

BloonsTowerDefense

“Programmazione ad Oggetti”

Alijon Batusha

Stiven Gjinaj

Daniele Martignani

Riccardo Penazzi

Agosto 2023

Indice

1 Analisi	3
1.1 Requisiti	3
1.2 Analisi e modello del dominio	4
2 Design	6
2.1 Architettura	6
2.2 Design dettagliato	7
2.2.1 Alijon Batusha	7
2.2.2 Stiven Gjinaj	8
2.2.3 Daniele Martignani	10
2.2.4 Penazzi Riccardo	14
3 Sviluppo	17
3.1 Testing automatizzato	17
3.2 Metodologia di lavoro	18
3.2.1 Alijon Batusha	18
3.2.2 Stiven Gjinaj	18
3.2.3 Daniele Martignani	19
3.2.4 Penazzi Riccardo	19
3.3 Note di sviluppo	19
3.3.1 Alijon Batusha	19
3.3.2 Stiven Gjinaj	20
3.3.3 Daniele Martignani	20
3.3.4 Penazzi Riccardo	20
4 Commenti finali	22
4.1 Alijon Batusha	22
4.2 Stiven Gjinaj	22
4.3 Daniele Martignani	22
4.4 Riccardo Penazzi	23

Capitolo 1

Analisi

1.1 Requisiti

L'obiettivo del progetto è quello di realizzare un videogioco in stile tower defense ispirato a Bloons TD 5(https://bloons.fandom.com/wiki/Bloons_TD_5). Lo scopo del giocatore è quello di impedire ai nemici, i quali percorrono percorsi prestabiliti, di raggiungere la fine del percorso posizionando in modo strategico torri che infliggono danno ai nemici. Se un nemico raggiunge la fine del percorso il giocatore perde una vita. Ogni volta che un nemico viene ucciso il giocatore riceve monete che può spendere per aggiungere difese oppure eseguire upgrade a quelle già esistenti. Il livello finisce quando il giocatore esaurisce le vite.

Requisiti funzionali

- Caricamento e gestione di differenti mappe, ognuna con uno specifico percorso per i nemici.
- Movimento e gestione dei nemici all'interno della mappa.
- Gestione e aggiornamento costante di una classifica che memorizzerà i migliori punteggi.
- Gestione delle risorse del giocatore, come ad esempio le monete, e dei parametri di vita del giocatore.
- Gestione degli upgrade delle difese, nello specifico permettere i miglioramenti solo se il giocatore rispetta alcune condizioni, come ad esempio avere sufficienti monete.

- Menù principale che all'avvio dell'applicazione permette di scegliere se giocare, consultare la classifica oppure chiudere l'applicazione.
- Gestione della difficoltà in modo incrementale proporzionale all'avanzamento nel livello e basata anche sulla difficoltà selezionata all'avvio del livello.
- Menù iniziale di gioco che permette al giocatore di scegliere difficoltà e livello.

Requisiti non funzionali

- Le interfacce grafiche dovranno essere il più user-friendly possibile.
- Esperienza di gioco più fluida possibile, prestando attenzione alla reattività dei comandi.
- Esperienza di gioco piacevole, prestando attenzione al bilanciamento del gioco, senza esagerare con la difficoltà.

1.2 Analisi e modello del dominio

Le entità fondamentali del gioco sono i nemici, ovvero i Bloon, e le torri. Il giocatore deve costruire e potenziare le torri per impedire il più possibile l'avanzata dei Bloon e conseguentemente rimanere in vita il più possibile, al fine anche di totalizzare un punteggio alto.

Il gioco è strutturato in livelli non connessi tra loro, la principale differenza tra un livello e un altro è la mappa. Ogni livello è a sua volta organizzato in ondate, il numero di ondate non è definito a priori ma ne verranno create fintanto che il giocatore rimarrà in vita.

Una volta scelti il livello e la difficoltà dal menù iniziale di gioco inizierà la prima ondata, ciascuna ondata è caratterizzata da numero di Bloon e potenza dei Bloon, per potenza si intende quanto sono resistenti e quindi difficili da fermare. Questi due parametri sono calcolati in prima battuta in modo proporzionale alla difficoltà selezionata e successivamente anche in base al progredire del livello. Per ogni Bloon che raggiunge la fine del percorso il giocatore perde una vita.

Le torri rivestono un ruolo fondamentale all'interno della dinamica di gioco, esse sono posizionabili solamente in certi punti specifici della mappa. Uccidendo i Bloon il giocatore guadagnerà monete con le quali potrà posizionare nuove torri oppure potenziare quelle già presenti sulla mappa.

Quando il giocatore ha esaurito le vite il livello termina e viene calcolato un punteggio, a questo punto il giocatore potrà scegliere di giocare nuovamente, consultare la classifica oppure chiudere l'applicazione.

