BrickBreaker Evolution

Alessandro Palladino, Alex Testa, Francesco Raso, Giacomo Totaro

A.A 2020-2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

BrickBreaker Evolution è un gioco ispirato al celebre arcade "Breakout", ma in una veste innovativa.
L'utente avrà a disposizione un numero limitato di vite, una barra e una pallina che dovrà far rimbalzare per distruggere i mattoncini. Lo scopo del gioco sarà riuscire a distruggere tutti i blocchi per poter avanzare a livelli sempre più impegnativi, bisognerà però stare attenti alle insidie che si celano all'interno dei mattoncini che potranno renderti il gioco più facile o più difficile a seconda della fortuna(powerup). Il giocatore perderà una vita quando la pallina colliderà con il bordo inferiore dell'area di gioco. Al termine delle vite, il punteggio finale verrà registrato in classifica.
L'utente inoltre potrà creare vari livelli di gioco, i quali avranno diversi temi ispirati a giochi retrò, incluse funzionalità per rendere l'esperienza più impegnativa.

Requisiti funzionali

- All'inizio della partita verrà generata una barra, una pallina e un numero limitato di vite.
- La barra può essere spostata solo orizzontalmente.
- La pallina potrà rimbalzare sulle pareti dell'area di gioco (esclusa quella inferiore), sui mattoncini e sulla barra; a seconda dell'angolazione la pallina assumerà traiettorie diverse.

- Le vite potranno essere selezionate (in base alla difficoltà scelta) nel menù di gioco e diminuiranno ogni volta che la pallina collide con il bordo inferiore. Al termine delle vite la partita terminerà.
- Al momento della distruzione di alcuni mattoncini verranno generati casualmente dei potenziamenti che potranno favorire od ostacolare la partita.
- All'interno del menù sarà possibile creare livelli personalizzati dove l'utente potrà settare, per ogni entità, un tema diverso e di conseguenza cambierà l'area di gioco.
- L'utente potrà inserire il proprio nome e a fine partita visualizzare il relativo punteggio conseguito all'interno di una classifica generale, sarà inoltre possibile consultarla all'interno del menù.

Requisiti non funzionali

- Gestire i dati persistenti tramite salvataggio su file.
- Implementazione della fonica nel menu e durante la partita.
- Creazione automatizzata dei livelli generata in modo randomico.

1.2 Analisi e modello del dominio

L'entità principale del modello applicativo è la GameBoard, che modella il quadrante di gioco in cui vi sono tutti i componenti rappresentati dai GameObject. Quest'ultimi compongono la mappa di gioco e sono: i mattoncini(Brick), una pallina(Ball), i powerup(PowerUp) e la barra (Paddle), impersonata dal giocatore(Player).
Gli elementi costitutivi del problema sono sintetizzati in Figura 1.1.
Le difficoltà principali saranno gestire le collisioni tra le entità e generare randomicamente powerup.
Data la complessità del progetto, in questa prima versione non verrà implementata la feature relativa agli easter egg e la creazioni di livelli randomizzati, la quale necessita di ulteriore studio che non potrà essere effettuato all'interno del monte ore previsto.

