Relazione di "Escape from University"

Marco Antolini
(marco.antolini6@studio.unibo.it)

Emanuele Bertolero
(emanuele.bertolero@studio.unibo.it)

Daniel Capannini
(daniel.capannini@studio.unibo.it)

Denis Caushaj
(denis.caushaj@studio.unibo.it)

4 luglio 2022

Indice

Capitolo 1

Analisi

Il software mira alla realizzazione di un gioco platform 2D. Il gioco è ambientato nell'Università di Bologna, in particolare nel Campus di Cesena. Lo scopo del gioco è quello di emulare il percorso di studi di uno studente. Per riuscire a laurearsi, lo studente dovrà essere in grado di superare tutti gli esami del proprio corso. Nel caso in cui si fallisca uno o più esami, svolti dallo studente in modalità quiz, si avrà l'occasione di poterli superare sfidando il professore in una lotta in modalità "shoot 'em up".

1.1 Requisiti funzionali

- La mappa dovrà essere facilmente navigabile e in 6 stanze dovrà trovarsi un prof da sfidare. Quest'ultimo dovrà inizialmente sottporre il giocatore ad un esame ed eventualmente ad una sfida "shoot 'em up".
- Il gioco dovrà essere utilizzabile su tutti i principali sitemi operativi e senza problemi relativi alla dimensione dello schermo.
- Dovrà essere possibile vedere una lista di risultati delle giocate precedenti.
- Il personaggio principale deve essere movibile tramite la tastiera e la mappa scorrevole in modo che lui rimanga al centro.

1.2 Analisi e modello del dominio

Nel gioco ci sarà un solo ed unico personaggio principale che dovrà interagire all'interno della mappa di gioco con diversi elementi. Tra questi troviamo i professori, ovvero i "nemici", che potranno sfidare il giocatore con degli

esami a quiz oppure lanciandogli proiettili danneggiandolo. Per cercare di sopravvivere il giocatore potrà avvalersi di uno shop dove comprare diversi aiuti. Gli elementi costitutivi il problema sono sintetizzati in Figura 2.11.

La difficoltà primaria sarà quella di riuscire a far interagire correttamente il personaggio con i tanti elementi sopra descritti.

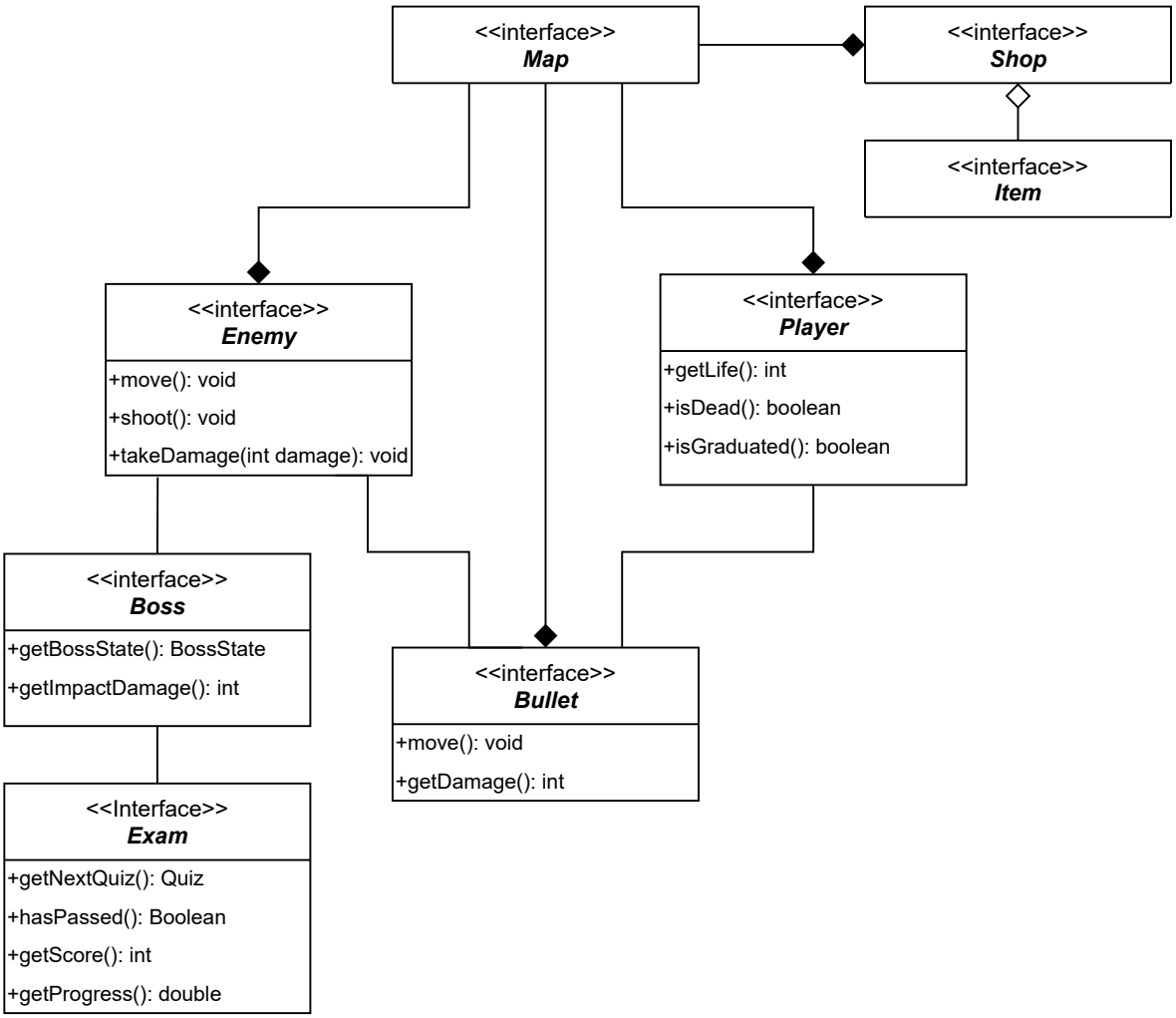
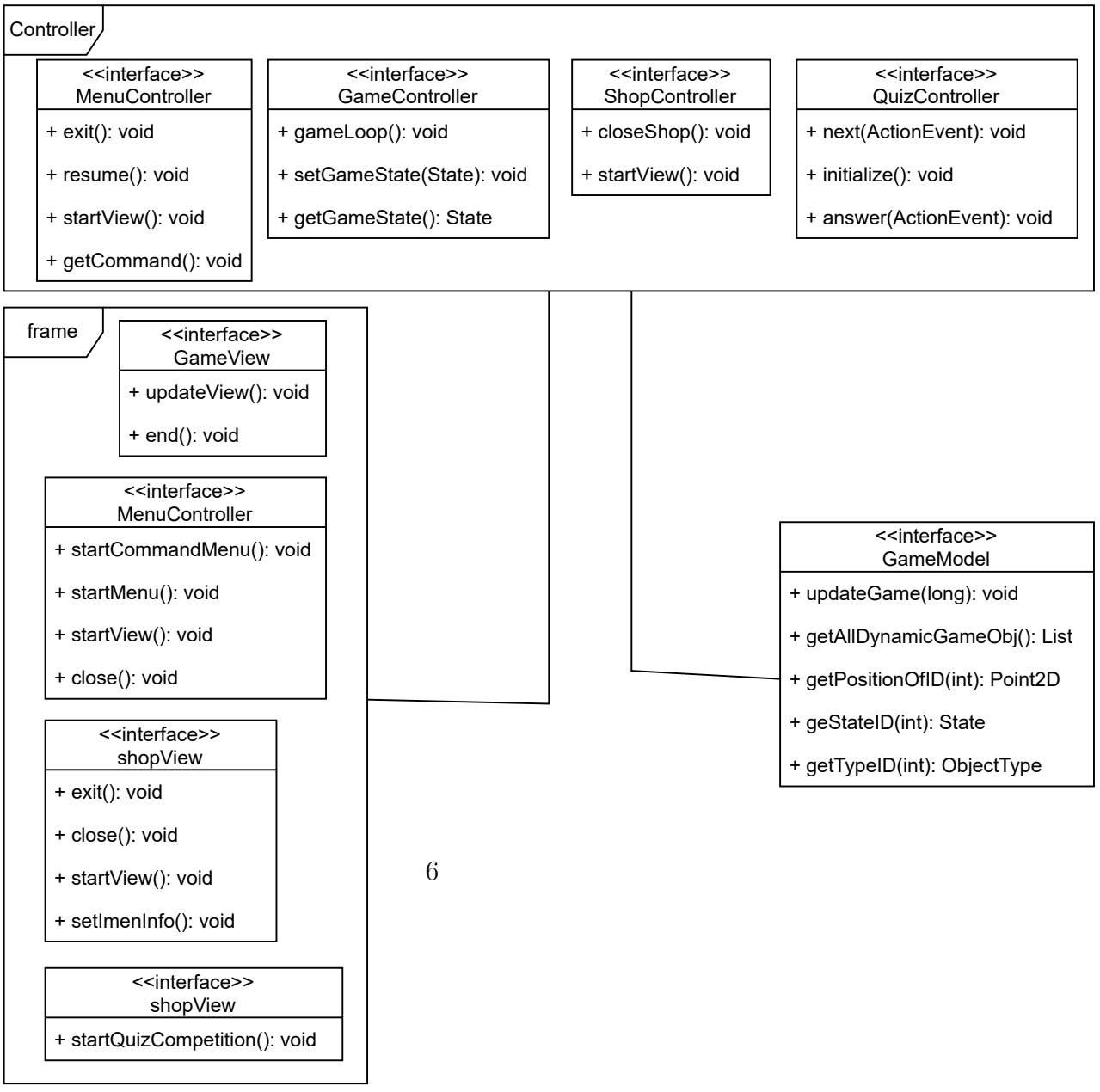