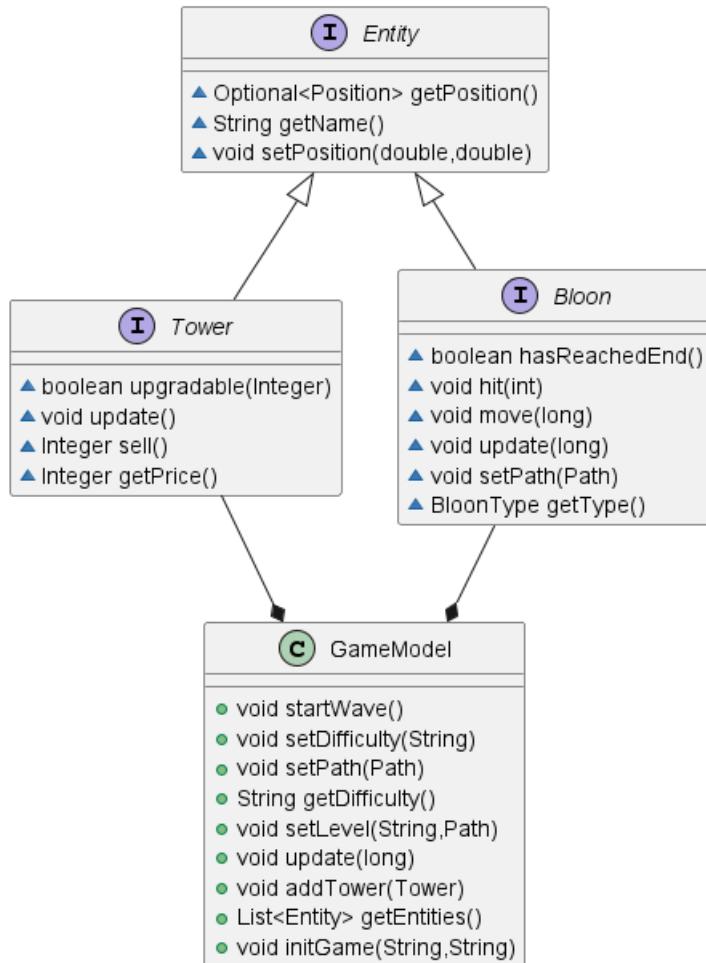


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti tra esse

Capitolo 2

Design

2.1 Architettura

L’architettura di questo software segue il pattern architettonicale Model-View-Controller (MVC). Per la parte grafica abbiamo scelto di utilizzare le librerie offerte da Java Swing, il giocatore interagisce con l’interfaccia grafica la quale si occupa di registrare le azioni eseguite e passarle al controller che non elaborerà direttamente i dati ma li passerà al model. Una volta che il model ha ricevuto i dati e ne ha registrato i cambiamenti si occuperà di dialogare con il controller per comunicare gli eventuali aggiornamenti e fare in modo che, tramite quest’ultimo, essi possano raggiungere la view che li mostrerà all’utente. Grazie a questa scelta un significativo cambiamento della view non porterà con sè cambiamenti anche sul model e sul controller.

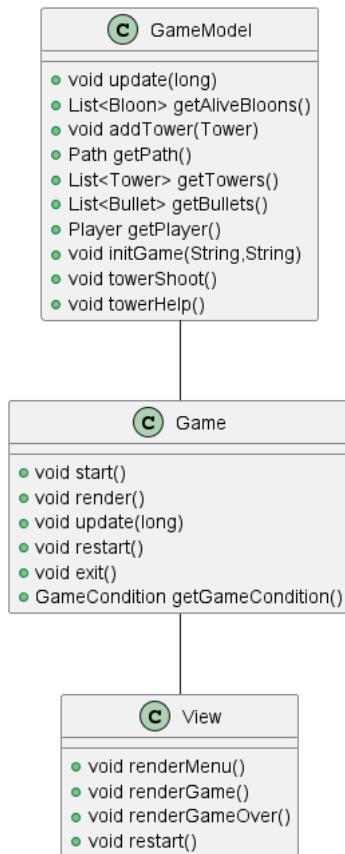


Figura 2.1: Schema UML architetturale del software. La classe GameModel è il model del sistema, la classe Game rappresenta il controller e la classe View rappresenta la view.

2.2 Design dettagliato

2.2.1 Alijon Batusha

Durante lo sviluppo del progetto, ho avuto il compito di gestire alcuni degli aspetti fondamentali del gameplay, tra cui il gameloop, la gestione della vita del giocatore, e la contabilità dei punti e delle monete.

Gameloop

Il gameloop è essenziale per garantire che il gioco proceda in modo coerente e ordinato. Ogni ciclo del loop esegue una serie di operazioni chiave: verifica degli input e aggiornamento del gioco. Questo assicura che ogni azione del giocatore venga registrata e che il gioco risponda di conseguenza, e che le entità del gioco (come nemici e torri) vengano aggiornate in modo appropriato.

Vita del Giocatore

La vita del giocatore è un indicatore cruciale della sua performance. Ho implementato un sistema in cui il giocatore inizia con un numero predefinito di vite. Ogni volta che un nemico raggiunge la fine del percorso, una vita viene sottratta. Se tutte le vite vengono perse, il gioco termina.

Punti e Monete

Il sistema di punteggio e monete è stato progettato per premiare il giocatore per le sue prestazioni. Ogni nemico sconfitto concede un certo numero di punti e monete. Le monete possono poi essere utilizzate per acquistare o potenziare le difese. Questo sistema incentiva i giocatori a sviluppare strategie efficaci e a migliorare le loro abilità nel corso del gioco.