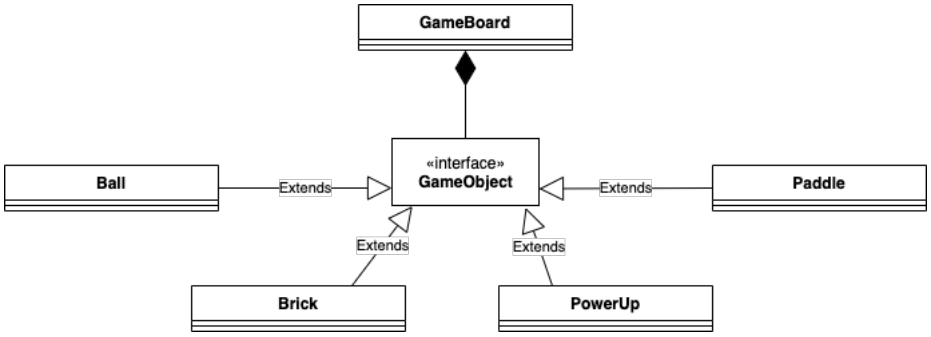


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

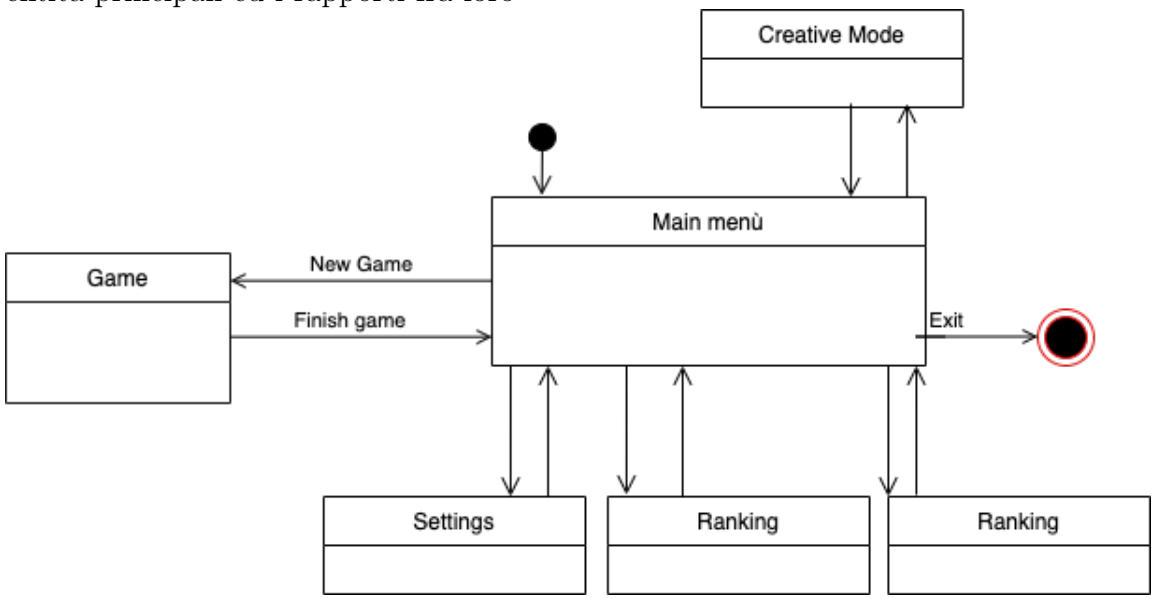


Figura 1.2: Diagramma a stati del videogioco

Capitolo 2

Design

2.1 Architettura

Per questa realtà, dopo un'attenta analisi, abbiamo deciso di implementare il pattern architetturale MVC, per ottenere un chiaro "Separation of Concerns" ovvero una stabile separazione tra oggetti che modellano il dominio applicativo e la loro relativa rappresentazione.

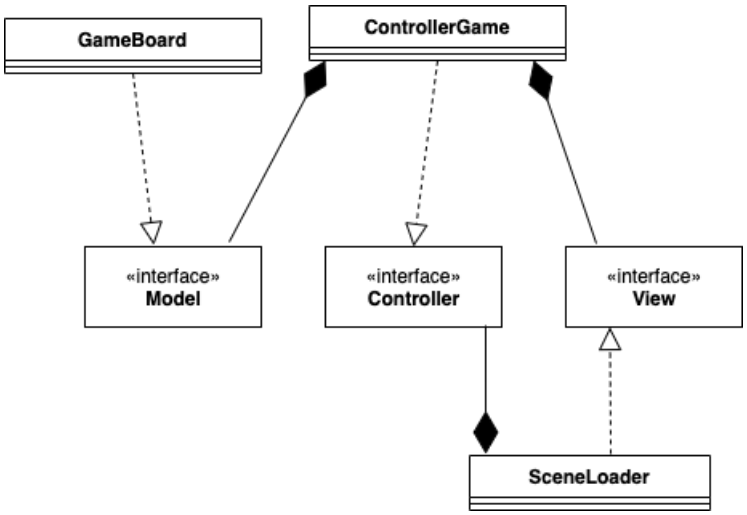


Figura 2.1: Schema UML relativo al MVC

Il Model viene interpretato dal nostro GameBoard che gestisce autonomamente le varie entità e le relative interazioni, rendendo manutenibile l'indipendenza tra le classi.
Quest'ultime, allora, si rendono protagoniste interrogandolo per ottenere informazioni.

Per far interagire Controller e View abbiamo scelto di implementare il pattern GameLoop che, avvalendosi di 3 stati principali : input, render e update eseguiti ciclicamente, compone il motore di gioco.
La View, sfruttando il rendering del GameLoop, riesce ad estrapolare le schermate dell'applicativo. Nella fase di render viene interrogato il GameBoard affinchè crei un contenitore di oggetti che solamente la View, attraverso il Canvas, riesce a disegnare.