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro.

Capitolo 2

Design

2.1 Architettura



EscapeFromUniversity segue il pattern MVC. Quando dalla schermata del menu viene avviato il gioco si attiva il GameController il quale a sua volta costruisce un GameModel che poi al suo interno si occupa di costruire tutti i compomenti necessari alla logica del gioco, e costruisce una GameView che si occupa della parte grafica. Il GameView e il GameModel non comunicano mai direttamente fra loro ma sempre attraverso il GameController che è la componente cardine dell'applicazione. Il GameController ha il metodo gameLoop() il quale permette l'aggiornamento del gioco, e i metodi getState e setState permettono di modificare e monitorare lo stato di gioco e di conseguenza eseguire le azioni opportune sia da parte del GameController che delle varie View.

L'applicazione presenta più view, le quali hanno ognuna un proprio controller che le gestisce; tutti questi componenti vengono gestiti dal GameController Che si occupa lui della loro creazione e del loro avvio. Il possedere per ogni view un proprio controller rende possibile l'implementazione della view in maniera diverse con vari strumenti senza andare a modificare il controller e senza assolutamente modificare quella di model che è completamente indipendente dal resto dell'applicazione.

2.2 Design dettagliato

2.2.1 Marco Antolini

- Launcher Ho gestito la creazione del launcher del gioco tramite JavaFx. Dalla classe di lancio del gioco `App` viene chiamato il `LauncherView` che carica il file fxml che consiste nella view del launcher a cui è collegato il controller `LauncherController`. Dal menu del launcher è possibile accedere ad una nuova schermata che mostra i crediti (gestiti dal `CreditsController` associato al rispettivo file fxml) o una ulteriore che mostra una leaderboard di tutti i vincitori del gioco (gestita anch'essa da un controller `LeaderboardController` associato a un file fxml). La leaderboard legge i risultati da un file di testo che viene creato all'interno di una cartella creata anch'essa al lancio del gioco (se non ancora esistente) all'interno della user home per fare in modo che il file sia accessibile e modificabile una volta finito il gioco quando è il momento di salvare il punteggio. Lettura e scrittura su file txt sono gestite dalle due classi `ReadFile` e `WriteFile`.

- Shop

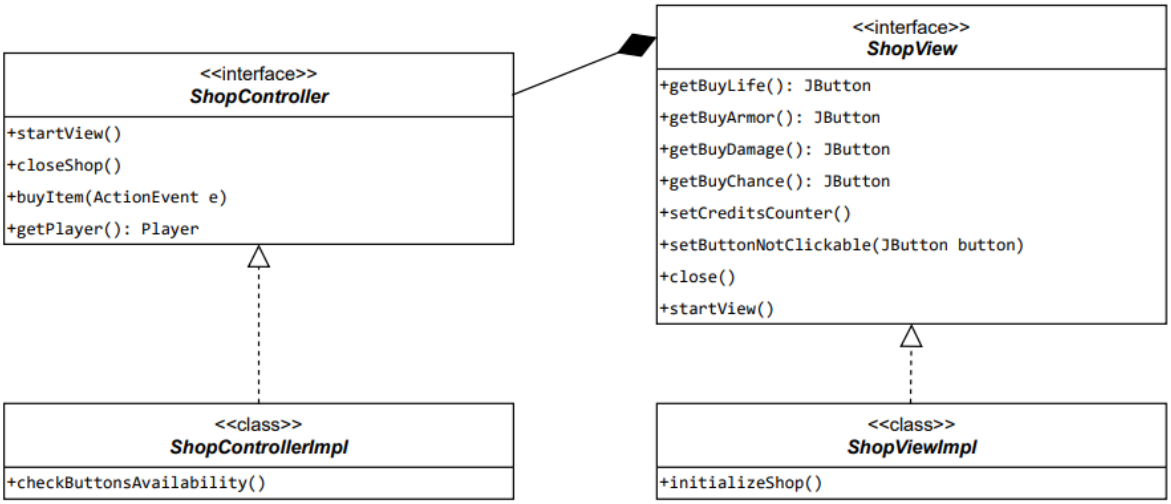


Figura 2.2: Schema UML di view e control dello shop.

Lo shop consiste in una schermata che il giocatore può aprire e con cui può interagire quando entra nella rispettiva stanza di gioco e si avvicina ai venditori. Tutto parte dallo `ShopController`, interfaccia che contiene alcuni metodi per la gestione dei bottoni (`closeShop()` e `buyItem()`) dello shop e un metodo `startView()` per avviare la view dello shop. Tale interfaccia è implementata da `ShopControllerImpl` che implementa i metodi precedentemente citati e ha al suo interno un ulteriore metodo `checkButtonsAvailability()` per controllare la possibilità di interagire con i bottoni e disattivarli nel caso i requisiti non siano soddisfatti. Quest'ultimo metodo chiama per l'appunto un metodo dell'interfaccia `ShopView` (che oltre a ciò, contiene alcuni metodi per la gestione dei vari componenti Java Swing presenti all'interno della view). Tale interfaccia è implementata da `ShopViewImpl` che contiene i metodi necessari per collegare la view al controller e per instanziare la schermata dello shop.