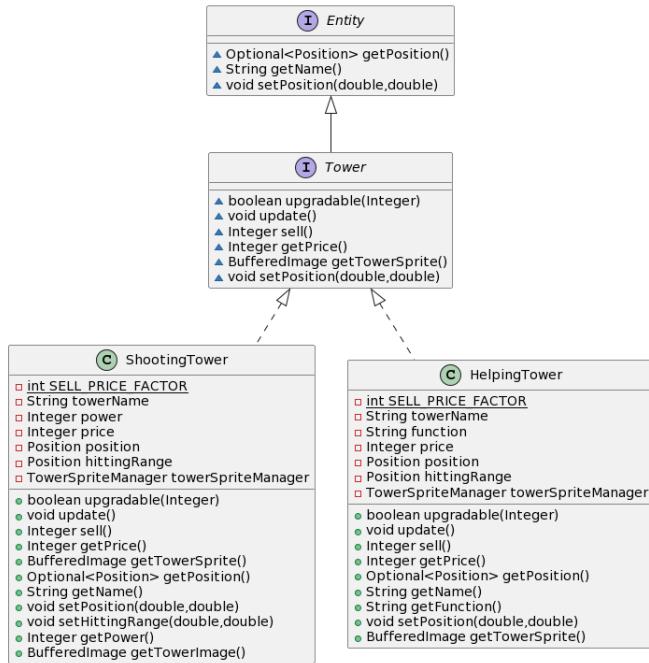
2.2.2 Stiven Gjinaj

Il miei compiti principali sono stati la creazione delle torri che sparano i bloon e quelle che aiutano le torri che sparano e la creazione dei menu, cioè il menu principale, il menù dove si sceglie la difficoltà, il menu che compare quando il giocatore perde e i menu in-game.

Creazione delle torri

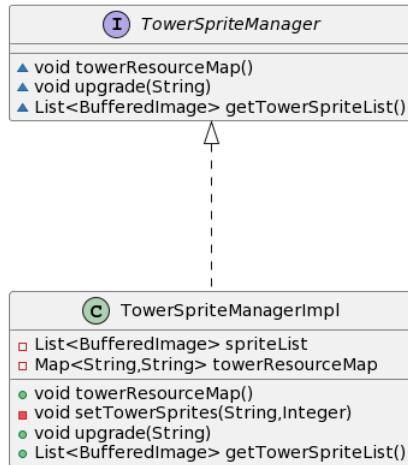
In generale l'interfaccia Tower contiene le proprietà principali delle torri. Usando questi metodi si può controllare se una torre può essere potenziata e poi c'è un altro metodo che gestisce i cambiamenti fatti alle torri quando vengono aggiornate. Ho deciso di non gestire i cambiamenti delle potenze e del raggio di sparo tramite metodi setter, ma di utilizzare fattori statici interni, ovvero ogni volta che una torre viene potenziata i poteri aumentano di 2. HelpingTower, a differenza di ShootingTower, hanno una proprietà function che viene usata per specificare il tipo di aiuto che danno. In ogni

caso c'è un fattore statico che si somma al potere o al raggio della torre che spara.



Gestione delle immagini

Le immagini non vengono gestite dalle implementazioni delle torri (ShootingTower e HelpingTower) ma da una classe a parte chiamate TowerSpriteManager. Questa classe mappa tutte le torri alle proprie path delle immagini poi nelle altre classi (es.: ShopMenu) si ha bisogno delle immagini per la prima versione di una torre basta chiamare il metodo apposito specificando il numero di upgrade dentro TowerSpriteManager.



Menù

I menu sono: il menu principale, il menu della difficoltà e di selezione della mappa, il menu del negozio e del potenziamento delle torri, il menu delle statistiche del giocatore. In generale ho cercato di organizzare la maggior parte dei tasti in funzioni riutilizzabili così da aumentare la leggibilità del codice in Java Swing e ogni tasto ha il proprio getter in modo da impostare l'ActionListener dalle altre classi.

2.2.3 Daniele Martignani

La mia parte prevede l'implementazione dei Bloon, la creazione del livello e delle ondate di gioco, la gestione dell'incremento di difficoltà del gioco.

Bloon

Il bloon è un entità di gioco, e il suo obiettivo è arrivare alla fine di un percorso prestabilito. Ogni bloon ha un valore numerico corrispondente alla vita, una velocità di movimento. Quando la vita raggiunge lo zero il nemico muore e il giocatore guadagna una certa quantità di monete, la quale verrà utilizzata dal giocatore per comprare e potenziare le torri. Per il movimento del nemico viene passato un oggetto Path che contiene le direzioni da seguire per raggiungere la fine del percorso. Esistono più tipi di nemici, i quali sono descritti nella classe `BloonType`. Ogni tipo di bloon ha velocità, vita e quantità di monete rilasciate alla morte diverse.

