mediante l'uso di git. Inizialmente nessuno di noi era esperto nel suo utilizzo ma dopo vari di fallimenti siamo riusciti ad utilizzarlo. In particolare Enrico Baroni è stato il membro a cui rivolgersi in caso di problemi con git.

3.3 Note di sviluppo

Riccardo Azzali

Io mi sono occupato della creazione del Player, del GameLoop e dell'Input.

Innanzitutto ho iniziato creando il GameLoop che essendo praticamente uno standard per nello sviluppo di giochi mi è stato facile trovare molteplici esempi da cui prendere spunto.

Per la classe Player mi è bastato estendere dalla classe EntityImpl che gestisce tutte le entità del gioco e poi creare prima la sua interfaccia Player con i metodi per il movimento e per lo sparo, che poi sono stati richiamati nella classe dell'Input.

Ho cercato di curare il più possibile la pulizia del codice e la sua leggibilità piuttosto che cercare di fare cose particolarmente complicate. Ho tuttavia incontrato qualche difficoltà con l'implementazione del sistema di Input per il quale ho usato un misto delle mie conoscenze e pezzi di codice su vari siti, https://stackoverflow.com primo fra tutti. Ho cercato inoltre di adattarmi il più possibile alle esigenze dei miei compagni, a volte stravolgendo alcune classi che avevo già pronte

Michele Pasi

Io mi sono occupato della creazione dell' Entity e della creazione di tutto ciò che riguarda i nemici.

Per prima cosa ho creato l'Interfaccia Entity che viene implementata nella classe astratta EntityImpl, adatta a gestire tutte le entità del gioco.

Successivamente mi sono focalizzato sui nemici creando l'interfaccia principale Enemy e la classe astratta AbstractEnemy, che si è rivelata di fondamentale importanza perché per lo sviluppo dei nemici e dello sparo ho sempre utilizzato i metodi all'interno di quest'ultima

Alberto Rossi

Io mi sono occupato principalmente dell'interfaccia della view e dei powerUp.

Durante l'implementazione della mia parte ho cercato di mettere a fuoco le skill imparate durante il corso seguito come: la lambda e degli stream per sostituire i vari cicli tradizionali.(for, do while, while)

Ho inoltre sfruttato due layout particolari (CardLayout e GroupLayout) per realizzare la parte di View e la classe BufferedImage per riuscire a caricare delle spriteSheet.

Per l'implementazione di alcune classi come ImageLooader mi sono avvalso dell'aiuto di vari siti come https://stackoverflow.com (il nostro salvatore)

Enrico Baroni

Io mi sono occupato della creazione delle meteore, dello score, del salvataggio su file dei migliori score e dell'interfaccia grafica del player.

Per lo sviluppo delle meteore ho rispettato le linee guida decise con i miei colleghi per lo sviluppo di entità nel gioco.

Invece per lo score ho deciso di dare importanza al livello raggiunto dal player infatti ho sviluppato il calcolo del punteggio nella classe implementata dal mio collega e poi per il salvataggio su file mi sono affidato a vari siti per creare il mio codice.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Riccardo Azzali

Nel complesso ritengo che il progetto sia stato gestito in maniera buona. Siamo arrivati abbastanza lunghi in quanto il lavoro si è stato rallentato dalla problematica di non poterci incontrare mai realmente in presenza

Nella mia parte del progetto ritengo che il gameLoop, il Player e la parte dell'input siano venute bene. Sono invece insoddisfatto di come è venuta la parte di View, probabilmente anche a causa del non utilizzo di JavaFX.

Michele Pasi

Il progetto, nonostante le difficoltà riscontrate tante delle quali dovute al fatto di non poterci incontrare, è ben riuscito. Abbiamo svolto il lavoro con massima serietà e dedizione ciò ci ha consentito di risolvere i problemi e le difficoltà.

Mi ritengo quindi soddisfatto della riuscita del gioco, in particolar modo della mia parte poiché son riuscito ad implementare nemici e sparo come si era stabilito in precedenza.

In futuro si potrebbero apportare migliorie al gioco ad esempio:

- Diversi Boss, con diverse modalità di sparo
- Nemici particolari che dopo la morte erogano punteggio raddoppiato
- Aggiungere effetti sonori
- Aggiungere diversi PowerUp
- Aggiungere, come nel cabinato, blocchi fissi

Alberto Rossi

Posso iniziare dicendo che in questo periodo di pandemia è stato abbastanza difficoltoso e frustrante non poter incontrarci di persona per poter realizzare questo progetto. Siamo arrivati abbastanza lunghi poichè io personalmente ho avuto problemi di salute che mi hanno rallentato abbastanza.