- Player

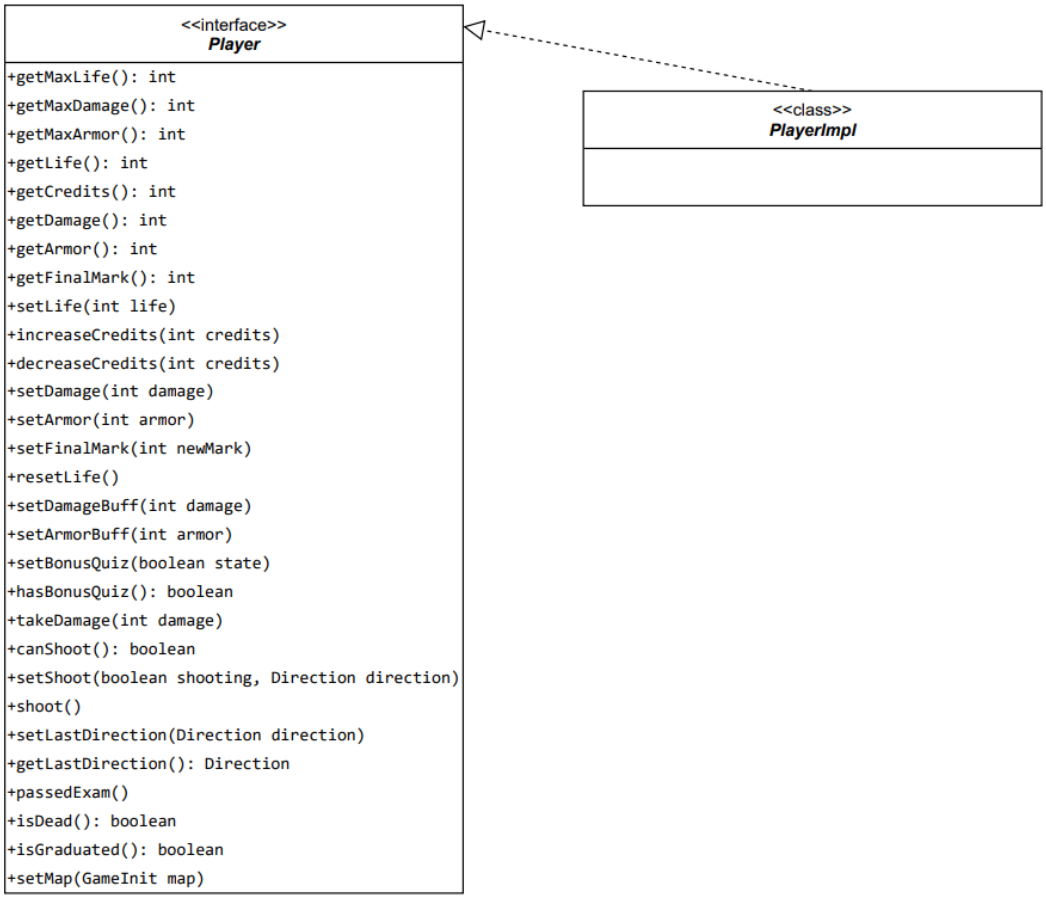


Figura 2.3: Schema UML del model del player.

Il `Player` è un'interfaccia che eredita da `DynamicGameObject` e implementa inoltre numerosi metodi per la gestione delle sue statistiche, gestione dello sparo in situazione di combattimento e gestione della sua posizione e direzione. Tale interfaccia è implementata da `PlayerImpl`.

- Sprite

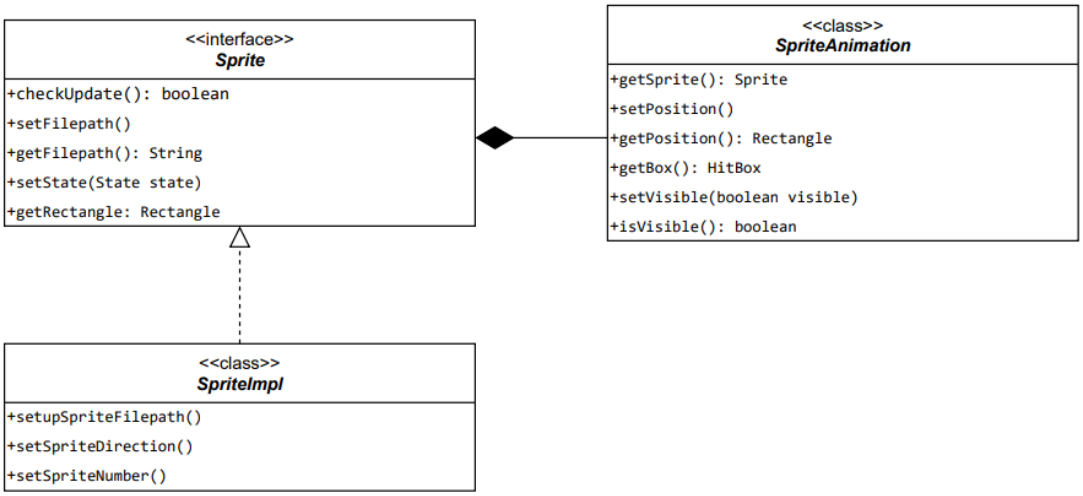


Figura 2.4: Schema UML del model degli sprite.

L'interfaccia `Sprite` e la relativa implementazione `SpriteImpl` insieme alla classe `SpriteAnimation` gestiscono gli sprite del player e dei boss e la loro animazione e contengono i metodi necessari per passare l'immagine e le informazioni collegate ad essa alla view.

2.2.2 Emanuele Bertolero

- **Model of Exam**

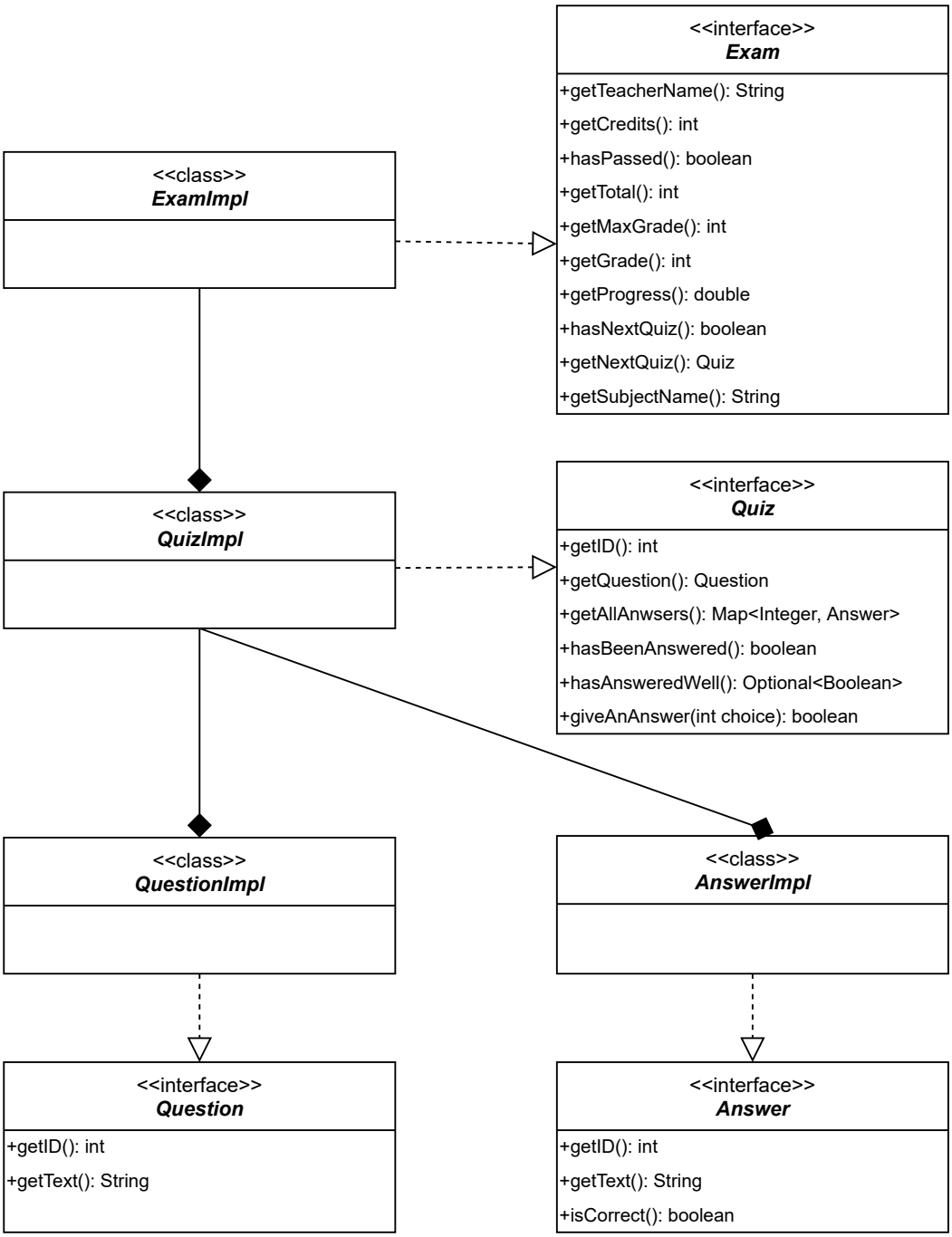


Figura 2.5: Schema UML modellazione in dettaglio la gestione degli esami.