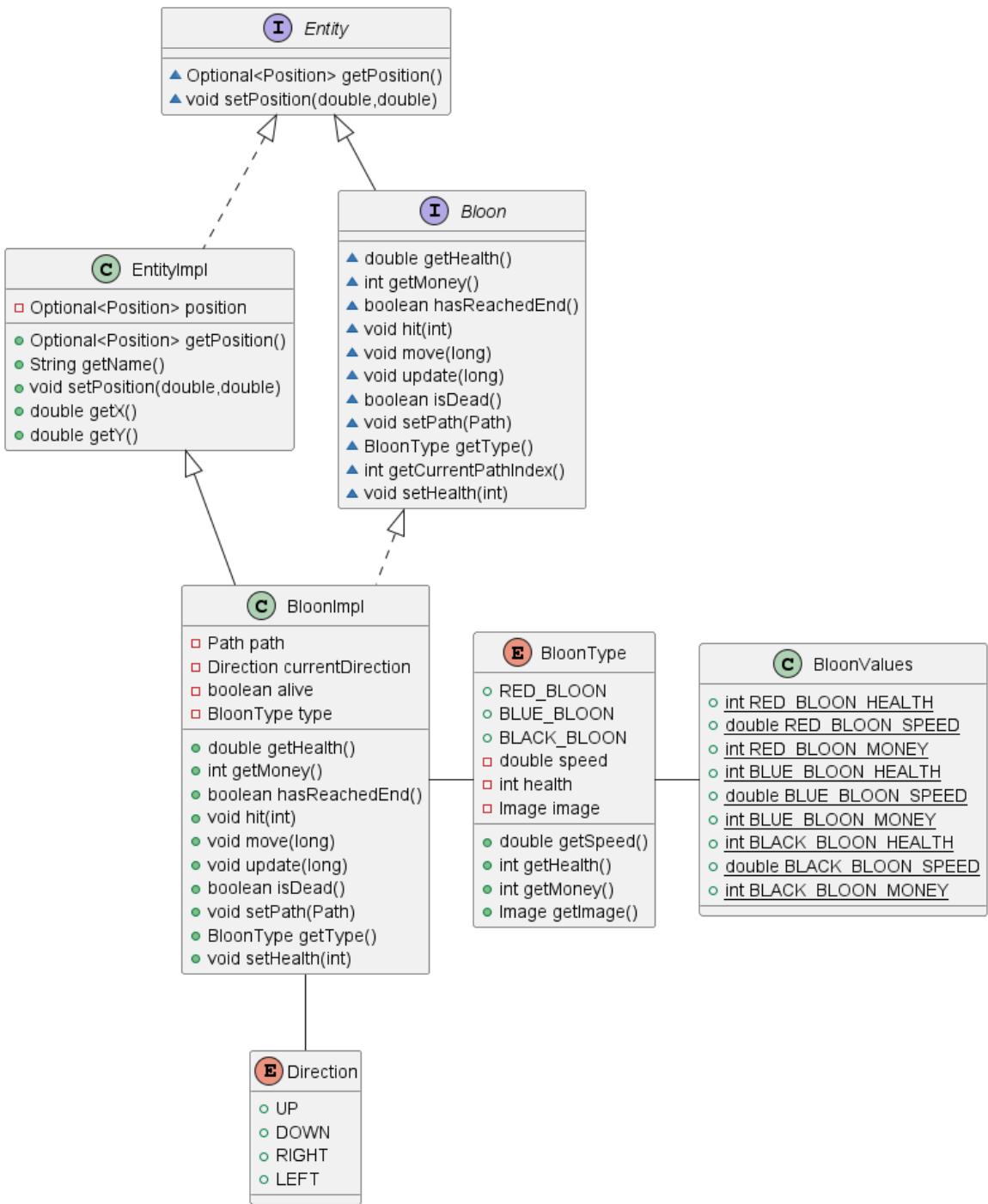


Figura 2.2: Schema UML della gestione dei Bloon

Ondate

Per la creazione di un livello vengono passati come parametri la difficoltà e la mappa scelte dal giocatore. La classe Level contiene un metodo il cui scopo è creare una nuova Wave. Ogni wave contiene un array di bloon che verranno poi spawnati nella mappa di gioco, la quantità di bloon per ogni ondata è casuale ma incrementata con l'avanzare delle ondate, inoltre la vita dei nemici aumenta a ogni ondata.

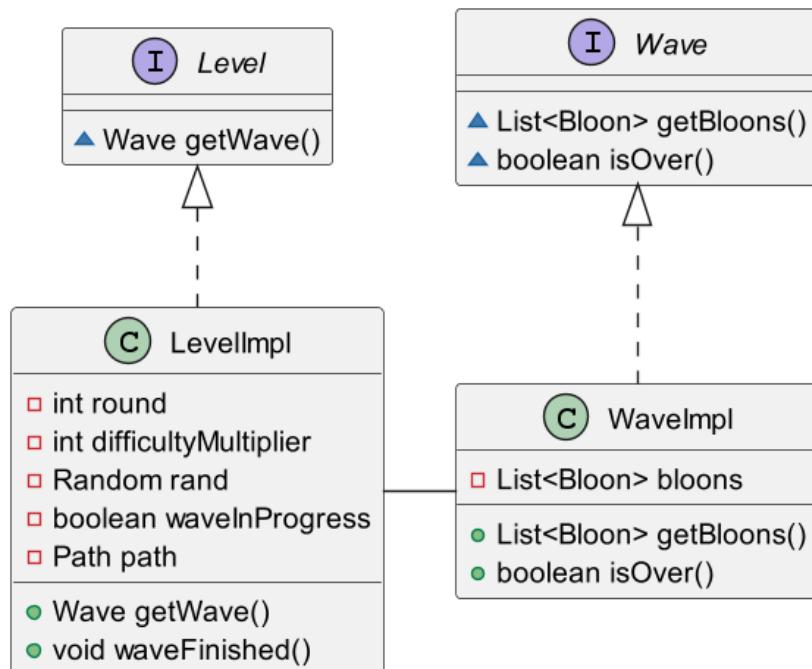


Figura 2.3: Schema UML delle ondate dei Bloon

Caricamento immagini Bloon

La classe Resources si occupa del caricamento delle immagini dei bloon ed è stata realizzata seguendo il pattern creazionale Singleton con inizializzazione ritardata e sicurezza per i thread. Questa scelta è stata fatta al fine di garantire l'esistenza di una singola istanza della classe per l'intera durata dell'applicazione. Tale istanza offre accesso alle texture dei vari oggetti di gioco, consentendo un uso efficiente e coordinato delle risorse. Per identificare le diverse immagini è stata creata una enumerazione (ItemType).