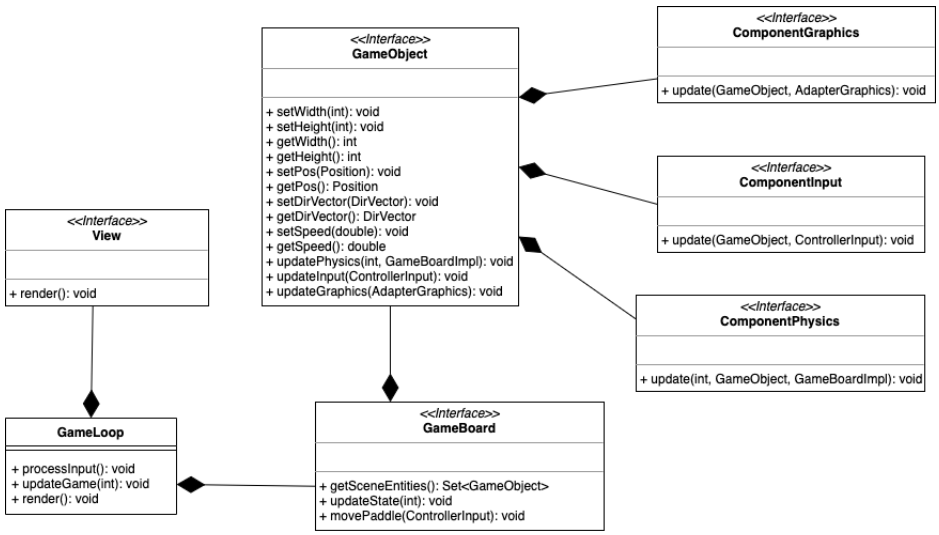


Figura 2.2: Schema UML che rappresenta l'interazione tra View e Controller

2.2 Design dettagliato

Siamo partiti dal Model costruendo il tutto intorno all'entità padre, ovvero GameObject, il quale definisce tutti gli oggetti all'interno del GameBoard. Inizialmente avevamo pensato di utilizzare un pattern Factory ma, riferendoci ad un concetto di estendibilità futura, ci siamo reindirizzati verso l'implementazione di una classe astratta.
A seguito di ciò, abbiamo deciso di adottare il pattern Component per specializzare a 360 gradi le sfaccettature delle entità. Così facendo, pensiamo di essere riusciti a mantenere una corretta separazione dei ruoli rispettando l' architettura.
Riprendendo il concetto visto nel GameLoop, anche il pattern Component si avvale del metodo update() che, dopo un periodo di tempo, avvisa il GameBoard di aggiornare le varie entità e, estendendo una classe astratta, beneficia così di diversi update.

Alessandro Palladino

Siccome siamo partiti dal sviluppare il model, uno dei primi concetti che ho deciso di implementare dato anche l'inesperienza é stato il GameBoard. Esso gestisce tutte le principali operazioni che modellano la nostra "tavola di gioco". Si avvale quindi di diversi metodi per poter aggiungere e rimuovere entità (GameObject) e richiamare i principali eventi che verranno poi gestiti da altri componenti.
Raggiunto una prima fase finale della classe e confrontandomi anche con gli altri, ho ritenuto opportuno iniziare a sviluppare il concetto di livello.
Per poter sviluppare più livelli ho pensato di sfruttare il pattern builder per una facile implementazione di quest'ultimi, attraverso questa scelta sono riuscito a sfruttarlo sia per la creazione dei livelli standard che per l'implementazione della "Creative mode", modalità dove anche l'utente medio può generare il proprio livello. A seguito della creazione di quest'ultimi ho affiancato un gestore che permettesse il salvataggio e il caricamento, e un enum che tenesse traccia dei livelli definiti standard, ossia quelli utilizzati nella modalità di gioco classica.

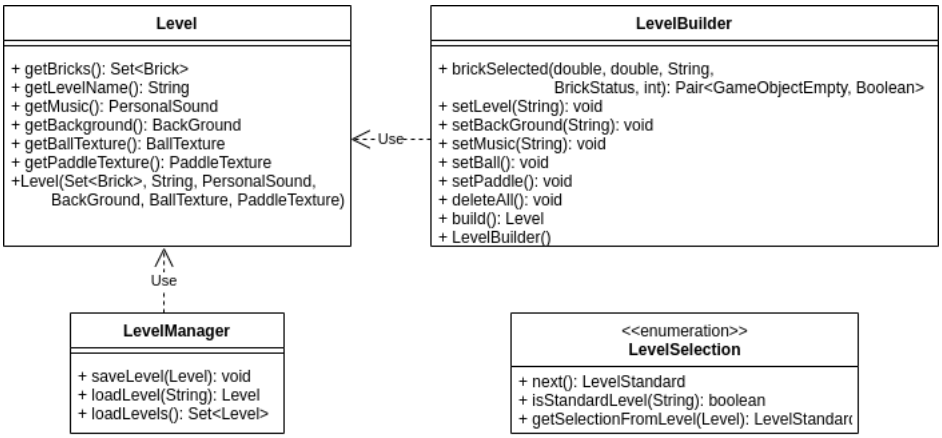


Figura 2.3: Schema UML che rappresenta tutto ciò che riguarda l'implementazione dei livelli.