L'UML presente in Figura 2.2 descrive come è stata modellata in dettaglio la gestione degli esami. La classe principale è la `ExamImpl`, implementazione dell'interfaccia `Exam`. Idealmente un esame è composto oltre che da informazioni di natura generale come il nome del docente, della materia e i relativi crediti anche da una serie di quiz, per questo la classe è stata pensata composta da altri oggetti fondamentali quali i `Quiz`. L'interfaccia `Quiz` viene implementata a sua volta dalla classe `QuizImpl`, anch'essa è composta da diverse altre classi senza le quali un quiz non avrebbe senso di esistere. Si tratta infatti delle classi `QuestionImpl` e `AnswerImpl`, implementazioni delle relative interfacce, che compongono il `Quiz`. Vista la complessità di un esame, composto da molte classi, nei punti successivi si vede come nel dettaglio è stata gestita la creazione di un esame con l'aiuto di pattern specifici.

- **Builder in Exam**

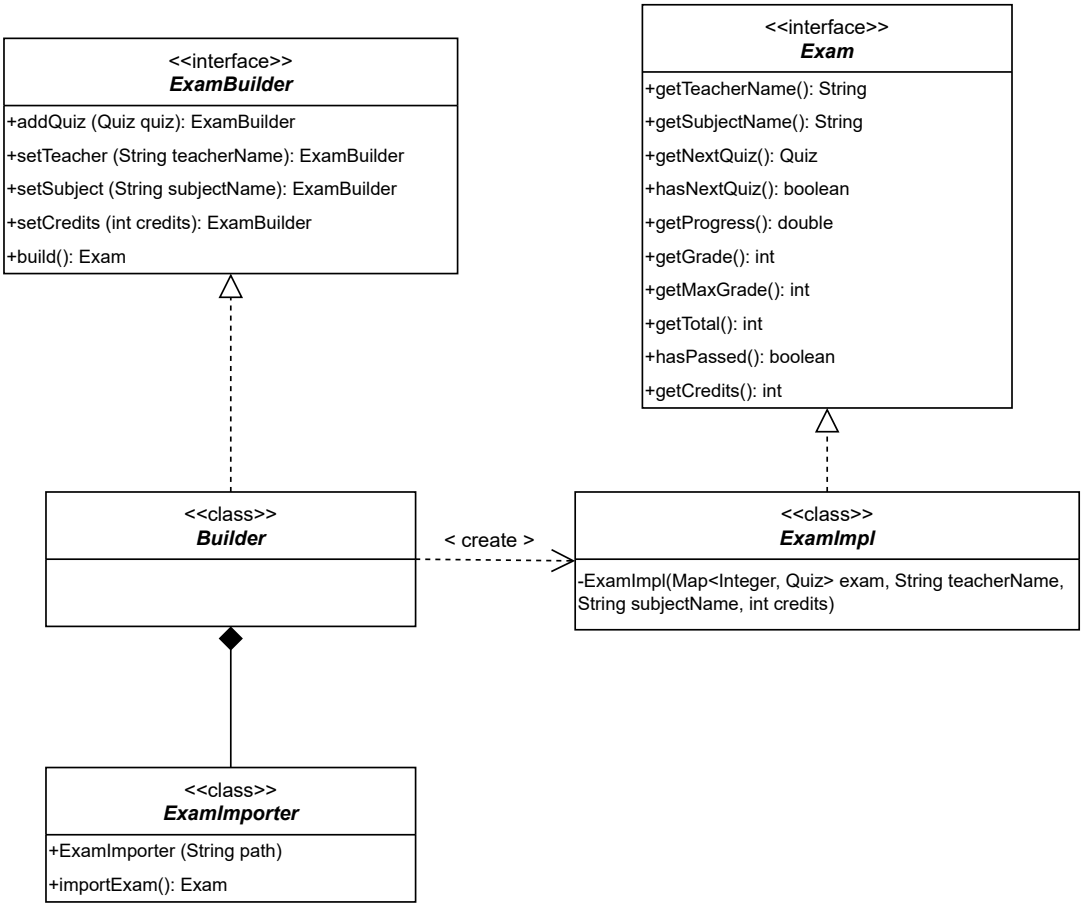


Figura 2.6: Schema UML del pattern Builder applicato agli Exam.

Come visto al punto precedente gli esami rappresentati dalla classe `Exam` sono composti da tante altre classi, è quindi necessario guidare l'utilizzatore alla costruzione di un esame. Per questo motivo è stato utilizzato il pattern builder, descritto dall'interfaccia `ExamBuilder` e poi implementato nella classe `Builder`, come indicato dall'UML in figura 2.3. Per fare ciò e stato impostato il costruttore di `ExamImpl` privato come previsto dal pattern in questione. L'utilizzatore per cotruire un esame dovrà quindi utilizzare i metodi `addQuiz()`, `setTeacher()`, `setSubject()` e `setCredits()`. Al momento della chiamata al metodo che cotruirà e ritornerà l'`Exam` vero e priprio, ovvero `build()`, verranno fatti dei controlli e creata la `Map<Integer, Quiz>` contente tutti i quiz aggiunti in fase di fabbricazione. In caso di errori e/o dimenticanze verranno lanciate le relative eccezioni indicando all'utilizzatore in modo dettagliato l'errore. Questo `Builder` è fondamentale all'interno della classe `ExamImporter` la quale, fornita la path del file con i quiz, tramite una chiamata al metodo `importExam()` importerà da suddetto JSON tutte le informazioni fondamentali per costruire un esame e tutti i quiz che contiene.

- **Builder in Quiz**

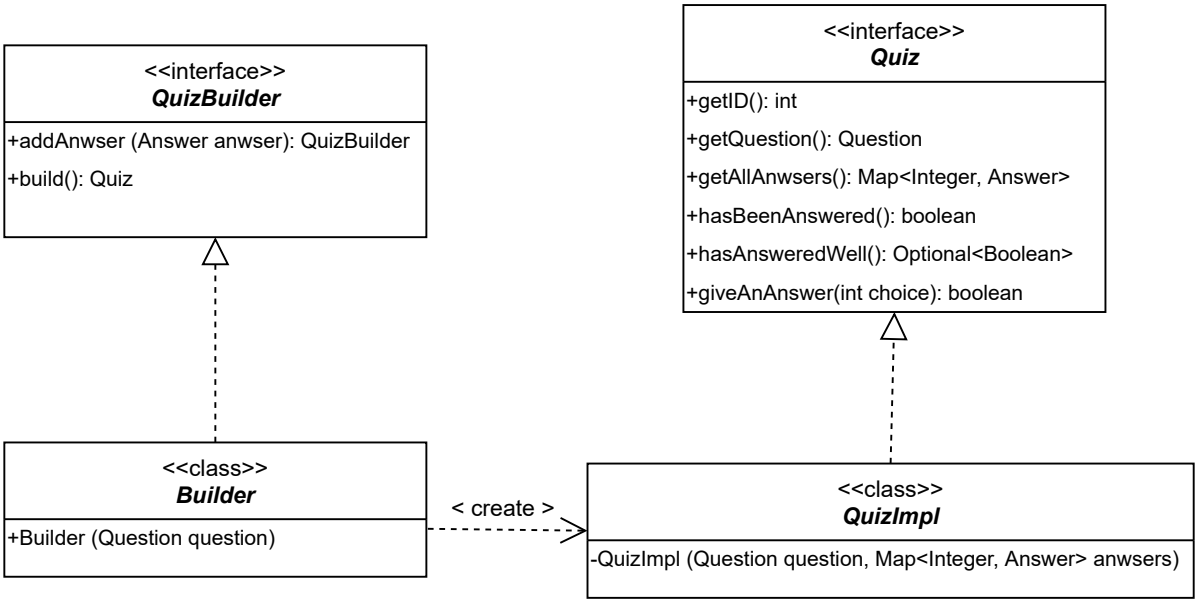


Figura 2.7: Schema UML del pattern Builder applicato ai Quiz.