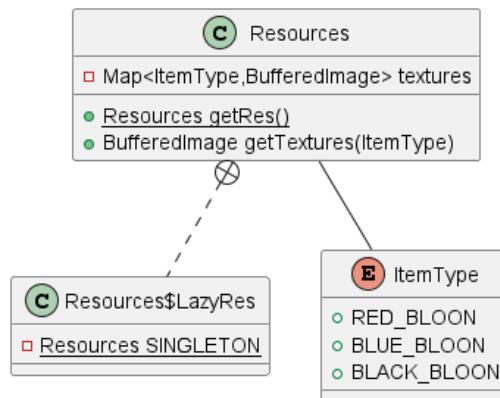


Figura 2.4: Schema UML del caricamento immagini

SoundManager

Nello stesso modo di Resources è stata creata la classe SoundManager. Per identificare i suoni è stata usata una enumerazione(SoundType)

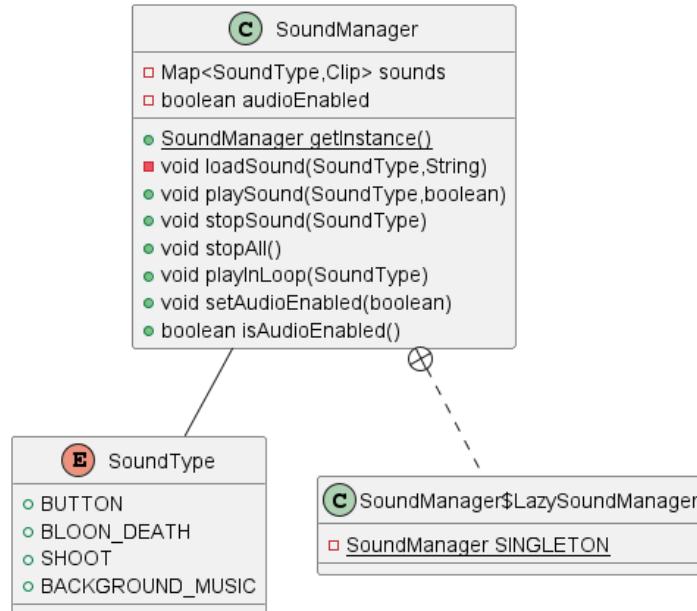


Figura 2.5: Schema UML del SoundManager

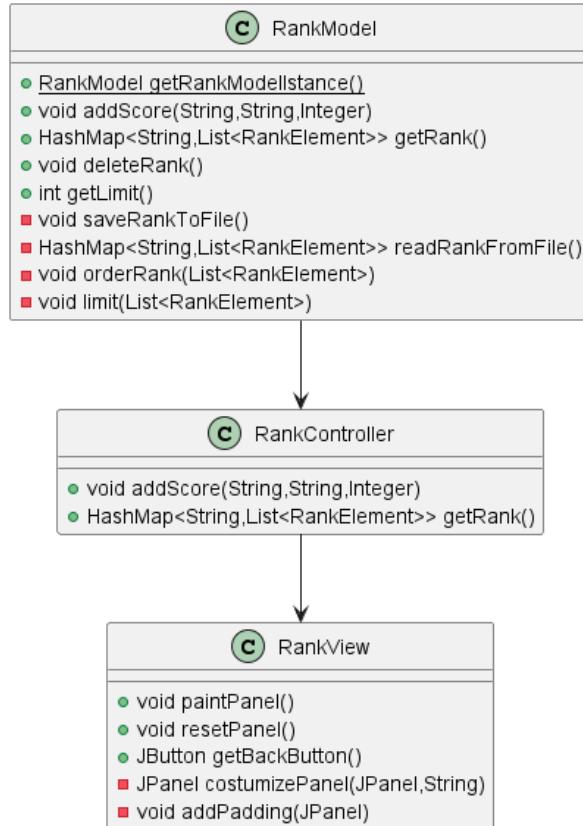
2.2.4 Penazzi Riccardo

Durante lo sviluppo del progetto mi sono principalmente dedicato alla creazione e alla prima gestione della mappa, all'implementazione e gestione della classifica dei punteggi ed al salvataggio dei punteggi.

Gestione della classifica

Per l'implementazione e la gestione della classifica ho strutturato il codice rispettando quanto più possibile il pattern MVC, creando quindi una classe dedicata al model, RankModel, una dedicata al controller, RankController e infine una dedicata alla view, RankView. Al primo avvio dell'applicazione, tramite il model, viene creato un file nella directory corrente che sarà utilizzato successivamente per salvare la classifica e leggerla in tutte le successive esecuzioni dell'applicazione. La classe controller mette a disposizione della view i metodi per interagire con la classifica, ovvero quelli per salvare un nuovo punteggio e leggere la classifica per mostrarla. In questo modo il model e il controller non dipendono direttamente dall view, se quindi in futuro si decidesse di optare per un'altra libreria grafica come JavaFX per un design più accattivante questa modifica sarebbe possibile senza impattare sulla parte logica.

Ho deciso di utilizzare il pattern Singleton nella classe RankModel poiché dopo un'attenta analisi ho pensato che fosse necessario rendere unica l'istanza di RankModel durante l'esecuzione del software per evitare il rischio di avere più model che tentano di scrivere sullo stesso file di salvataggio andando così a perdere importanti dati, o peggio, causando problemi durante l'esecuzione che obbligano all'arresto dell'applicazione.