Facendo un piccolo approfondimento riguardo la creazione del livello ho deciso di sfruttare un costruttore che genera l'oggetto Level contenente le principali informazioni relative ai Brick, Ball, Paddle e alle personalizzazioni grafiche. Questo viene sfruttato dal LevelBuilder che, in base ad una posizione 2d e ad una serie di altri parametri, genererà tutte le caratteristiche necessarie per creare il livello.

Ultimato anche la creazione dei livelli ci siamo resi conto che fosse necessario implementare il motore di gioco. Dopo essermi documentato su internet, aver consultato diversi codici su github e aver esaminato attentamente il funzionamento del seminario svolto dal prof. Ricci ho constatato che probabilmente il miglior pattern applicabile fosse il GameLoop, cosi facendo avremmo avuto un costante aggiornamento della scena che ci avrebbe aiutato con la rappresentazione delle entità presenti. Cercando di implementarlo il più fedelmente possibile ho deciso che il loop fosse scandito principalmente in base a 3 stati possibili: una fase di inizio(start), una fase di pausa(pause, utilizzata per attendere l'input e quindi il vero inizio della partita) e una fase di svolgimento(running). Per impostare però delle condizioni di uscita ho pensato di aggiungere ulteriori fasi che, con degli opportuni controlli, avrebbero reindirizzato verso altre scene(schermata fine gioco, ritorno al menu, ecc...). A seguito della gestione delle fasi e degli oppurtuni controlli il GameLoop quindi si interfaccia principalmente con GameBoard, per quanto riguarda la gestione delle entità, e con GameState per definire lo stato del gioco. Riguardo a quest'ultimo al suo interno vi è lo stato corrente ed in base ad esso si inizializza in modo diverso il processo di generazione della scena, controlla inoltre le impostazioni generali, la tipologia di gioco selezionato e la gestione dello score del player.

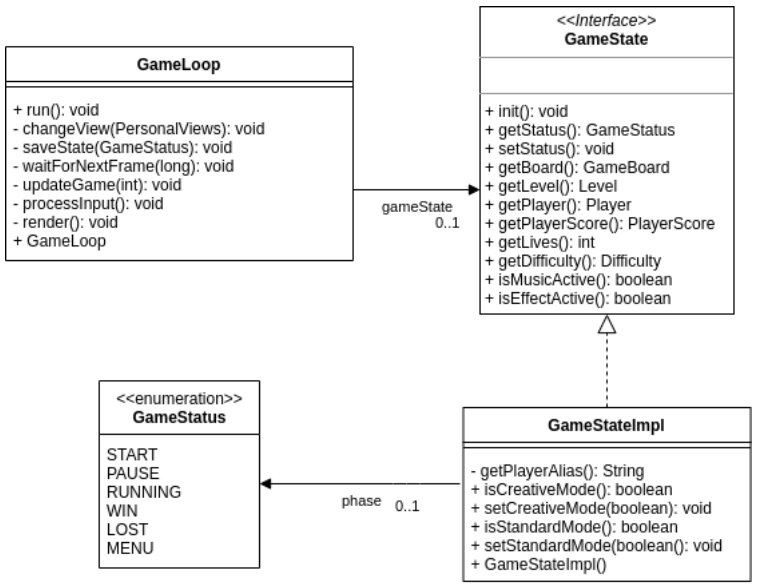


Figura 2.4: Schema UML riguardante il funzionamento di GameLoop.

Arrivati a questo punto mi sono accorto della mancanza relativa al caricamento di un determinato livello e ho deciso in comune accordo con gli altri di implementarlo separatamente. Questa scelta è dovuta al fatto che abbiamo ritenuto più corretto dividere le impostazioni generali come difficoltà e suoni dall'impostazione del caricamento di un livello. Tutto ciò viene quindi utilizzato in fase di salvataggio nella modalità creativa e come caricamento di un livello in base alla modalità di gioco selezionata.

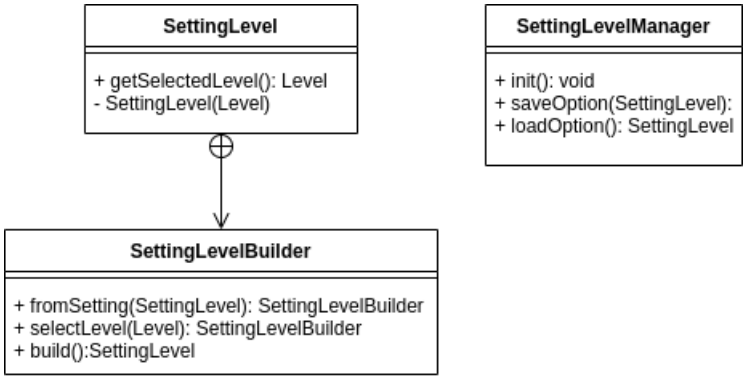


Figura 2.5: Schema UML riguardante la gestione del caricamento del livello.