I Quiz sono fondamentali all'interno di un esame, infatti lo compongono. Ogni quiz deve avere una sola domanda (`Question`) e ben quattro

risposte (`Answer`). Far rispettare questo vincolo sarebbe stato ben difficile senza l'utilizzo del Builder pattern. Il Builder permette infatti la creazione guidata attraverso dei metodi di un `Quiz`, questi metodi sono per l'appunto l'`addAnswer()` che permette l'aggiunta di una risposta per volta e del costruttore stesso che richiede obbligatoriamente una ed una sola domanda. Al momento della chiamata al metodo `build()`, che come previsto dal pattern deve creare il prodotto vero e proprio, vengono controllati i vincoli sopra indicati, se la verifica ha esito negativo vengono lanciate le relative eccezioni con l'indicazione dell'errore commesso, altrimenti se ha esito positivo viene creato il `QuizImpl`. L'oggetto creato `QuizImpl` è l'implementazione dell'interfaccia `Quiz` che modelliza per l'appunto un quiz. Questo oggetto ha costruttore privato come previsto dal pattern, così che l'utilizzatore non instanzi direttamente un `QuizImpl` ma utilizzi il suo `Builder` implementazione della relativa interfaccia `QuizBuilder`.

2.2.3 Daniel Capannini

- **Boss:**

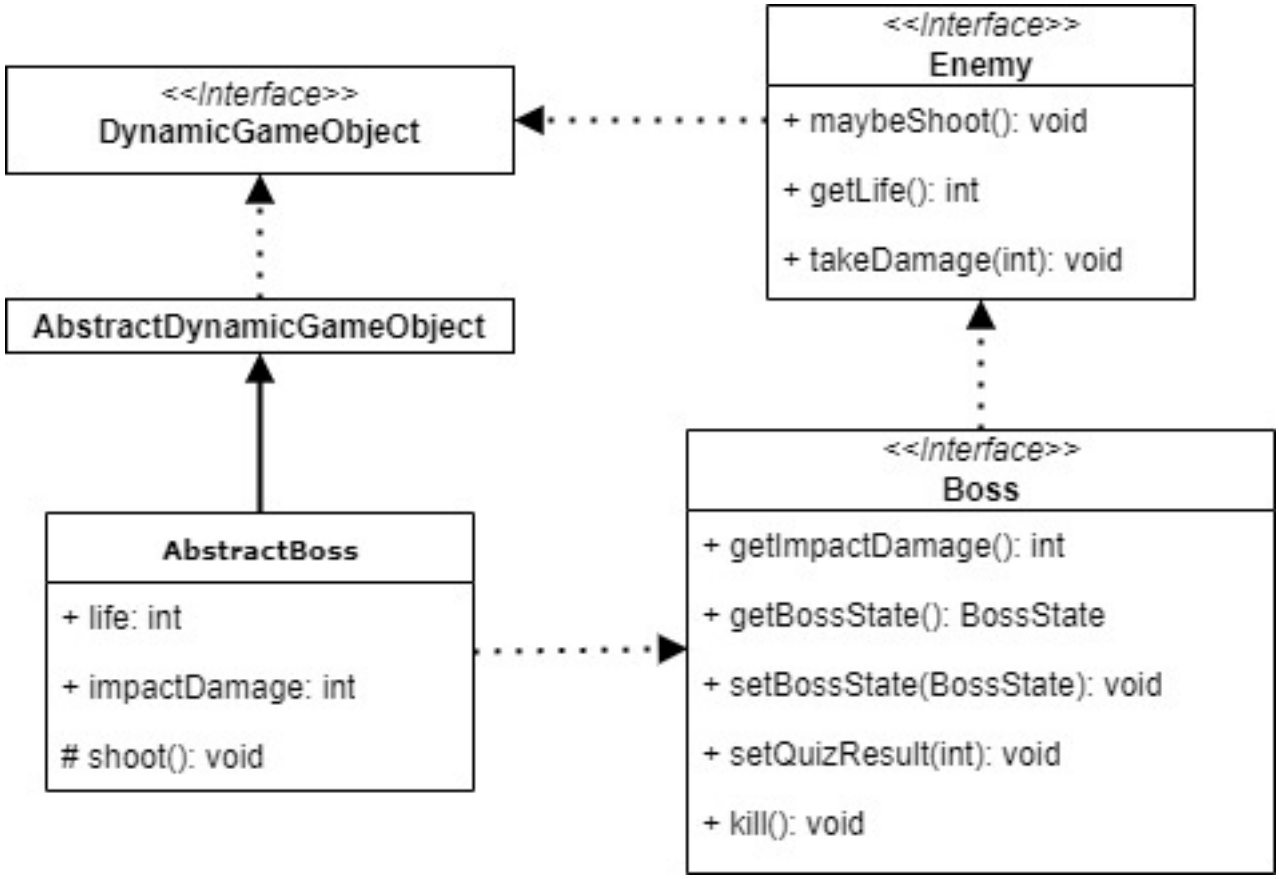


Figura 2.8: Schema UML della struttura di Boss.

I Boss presenti nel gioco sono una delle espressioni della classe DynamicGameObject poiché implementano la classe classe Enemy che estende l'interfaccia DynamicGameObject e AbstractBoss estende la classe AbstractDynamicGameObject. L'interfaccia Enemy non ha classe che la estende direttamente perché non sono presenti nel gioco personaggi che hanno quelle esatte caratteristiche ma può essere utile in caso di future aggiunte al gioco.

AbstractBoss gestisce tutta la fase di esistenza dell'oggetto, dalle collisioni con altri oggetti, la gestione della vita la propria morte, al metodo maybeShoot() al cui interno è presente il metodo astratto shoot() che verra implementato in maniera diversa per ogni boss, in questo è molto facile rendere diversi i boss uno dall'altro.

Grazie alla Enum presente all'interno dell'interfaccia Boss, che rappresenta lo stato di esso, viene gestita in modo diverso l'interazione con

gli oggetti e il suo comportamento.

- **Boss Factory:**

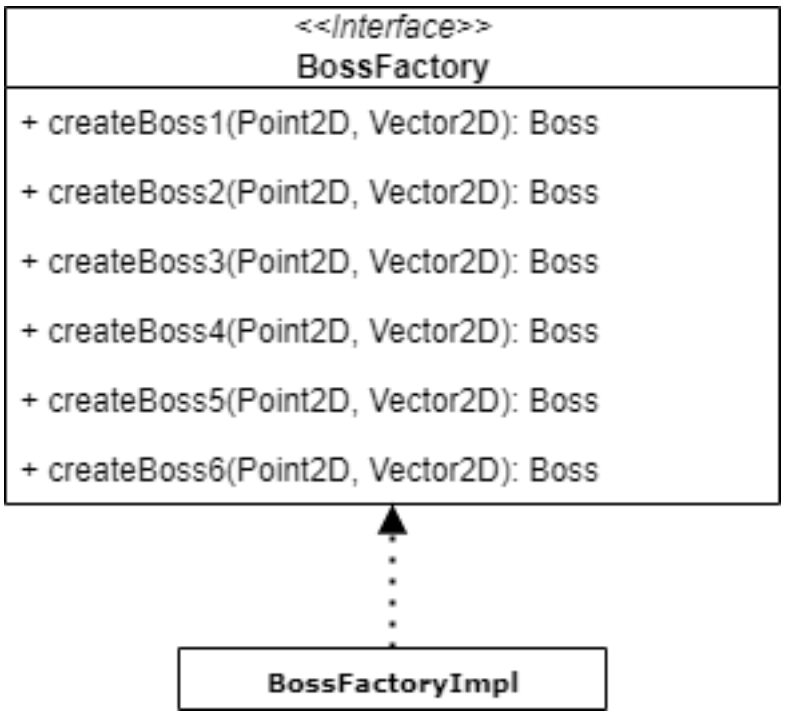


Figura 2.9: Schema UML della struttura della factory.

Questa factory di boss al suo interno ha un metodo per la creazione di ogni tipo di boss presente al momento nel gioco, che grazie a delle classi anonime che implementano il metodo astratto shoot() di AbstractBoss, creando cosi il boss che si vuole.