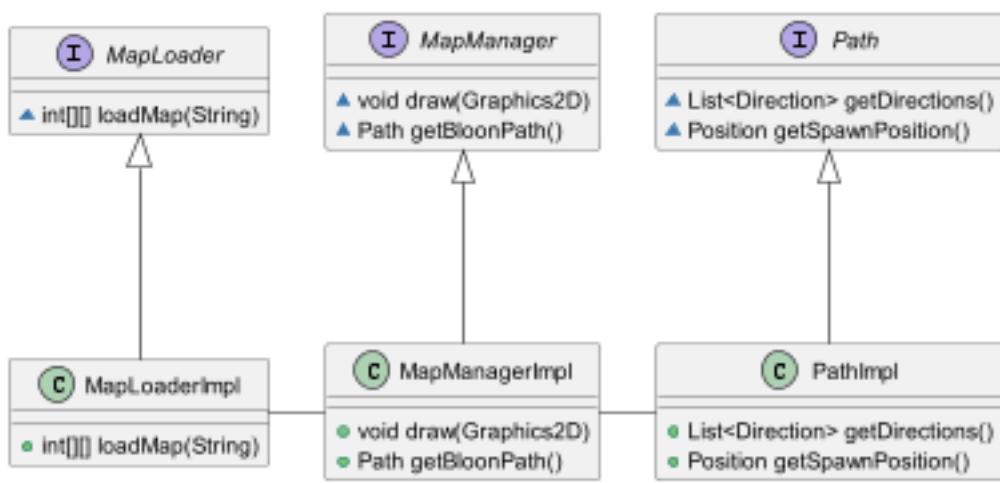


Creazione mappa

Per quanto riguarda invece la creazione della mappa ho cercato di mantenere questa fase quanto più estendibile in termini di aggiunta di livelli. Ho quindi creato un’interfaccia e una rispettiva classe che la implementa che si occupa di caricare la mappa da un file di configurazione, all’unico metodo presente è sufficiente passare il nome della mappa che si desidera caricare, in questo modo si potranno aggiungere quante mappe si vorranno in futuro.

Lo stesso discorso si può fare per il percorso dei nemici, che va di pari passo con la creazione e l’aggiunta di nuove mappe, anche in questo caso ho cercato di salvaguardare il più possibile l’estendibilità. L’interfaccia è Path e la classe che la implementa è PathImpl, anche in questo caso è quindi sufficiente passare il nome della mappa di cui si desidera caricare il percorso.

Entrambe queste due classi sono gestite da MapManagerImpl, la quale estende l’interfaccia MapManager.



Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per quanto concerne i test automatici abbiamo utilizzato la suite JUnit 5. Abbiamo deciso di sottoporre a test automatizzati le classi dedicate al caricamento di dati da file, come la mappa o il percorso dei bloon, ma anche le entità del gioco e in generale aspetti della logica del gioco, ed infine anche la classifica. I test realizzati sono quindi:

- **BloonImplTest:** il ruolo di questa classe è verificare la corretta creazione dei bloon, il loro movimento, e verificare se dopo essere stati colpiti con un colpo mortale risultino morti.
- **LevelImplTest:** il ruolo di questa classe è verificare il corretto funzionamento della creazione di un livello e la corretta creazione di più ondate(Wave) di gioco.
- **MapLoaderTest:** il ruolo di questa classe è verificare il corretto caricamento dei file che rappresentano la mappa, nello specifico la mappa letta viene confrontata con una mappa attesa uguale a quella scritta nei file.
- **PathTest:** il ruolo di questa classe è verificare il corretto caricamento dei file che rappresentano il percorso dei bloon, nello specifico il percorso letto viene confrontato con un percorso atteso che è uguale a quello scritto nel file.
- **RankModelTest:** il ruolo di questa classe è verificare il corretto funzionamento del modello della classifica, nello specifico vengono aggiunti punteggi in entrambe le classifiche assicurandosi che l'ordine sia corretto e viene testato il limite di cinque punteggi per classifica.

- **TowerTest:** il ruolo di questa classe è verificare la configurazione corretta delle torri, la potenza, il raggio di sparo/aiuto e se il manager delle immagini è stato configurato correttamente.

Inoltre durante tutto lo sviluppo del gioco sono stati anche eseguiti test manuali riguardanti il game loop e altri parametri come ad esempio il guadagno delle monete.

3.2 Metodologia di lavoro

Il primo aspetto su cui siamo concentrati è stato l'architettura che il nostro software avrebbe dovuto avere. Abbiamo ragionato su quali fossero le principali classi e interfacce da implementare il prima possibile in modo da non impedire l'avanzamento generale dello sviluppo e lavorare agevolmente anche in modo asincrono. Procedendo con lo sviluppo è stato sempre più necessario incontrarsi soprattutto quando abbiamo iniziato ad unire i pezzi. Per ciò che riguarda il DVCS abbiamo scelto di utilizzare Git e Github, creando un'organizzazione e successivamente un repository su quest'ultimo, iniziando lo sviluppo su un unico branch master finché ognuno di noi ha lavorato su parti separate del software, mentre nelle fasi mediane e finali si è reso necessario creare altri branch dove fare modifiche e unirli successivamente.

3.2.1 Alijon Batusha

Durante la fase di sviluppo, mi sono concentrato su tre componenti chiave del gioco:

- Gameloop
- Gestione del Giocatore (Player)
- Punti e Monete

3.2.2 Stiven Gjinaj

I miei compiti sono stati:

- Creazioni delle torri
- Gestione delle immagini delle torri
- Creazione dei menù

3.2.3 Daniele Martignani

I punti da me implementati sono stati:

- Creazione dei Bloon
- Creazione di Level e delle ondate di gioco(Wave)
- Visualizzazione dei nemici nella mappa
- Caricamento e gestione delle immagini dei bloon
- Gestione dei suoni di gioco

3.2.4 Penazzi Riccardo

Le attività da me svolte sono state:

- Caricamento e prima gestione della mappa (package *btd.model.map*).
- Caricamento del percorso dei nemici (package *btd.model.map*).
- Gestione della classifica (package *btd.model.score* e *btd.controller.score*).
- Visualizzazione della classifica (package *btd.view.score*).

Parti in cui abbiamo lavorato insieme:

- Classe *GameModel*: è la classe che funge da motore di gioco.
- Classe *View*: è la classe che si occupa di visualizzare correttamente le varie scene di gioco.
- Risoluzione di problemi legati a Spotbugs, CheckStyle e PMD specialmente nelle fasi di unione.