Per poter implementare questa funzione ho utilizzato ancora una volta il pattern Builder, in questo caso all'interno della stessa classe, in quanto si ritiene fosse abbastanza chiaro il suo funzionamento e che non generasse della confusione. Ad esso ho affiancato un SettingLevelManager che gestisce il salvataggio e il carimento delle opzioni, oltre all'inizializzazione del file contenente il livello di default.

Essendo ormai arrivati verso la fine ciò che restava da implementare erano i controller. Mi sono occupato principalmente di generare la Creative mode, il map editor e il controller del next level(al termine di un livello mostra il resoconto all'interno di una schermata).

Per quanto riguarda l'aspetto della creazione dei livelli personalizzati ho cercato quindi di raggruppare tutte le funzioni possibili per l'utente all'interno della schermata Creative Mode, la quale offre la possibilità di mostrare all'utente i livelli creati da lui, per ogni livello selezionato mostra le caratteristiche e infine la presenza di 3 pulsanti: ritorno al menù, creazione del livello e avvia il livello selezionato.
Passando quindi alla creazione del livello si entra in una nuova schermata dove l'utente può selezionare varie personalizzazioni del gioco. Una volta che

avrà inserito tutti i componenti necessari potrà salvare e successivamente giocare quel livello. Ho cercato di mantenere una visualizzazione dei contenuti molto semplicistica per non creare difficoltà all'utente durante il suo utilizzo.

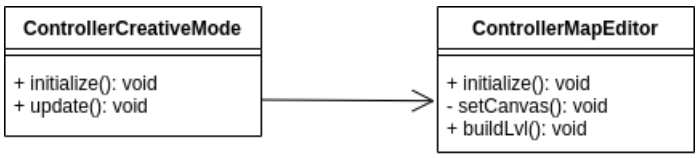


Figura 2.6: Schema UML che rappresenta le due view con all'interno i metodi più rilevanti.

Andando ad esaminare la parte riguardante la personalizzazione inizialmente si era pensato di dividere la rappresentazione dei blocchi del brick da quelli del powerup, scelta che verso la fine ho dovuto riprogettare in quanto generava problemi all'interno dell'userExperience. Ho optato perciò per l'inserimento di un unico comboBox dove l'utente potesse vedere un'anteprima del brick selezionato e affianco ad esso due checkBox dove può decidere se farlo diventare un powerup, oppure renderlo indistruttibile. Selezionando quindi il componente brick esso potrà essere inserito all'interno di una griglia che richiamerà a sua volta il levelBuilder che andrà ad inserire tutte le specifiche nel level. Al termine della creazione del livello vi sono 3 pulsanti: ritorno alla creative mode, salva il livello e pulisci la griglia. Andando a salvare il livello sarà creato nella cartella preimpostata e di conseguenza verrà caricato quando l'utente farà partire quel livello.

Alex Testa

Menù

Personalmente mi sono cimentato nella realizzazione delle schermate riguardanti la gestione del menù di gioco (Menù, Settings, Tutorial, Ranking, Character, Difficulty), seguendo le linee guida del pattern ModelViewController (MVC), consentendomi di dividere la creazione (Model) e la manipolazione (Controller) dei dati, dalla relativa presentazione (View). Per definire uno scheletro comune, i controller riguardanti il menu, si servono di un'interfaccia che li accomuna `FXMLMenuController`. In particolare non mi sono servito dell'utilizzo del pattern Observer, inizialmente implementato, poichè ogni visualizzazione avente il relativo controller è ideato come elemento autonomo, il quale non ha bisogno di comunicare eventi o creare dipendenze tra i vari controller.