- **Bullet:**

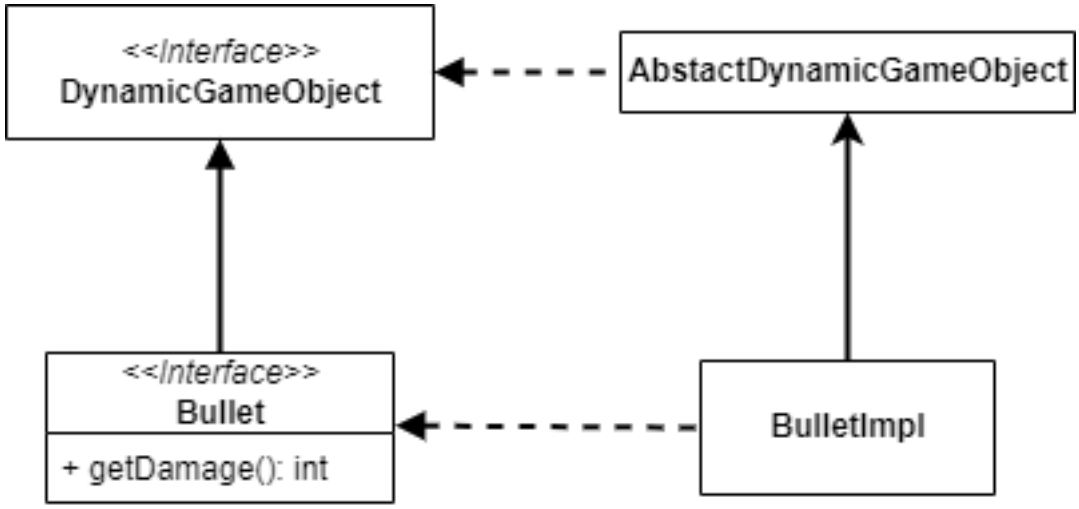


Figura 2.10: Schema UML della struttura dei Bullet.

I Bullet sono una parte della strutture del gioco. L'interfaccia Bullet estende l'interfaccia DynamicGameObject e perciò possiede tutte le sue caratteristiche e in aggiunta ha il metodo GetDamage il quale permette di sapere quanti danni infligge al bersaglio, la classe BulletImpl estende AbstractDynamicGameObject e implementa l'interfaccia Bullet.

- **Bullet factory:**

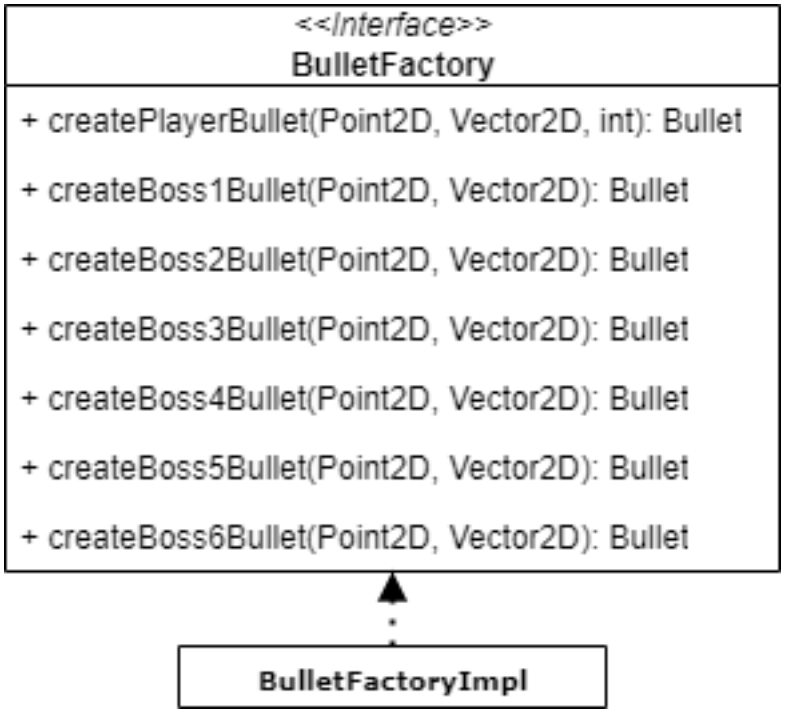


Figura 2.11: Schema UML della struttura della factory.

Questa factory permette di costruire qualsiasi tipo di Bullet che viene usato nel gioco, tramite l'interfaccia e la sua implementazione di cui ogni metodo restituisce un oggetto di tipo Bullet con le specifiche relative ad ogni personaggio.

2.2.4 Denis Caushaj

- **Creazione della mappa**
  La parte di progetto da me sviluppata si concentra maggiormente sulla creazione e sulla gestione della mappa di gioco, inizialmente disegnandola, per poi modellarla, stamparla a video e infine controllarla in modo corretto.
  La mappa creata è una "Tile Map" utilizzando quindi il concetto di Tile. Dopo averla esportata come file ".tmx" (Tiled Map XML) ho realizzato la classe `TMXMapParser` che permette di salvare le informazioni dal file utili a poter modellare la mappa in modo corretto, generando un oggetto di tipo `MapProperties`, contentente una lista di `Tileset` e una di `Layer`. Quest'ultima è composta a sua volta da altre informazioni, quali una lista di `Tile`.

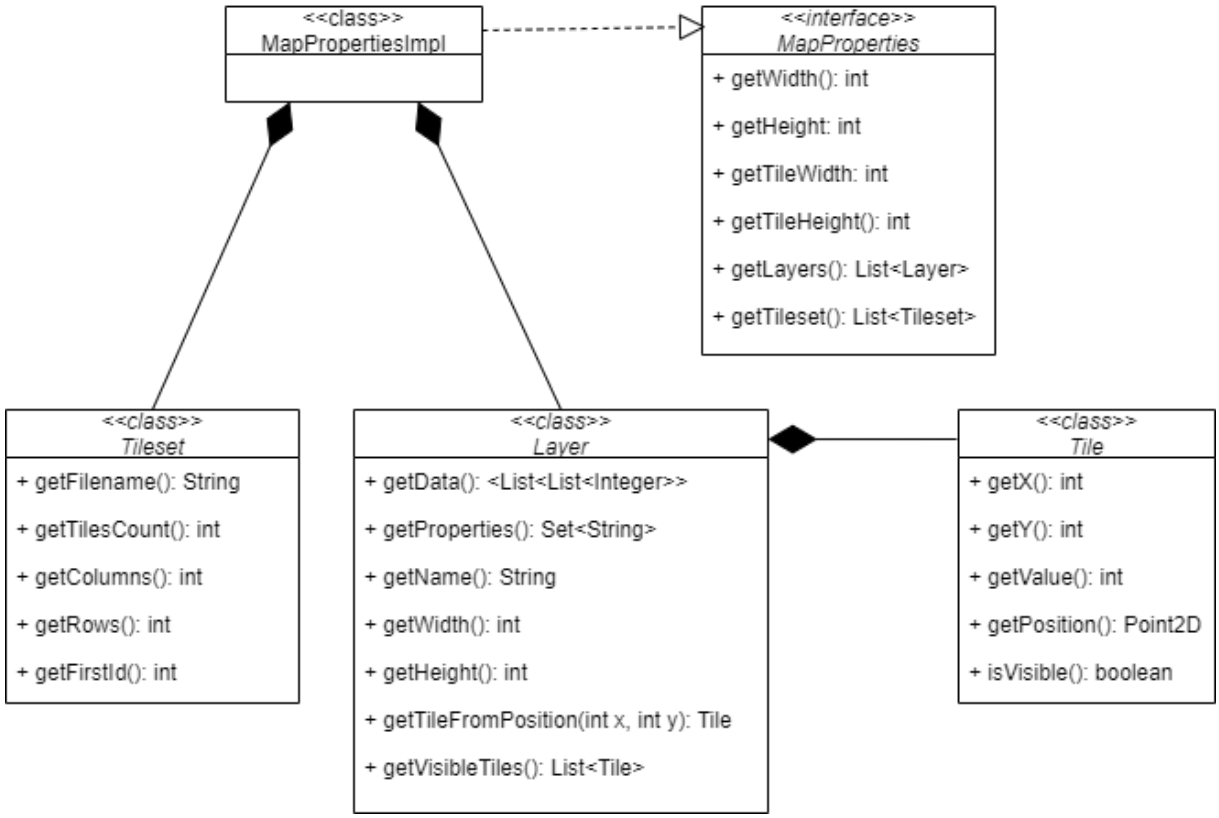


Figura 2.12: Schema UML che descrive i collegamenti tra gli elementi principali di una Tile Map

- **Camera**
  Essendo la mappa un'immagine unica, è necessario astrarre il concetto di camera per visualizzare una porzione sola della mappa. L'interfaccia `Camera` infatti ha il compito di calcolare la porzione di mappa da proiettare sullo schermo. Poiché l'obiettivo è quello di creare una camera mobile in grado di seguire il giocatore è stata creata la classe `PlayerCameraImpl` che implementa l'interfaccia Camera. L'implementazione di questa classe permette d'altronde di calcolare la porzione di mappa da proiettare a prescindere dalle proporzioni dello schermo o della finestra, centrando sempre alla perfezione il giocatore.