3.3 Note di sviluppo

3.3.1 Alijon Batusha

Riutilizzo del Codice

Ho adottato un approccio DRY (Don't Repeat Yourself) per garantire che il codice fosse efficiente e facilmente manutenibile. Questo ha comportato la creazione di funzioni modulari per gestire aspetti simili del gameplay.

Leggibilità e Funzionalità

Ho suddiviso le funzionalità in funzioni separate per migliorare la leggibilità e facilitare eventuali modifiche o aggiornamenti futuri.

3.3.2 Stiven Gjinaj

Dato che il mio compito è stato principalmente fare i menu, ho cercato di ripetere meno possibile il codice scrivendo funzioni riutilizzabili per fare i tasti o le label. Per quanto riguarda la parte funzionale, ho cercato di scrivere funzioni diverse per ciascuna funzionalità in modo da aumentare la leggibilità e la riusabilità.

3.3.3 Daniele Martignani

Ho utilizzato occasionalmente Lambda Expression ed Iterator in alcune parti di codice da me sviluppato. Ad esempio:

<https://github.com/BloonsTowerDefense/OOP22-btd/blob/1994de56fe31b28cf7882ab7ed5src/main/java/btd/model/GameModel.java#L141C9-L141C9>

<https://github.com/BloonsTowerDefense/OOP22-btd/blob/1994de56fe31b28cf7882ab7ed5src/main/java/btd/model/GameModel.java#L135C8-L135C8>

Fonti esterne utilizzate

- Suoni free copyright presi da: <https://mixkit.co/>

3.3.4 Penazzi Riccardo

Utilizzo di Lambda expression e stream

Nelle parti di codice da me sviluppato, quando possibile, ho cercato di utilizzare il più possibile Lambda expression, Stream ed Iterator. Ad esempio:

<https://github.com/BloonsTowerDefense/OOP22-btd/blob/1994de56fe31b28cf7882ab7ed5src/main/java/btd/model/score/RankModel.java#L139>

<https://github.com/BloonsTowerDefense/OOP22-btd/blob/1994de56fe31b28cf7882ab7ed5src/main/java/btd/model/map/MapLoaderImpl.java#L32>

<https://github.com/BloonsTowerDefense/OOP22-btd/blob/1994de56fe31b28cf7882ab7ed5src/main/java/btd/model/Map.java#L32>

src/main/java/btd/model/map/MapLoaderImpl.java#L42

Utilizzo libreria util.Serializable per memorizzare permanentemente i punteggi

Fonti esterne utilizzate

- Per risolvere due warning di spotbugs nella classe MapElementImpl ho trovato una guida su come eseguire la copia di un oggetto del tipo BufferedImage al seguente indirizzo <https://bytenota.com/java-cloning-a-bufferedimage#:~:text=We%20simply%20create%20a%20new,finally%20compare%20these%20two%20objects.&text=The%20output%20result%20we%20should,two%20objects%20are%20the%20different.>

Capitolo 4

Commenti finali

4.1 Alijon Batusha

Nel gestire il gameloop e le dinamiche del giocatore, ho compreso profondamente l'importanza di un codice ben strutturato. La collaborazione all'interno del gruppo ha sottolineato per me l'importanza del lavoro di squadra. Ogni discussione e feedback ricevuto ha affinato la mia visione e approccio rendendo il processo di sviluppo sia stimolante che educativo.

4.2 Stiven Gjinaj

Lo skill principale che ho acquisito con questo progetto è stato il lavoro in gruppo dato che ho sempre avuto problemi con il lavoro in gruppo. Per quanto riguarda gli aspetti tecnici, ho migliorato le conoscenze in Java Swing e in generale ho acquisito buone conoscenze della programmazione per i videogiochi. L'unica difficoltà che ho incontrato e che riguarda gli aspetti organizzativi con il gruppo, è stato svolgere i miei compiti in tempo così da permettere anche agli altri compagni del progetto di andare avanti.

4.3 Daniele Martignani

Sono soddisfatto del lavoro svolto perché, nonostante le difficoltà incontrate durante lo sviluppo, siamo riusciti a portare a termine il progetto. Tuttavia, ritengo che il codice da me scritto potrebbe essere migliorato e, se avessimo dedicato più energie alla fase di analisi del progetto e mantenuto un impegno costante durante tutto lo sviluppo, avremmo ottenuto risultati migliori e impiegato meno tempo nelle fasi successive.

4.4 Riccardo Penazzi

La mia opinione riguardo questo progetto è abbastanza contrastante, da una parte sono soddisfatto del lavoro svolto perchè comunque è stato portato a termine e il risultato mi piace, d'altra parte però non lo sono per nulla, riconosco di non aver sfruttato al meglio le potenzialità offerte dal linguaggio Java e dal DVCS che abbiamo utilizzato. Sicuramente il codice da me scritto può essere molto migliorato, un problema che riconosco è di aver un po' tralasciato inizialmente la parte di analisi e questo ha avuto pesanti conseguenze nelle fasi successive. Lavorare in gruppo non è stato facile, non essendo abituato a farlo e all'inizio ci ho messo un po' ad entrare in questa meccanica, ma poi ho capito come sfruttarla per avere confronti, chiedere opinioni ed analizzare il problema sotto altri punti di vista. Questo è stato per me il primo vero progetto e, nonostante quanto detto prima, mi ha sicuramente aiutato a comprendere meglio molti aspetti del linguaggio. Per quanto riguarda gli sviluppi futuri, mi piacerebbe ampliare la quantità di mappe disponibili e aggiornare la grafica per ottenere un aspetto più accattivante e visivamente più stimolante, magari passando a JavaFX.