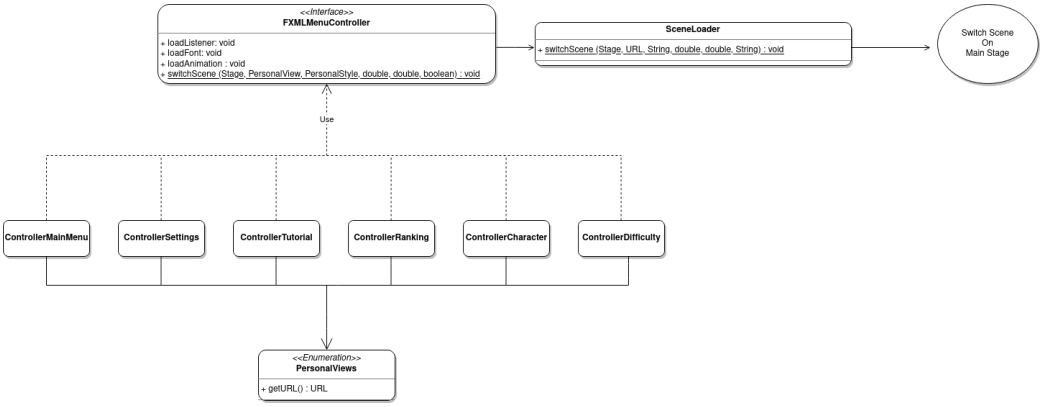


Figura 2.7: Schema UML di un tipico cambio di scena

Principalmente, i controller delegano, il cambio di scena allo stage corrente, attraverso un metodo implementato nell'interfaccia scheletro `FXMLMenuController`, che gli consente di delineare una via comune, evitando ripetizioni esaustive. Inoltre, per facilitare tale cambio, si è pensato di implementare attraverso un' enumerazione `PersonalViews` tutti i percorsi relativi alle varie schermate, consentendo una maggiore efficienza nel caso del cambio di un percorso (evitando l'effetto di errori a cascata) e limitando opportune sviste nell'errata stesura di uno specifico path.

Figura 2.8: Schema UML che rappresenta la delegazione del metodo `switchScene`

Dettagliatamente `FXMLMenuController` delega a `SceneLoader` il compito di cambiare scena all'interno del medesimo stage, consentendo inoltre ad alcune schermate di gioco di non ridimensionarsi. Il grande vantaggio di `SceneLoader` è la totale riutilizzabilità in più contesti anche all'interno del programma.

Giocatore

Un'aspetto di maggior rilevanza durante l'analisi è quello relativo alla gestione della vita e del punteggio, l'idea principale è quella di non separare il concetto di utente dal relativo giocatore, al quale verranno attribuiti un punteggio, un alias (inteso come nome del giocatore) identificativo, e un numero di vite (che sarà limitato ad un numero massimo `n`) e la loro relativa gestione, nello specifico mi sono servito di un pattern `Builder` per la costruzione corretta e semplificata dell'oggetto `Player` in questo modo la creazione è indipendente dalle varie parti dell'oggetto e da come tali vengono assemblate, inoltre le operazioni (intese come modifica) sulla vita e sul punteggio vengono gestite (attraverso un metodo) via `Strategy`, ciò mi consente di variare il comportamento della relativa gestione, eludendo l'ereditarietà e consentendomi in un futuro di definire più strategie per la gestione di tali attributi, aumentando notevolmente la scalabilità. Attualmente le strategie offerte implementano algoritmi che consentono solamente l'operazione additiva o sottrattiva (gestiti in un unica implementazione) con relativo aggiornamento dell'attributo preso in considerazione, in seguito potremmo estenderli per una gestione del gioco più dinamica.

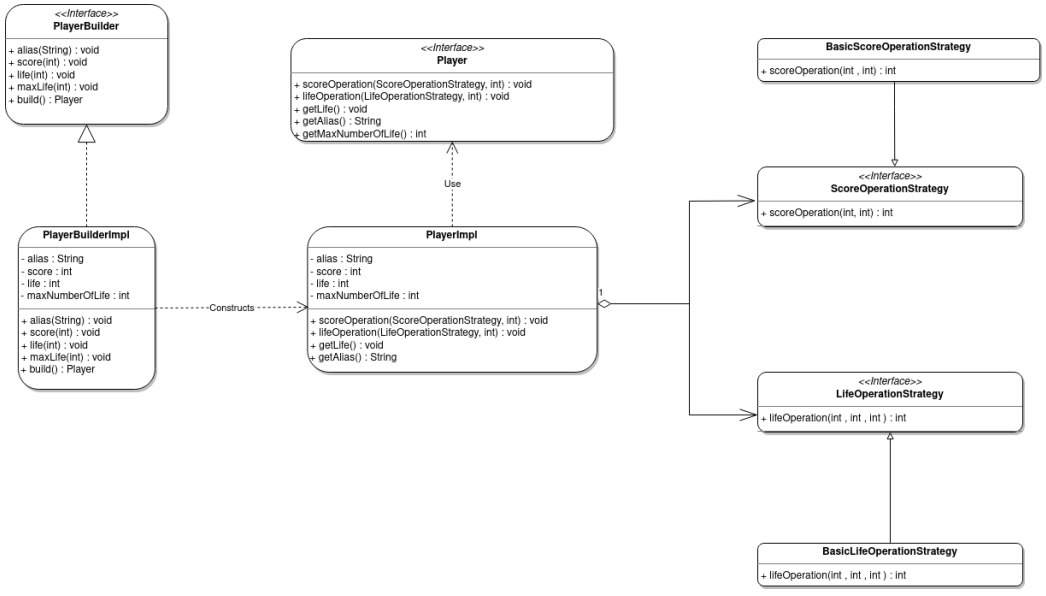


Figura 2.9: Schema UML rappresentativo il giocatore (`Player`)