Sono abbastanza soddisfatto di ciò che abbiamo creato anche se ovviamente si poteva fare meglio, ma credo che come primo progetto di questo genere ce la siamo cavata.

Enrico Baroni

Il progetto sotto il mio punto di vista è un'ottima partenza per imparare lo sviluppo di programmi in gruppo.

Da questa esperienza ho imparato che lavorare in gruppo è molto importante e soddisfacente, ho capito che ascoltare gli altri ti stimola la conoscenza e diventa più facile risolvere i problemi tramite il confronto.

In questo periodo non è stato facile lavorare tutti insieme perché è mancato il potere vedersi e l'interagire solo tramite computer sicuramente è più limitativo. ma la ritengo in tutto e per tutto una bella esperienza.

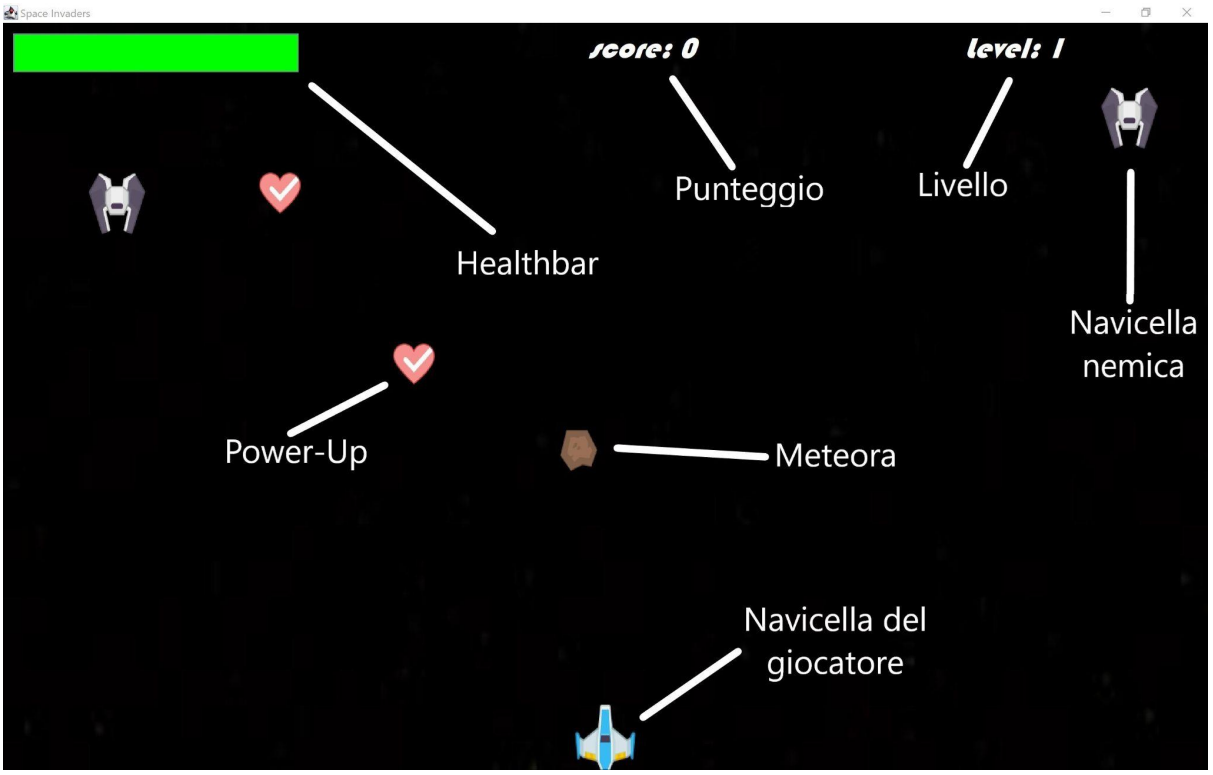
Appendice

Guida giocatore

All’avvio del gioco il menu permette di iniziare una nuova partita, consultare i comandi

di gioco e visualizzare i migliori punteggi.

Avvio partita



All'avvio della schermata di gioco il giocatore ha il controllo della navicella posta in basso, i tasti per gestirne i movimenti sono ← (freccia direzionale sx) →(freccia direzionale dx) e SPACE per lo sparo e P per la pausa.

Lo scopo del gioco è quello di ottenere il punteggio migliore. Le entità presenti si suddividono in:

- Nemici (Basic e Boss)
- Ostacoli (Meteore)
- Power-Ups (Health, Freeze, Shield e Fast Shot)

Le munizioni del giocatore sono illimitate. La partita termina quando o la barra della vita del giovatote arriva a zero oppure quando una qualsiasi navicella nemica

raggiunge il fondo dello schermo.