Capitolo 5

Guida utente

All'avvio del software l'utente si troverà davanti al menù principale che presenta 3 pulsanti che permettono rispettivamente, partendo da sinistra:

- aprire menù di gioco
- visionare la classifica
- chiudere il gioco

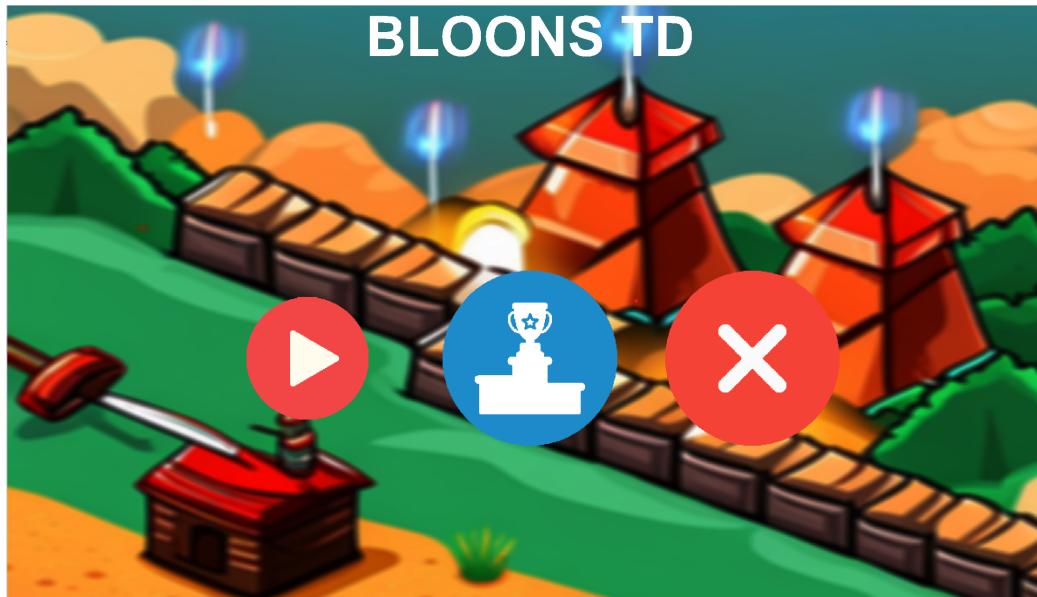


Figura 5.1: Immagine menù principale

Premendo quindi il pulsante per aprire il menù di gioco si apre un ulteriore menù dove è possibile selezionare la difficoltà di gioco e la mappa. Una volta

selezionati questi due parametri per avviare il gioco vero e proprio basterà premere il pulsante "play". Premendo la freccia in alto a sinistra si tornerà indietro al menù precedente.

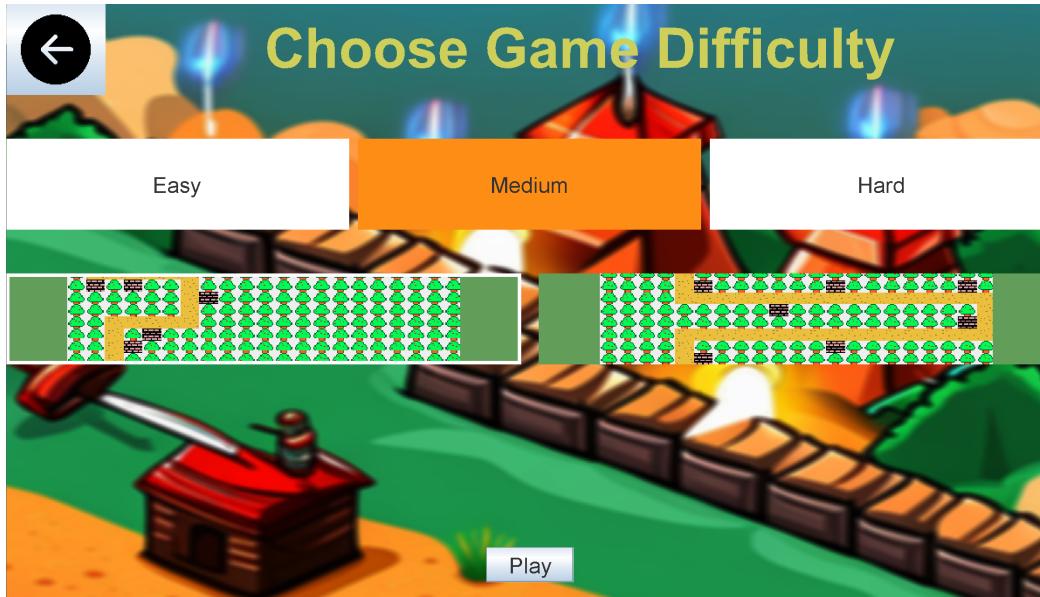


Figura 5.2: Immagine menù di gioco

Una volta avviato il gameplay per posizionare una torre è necessario selezionarla prima dal menù delle torri e posizionarla scegliendo una cella tra quelle disponibili. Ogni torre ha un costo e le monete del giocatore, così come la vita, sono visionabili in alto a destra.



Figura 5.3: Immagine gameplay

Una volta finite le vite comparirà la schermata game over, dove sarà chiesto di inserire il proprio nome per essere inseriti nella classifica, premendo il pulsante "save score" si verrà riportati al menù di gioco, dal quale sarà possibile avviare una nuova partita oppure tornare al menù principale per visionare la classifica.



Figura 5.4: Immagine game over