Classifica

Per la gestione della classifica ho cercato di generalizzarne il concetto, separandolo dal relativo giocatore (`Player`) cercando di accrescerne il riuso, le parti di interesse che la compongono sono, il relativo alias e il punteggio assunto durante la partita, inoltre nella modellizzazione della realtà che stiamo manipolando ho deciso di dare possibilità di inserire giocatori con punteggio nullo (uguale a zero) i quali hanno avviato una prima fase del gioco, senza portarla a termine. Un'altro aspetto fondamentale è l'univocità dell'alias il quale non può essere ripetuto ma bensi' sovrascritto, causando la perdita del punteggio precedente, qualora ve ne sia già uno equivalente presente.
Nello specifico ho ipotizzato che la classifica potesse essere ordinata (secondo un criterio) avvalendomi del pattern `Strategy` il quale mi consente di definire più metodologie di ordinamento, preferendolo all'utilizzo dell'omonimo pattern `Template Method`, poichè mi garantisce una maggiore interscambiabilità nel tempo.
Attualmente l'unica strategia implementata mi consente di riordinare la classifica basandomi sul punteggio in maniera decrescente.
Personalmente ho scelto di evitare di deviare la costruzione di tale oggetto mediante un pattern `Builder` poichè ritengo che tale entità sia abbastanza semplice in fase di costruzione e non disponga di particolari vincoli.

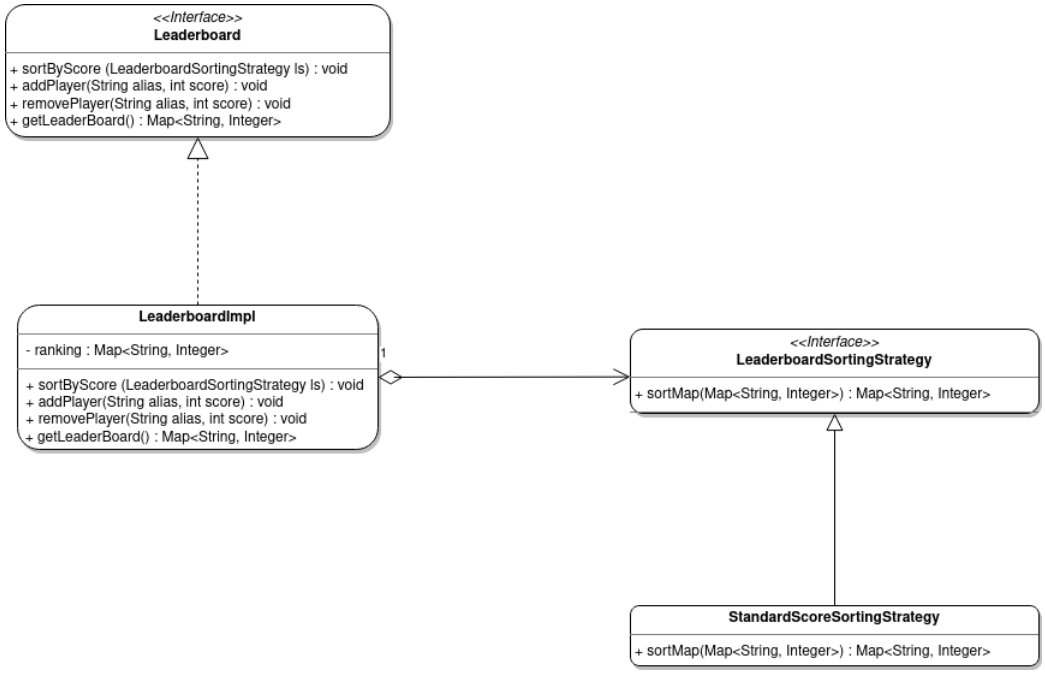


Figura 2.10: Schema UML rappresentativo della gestione della classifica (`Leaderboard`)

Salvataggio

Un'aspetto chiave del gioco, evidenziato in fase di analisi,
è quello di poter salvare il proprio punteggio accumulato durante la partita. Per la gestione di tale compito mi sono avvalso della creazione di una classe di supporto (`IOLeaderboard`) la quale permette il salvataggio dei dati, data una specifica classifica, consentendomi di creare un file dettagliato, contenente tutto l'elenco con i vincoli imposti.
Il formato in cui i dati vengono rappresentati è json, questa scelta mi consente un maggior riuso del file, poichè json supporta una moltitudine di linguaggi, inoltre apre le porte per un possibile sviluppo per lo scenario web, consentendo un notevole riutilizzo e maggior cross-platform.