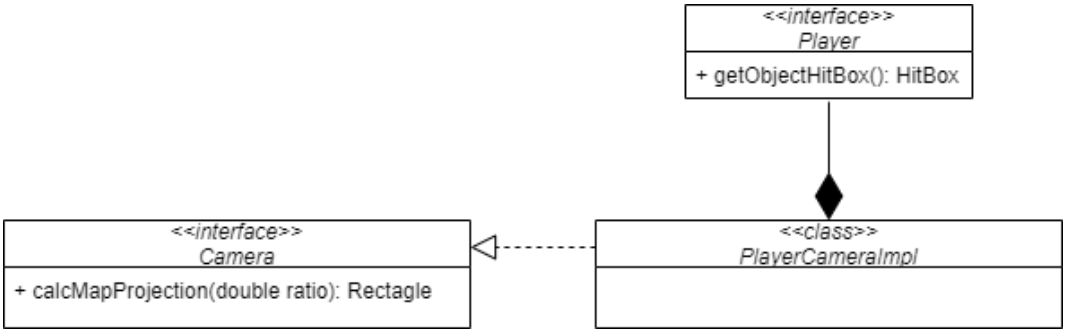


Figura 2.13: Schema UML che descrive l'implementazione della camera

- **Proiezione sullo schermo**
  La procedura per proiettare la mappa sullo schermo è composta da tre passi fondamentali, ovvero dalla ricerca del tile giusto da disegnare, disegnarlo al momento giusto e infine proiettarlo.
  La ricerca del tile viene fatta grazie alla classe `TileSearcherImpl`, che in primis cerca il tile da disegnare tra i vari tileset e ne calcola la posizione. Dopo aver definito l'insieme dei tile da disegnare la classe `LayersControllerImpl` si occupa di controllare quali di questi sono visibili e quali no, in modo tale da proiettare solo i tile che è logico vedere in base a dove si trova il giocatore. Infine le classi `CanvasDrawerImpl` e `TileDrawerImpl` si occupano della proiezione dello schermo.

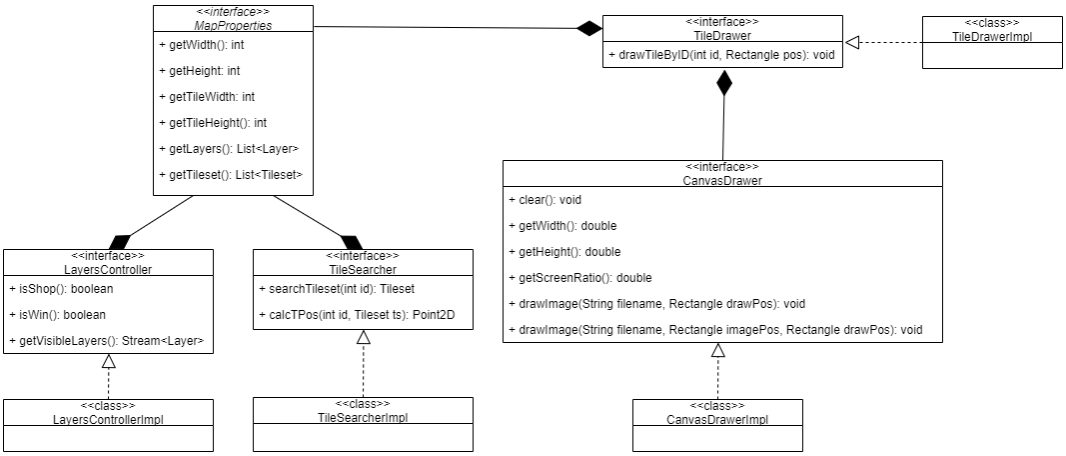


Figura 2.14: Schema UML che descrive il processo di proiezione sullo schermo

- **Ostacoli**
  Essendo gli ostacoli a tutti gli effetti delle immagini, per poter determinare in modo semi-automatico la loro natura vengono usate le classi

`Obstacle` e `ObstacleImpl`.
Grazie a queste classi si può ottenere una lista di ostacoli (`ObstacleObject`), dividendole per tipo, in modo tale da poter avere un effetto specifico quando il giocatore collide.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In Escape From University il testing automatizzato è stato utile per testare principalmente le classi che necessitavano di un riscontro immediato per poter procedere con l'implementazione di altro codice.

3.1.1 Emanuele Bertolero

- **TestExam**: in questa classe di test vengono effettuati alcuni test sulla creazione di `Question`, `Answer`, `Quiz` e `Exam` utilizzando i relativi Builder. Nei vari test vengono volutamente non rispettiti dei vincoli necessari e verificate che le eccezioni generate siano quelle attese.
- **HitBoxTest**: in questa classe di test vengono effettuati alcuni test sulla creazione delle `HitBox` e alcuni casi di collisioni tra HitBox oltre a verificare che i valori ritornati dai metodi della classe sia quelli attesi.

3.1.2 Daniel Capannini

- **TestBullet**: in questa classe di test vengono effettuati alcuni test sulla creazione di Bullet attraverso la BulletFactory.
- **TestBoss**: in questa classe di test vengono effettuati alcuni test sulla creazione dei Boss attraverso la BossFactory.

3.1.3 Denis Caushaj

- **TestTileset**: in questa classe di test vengono effettuati alcuni test sul corretto funzionamento di `TMXMapParser`, controllando che i tile-

set siano stati parsati correttamente; vengono inoltre effettuati i test sul corretto funzionamento di `TileSearcher`, controllando che i tile vengano cercati nel tileset giusto.

3.2 Metodologia di lavoro

3.2.1 Marco Antolini

Ho sviluppato in autonomia:

- Gestione del launcher: **launcher.***
- Tutti i file fxml relativi al launcher: **resources/layouts/Launcher.fxml**, **resources/layouts/Credits.fxml** e **resources/layouts/Leaderboard.fxml**
- Model del player: **model.player.***
- Controller e view dello shop: **shop.***
- Gestione degli sprite: **sprites.***
- Delle enum utili per la gestione di stati di gioco, movimento e direzione: **model.GameState**, **model.gameObject.State** e **model.gameObject.Direction**
- Delle classi di utility: **utilities.LauncherResizer** e **utilities.OSFixes**

3.2.2 Emanuele Bertolero

Ho sviluppato in autonomia:

- Gestione dei quiz: **model.quiz.***, **quiz.***,
- Pagina di end: **model.inGame.end.***
- HitBox: **model.basic.HitBox**, **model.basic.HitBoxImpl**
- Tutti i file fxml relativi ai quiz e alla end: **resources/layouts/Quiz.fxml** e **resources/layouts/End.fxml**

Ho collaborato alla creazione di:

- Controller generale del gioco: **inGame.GameController** e **inGame.GameControllerImpl**
- GameObject classi base degli oggetti del gioco: **model.gameObject.***
- Parte del player relativa all'interazione con i quiz: **model.gameObject.player.***

3.2.3 Daniel Capannini

ho sviluppato in autonomia:

- model.gameObject.bullet*
- model.gameObject.boss.*
- model.Model/ModelImpl
- menu.*
- ho collaborato alla creazione di model.gameObject.* inGame.*

3.2.4 Denis Caushaj

Porzioni di progetto sviluppate in autonomia

- Creazione della classe che gestisce la conversione della mappa da file a oggetti: **model.map.TMXMapParser**;
- Implementazione della mappa:

  -Model: **model.map.*** (fatta eccezione per MapManager* svolta in collaborazione);

  -Control: **controller.map.***

  -View: **view.map.***
- Implementazione del negozio:

  -Model: **model.map.shop.***
- Implementazione di classi di utility:

  -Model: **model.basics.Rectangle**
- Disegno della mappa con l'utilizzo dell'applicazione Tiled

Porzioni di progetto sviluppate in collaborazione con i colleghi

- **model.GameModel***
- **model.basics.HitBox**

- **model.basics.Point2D**
- **model.gameObject.GameObjectType**
- **model.map.MapManagerImpl**
- **launcher.Launcher***
- **inGame.GameController***
- **inGame.GameViewImpl**
- **inGame.ShopControllerImpl**

Codice riadattato

- **view.map.canvas.ResizableCanvas** da StackOverflow;
- **model.basics.Hitbox** metodo isColliding da `https://www.baeldung.com/java-check-if-two-rectangles-overlap#:~:text=Java%20Implementation&text=Our%20isOverlapping()%20method%20in,we%20compare%20their%20y%2Dcoordinates`

3.3 Note di sviluppo

3.3.1 Marco Antolini

- Uso di `JavaFX` per la parte grafica;
- Uso di `lambda expressions`.

3.3.2 Emanuele Bertolero

- Uso di `JavaFX` per la parte grafica;
- Uso di `lambda expressions`;
- Uso di `Stream`;
- Uso dei `pattern`.

3.3.3 Daniel Capannini

- Uso di `lambda expressions`;
- Uso di `Stream`;
- Uso di `Java.swing` e `JavaFX`.

3.3.4 Denis Caushaj

- Uso di `lambda expressions` per rendere il codice più semplice e comprensibile;
- Uso di `Stream` per la visitazione delle strutture dati;
- Uso di `JavaFX` per lo sviluppo della parte grafica.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Marco Antolini

Questo progetto è stata la mia prima esperienza di lavoro di gruppo che ha richiesto un notevole carico di impegno e di collaborazione. Essendo tutti alle prime armi, non sono mancate le difficoltà soprattutto organizzative. La suddivisione iniziale dei compiti di ciascuno poteva essere svolta meglio ma poi abbiamo cercato di rimediare in corso d'opera. É mancata anche abbastanza una coordinazione durante lo svolgimento della propria parte.. ognuno di noi aveva un'idea su come sarebbero state gestite certe cose che in certi casi era differente dall'idea di altri poichè non avevamo dato sufficiente attenzione ad una vera e propria modellazione generale inizialmente. A causa anche di difficoltà nell'incontrarci ci siamo poi trovati a dover risolvere alcuni problemi in breve tempo. Personalmente ritengo che questo progetto mi abbia aiutato e insegnato molto; mi ha fatto capire che per un lavoro di gruppo è necessara davvero tanta coordinazione, mi ha permesso di imparare meglio l'uso di gui tramite java swing e javafx, l'uso di generici e stream (nonostante poi per le ragioni citate prima, per fare in modo che le parti di ciascuno di noi combaciassero, io li abbia eliminati dalla mia parte), l'uso di lamba expressions e in generale le mie capacità di programmazione.

4.1.2 Emanuele Bertolero

Questo progetto è stato per me una prima esperienza di lavoro in gruppo, come lo è stata anche per i miei compagni. In questa autovalutazione ci terrei a sottolineare che, nonostante le tante difficoltà, sono abbastanza soddisfatto del lavoro finito. Poiché e la prima volta che lavoro in gruppo ho da subito

riscontrato le normali prime difficioltà, come quella del coordinarsi, del darsi degli obbiettivi comuni e di organizzazione in generale. Ci terrei a sottolineare quello che secondo me è il problema maggiore, ovvero la suddivisione dei compiti. In un'ottica migliorativa e non critica sottolineo che i compiti potevano essere suddivisi in modo più equo, per quanto riguarda la parte a me assegnata, ovvero la gestione dei quiz e le interazioni sono state abbastanza fattibili e andando avanti con lo sviluppo mi sono accorto del tanto model rispetto al minor control e view. Ma guardando ai miei compagni non ho potuto non notare che alcune parti erano veramente più pesanti e complesse da gestire per una sola persona. Per lavori futuri avremo sicuramente più esperienza per poter gestire al meglio anche questo aspetto, che non è l'unico migliorato grazie allo svolgimento del progetto, ma anche le mie capacità di programmazione, di uso degli stream, dei pattern e degli oggetti in generale posso affermare essersi notevolemente sviluppate.

4.1.3 Daniel Capannini

Questa è stata la prima volta in cui ho fatto un progetto di gruppo di programmazione e ritengo che sia stato molto difficile collaborare e certe volte frustrante.

Ritengo che devo migliorare molto nel'uso dei test jUnit e soprattutto di usarli in modo attivo per vedere in anticipo dei possibili futuri bug.

Il lavoro nel complessivo ritengo che in certe parti possa essere riutilizzato in futuro nel caso di progetti simili.

4.1.4 Denis Caushaj

Questo è stato il primo progetto di programmazione che ho dovuto realizzare in gruppo. Purtroppo non mi posso ritenere particorlarmente soddisfatto del risultato finale, poiché mi rendo conto del fatto che il progetto non è ben ottimizzato a causa della poca organizzazione del gruppo, dovuta all'inesperienza di tutti i membri del gruppo. Credo, nonostante ciò, che l'esperienza sia stata molto costruttiva, avendo io preso coscienza di tutto ciò che abbiamo sbagliato dal punto di vista organizzativo: sicuramente mi potrà tornare utile in futuro per poter progettare in gruppo sviluppando un software di qualità.
Oltre al problema organizzativo di cui ho parlato in precedenza, personalmente credo che sia stato altrettanto difficile trovare le forze per lavorare alla mappa, impegno che avevo estremamente sottovalutato durante la divisione dei compiti, tuttavia posso dire di ritenermi particolarmente soddisfatto della parte sviluppata esclusivamente da me. Sono convinto di aver imparato

molto in questi mesi di programmazione, soprattutto per quanto riguarda l'utilizzo degli stream, in quanto mesi fa ero convinto di non essere in grado di poter imparare ad usarli, cosa che invece sono riuscito a fare grazie al progetto stesso.
Infine, per quanto trovi interessante l'idea iniziale del software, non credo che avrà futuro, anche se credo che a tempo perso potrei cercare di completare lo sviluppo.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Denis Caushaj

Ritengo che il corso sia strutturato complessivamente bene, anche se non mi trovo molto d'accordo con alcune scelte. In particolare credo che le lezioni riguardanti l'utilizzo di lambda expression e stream siano state troppo approssimative. Credo che dovrebbe essere dedicato a questi argomenti più tempo sia a lezione che in laboratorio, a discapito di altri argomenti più semplici. Mi sento di consigliare ciò in quanto mi sono trovato in difficoltà nel loro utilizzo per molto tempo, nonostante ad esempio gli esercizi di laboratorio mi siano risultati abbastanza semplici da risolvere.
Inoltre credo che il numero di crediti assegnati non sia ben proporzionato rispetto al numero di ore dedicate allo studio.

Appendice A

Guida utente

Il funzionamanto è molto semplice:

Per il movimento del personaggio si usano i tasti A/S/D/W come avviene in pressoche tutti i giochi.

Per sparare basta cliccare il tasto Space, ma solo mentre si sta combattendo.

Per interagire con i Boss e lo shop basta scontrarsi con loro.

Per aprire il menu è sufficiente premere il tasto esc.

Appendice B

Esercitazioni di laboratorio

B.0.1 Denis Caushaj

- Laboratorio 05: `https://virtuale.unibo.it/mod/forum/discuss.php?d=87881#p138615`