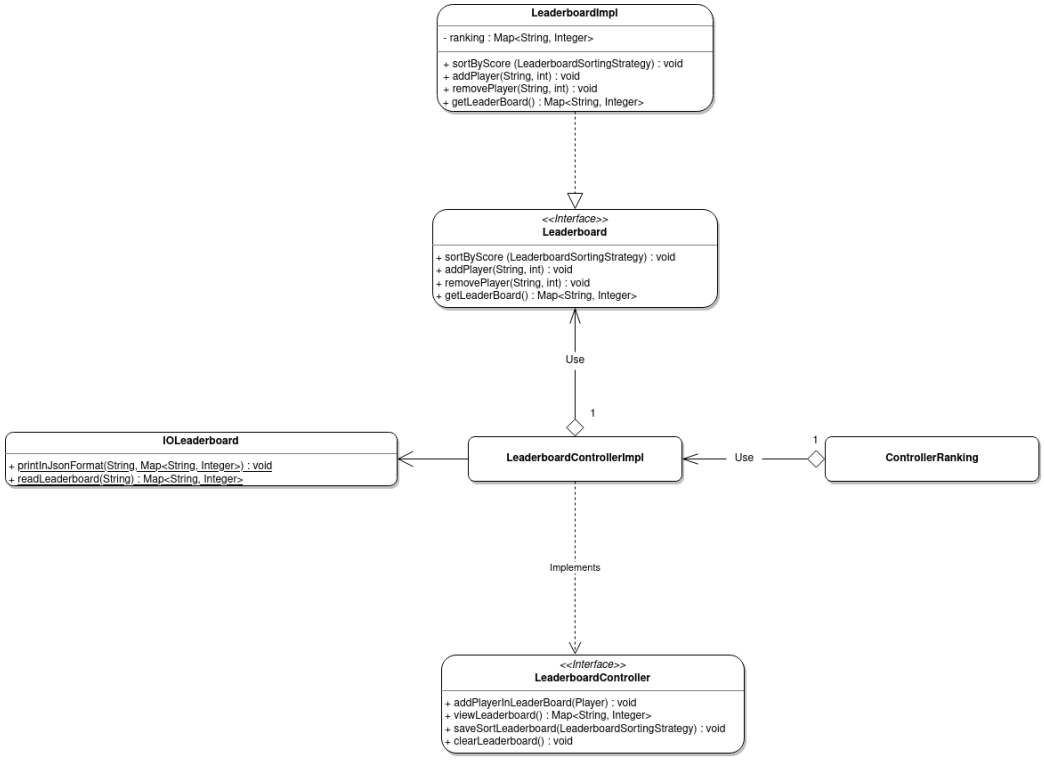


Figura 2.11: Schema UML rappresentativo della gestione della classifica (`LeaderboardControllerImpl`)

N.B Con l'intento di aumentare la riusabilità della parte relativa al controller, ho pensato di creare ulteriori controller aggiuntivi (`LeaderboardController`) che mi permettono di interagire con i dati, offrendo la possibilità di modificarli, leggerli e salvarli.
Questa decisione personale mi consente il totale riutilizzo della parte controller, modificando la parte di rappresentazione grafica.
Sottolineo inoltre la possibilità di operare su più file, per una gestione futura.
(In maniera speculare verranno gestiti i settaggi di gioco con `SettingsController`).

Francesco Raso

Rispettando il pattern MVC, la parte di model prevedeva la creazione delle classi relative ai Brick, per i quali ho utilizzato il pattern Builder. Inoltre, mi sono occupato della creazione e gestione dei PowerUp, istanziati direttamente dai Brick "contenitori" al momento della loro distruzione. Questi ultimi vengono gestiti da un controller dedicato all'interno dell'eventHandler, in grado di attivare/disattivare i PowerUp che andranno a modificare gli attributi di gioco.
Sono previste due tipologie di PowerUp:

PowerUp istantanei:

- LifeUp/LifeDown(aggiunge/rimuove una vita al giocatore).

PowerUp a tempo:

- SpeedUp/SpeedDown(aumenta/riduce la velocità della pallina).
- DamageUp/DamageDown(aumenta/riduce i danni della pallina).

Il tipo di PowerUp viene generato randomicamente dall'enum PowerUpType. Il controller dei PowerUP a tempo usufruisce della classe PowerUpTimer, rendendo il loro utilizzo temporaneo, e disattivandoli al termine del tempo prefissato.
Entrambe le entità sono dotate, secondo il pattern Component, dei componenti grafico, fisico e input.
Sia la componente input che quella fisica dei Brick vengono inizializzate come componenti vuote essendo oggetti privi di movimenti o input da parte del giocatore, mentre quella grafica viene gestita dal controller per permettere alla view di visualizzarli correttamente. I PowerUp hanno invece una componente input vuota, una componente fisica per gestirne il movimento e le collisioni, e una grafica per disegnarla nella board di gioco.

Nella parte di view ho creato le schermate di GameOver e GameFinal con i relativi controlli rispettando il pattern MVC per ottenere dalla Leaderboard il punteggio del giocatore e il miglior punteggio ottenuto in classifica. Ho cercato di mantenere uno stile in linea con il tema del gioco ed ho mantenuto delle schermate semplici ed intuitive.

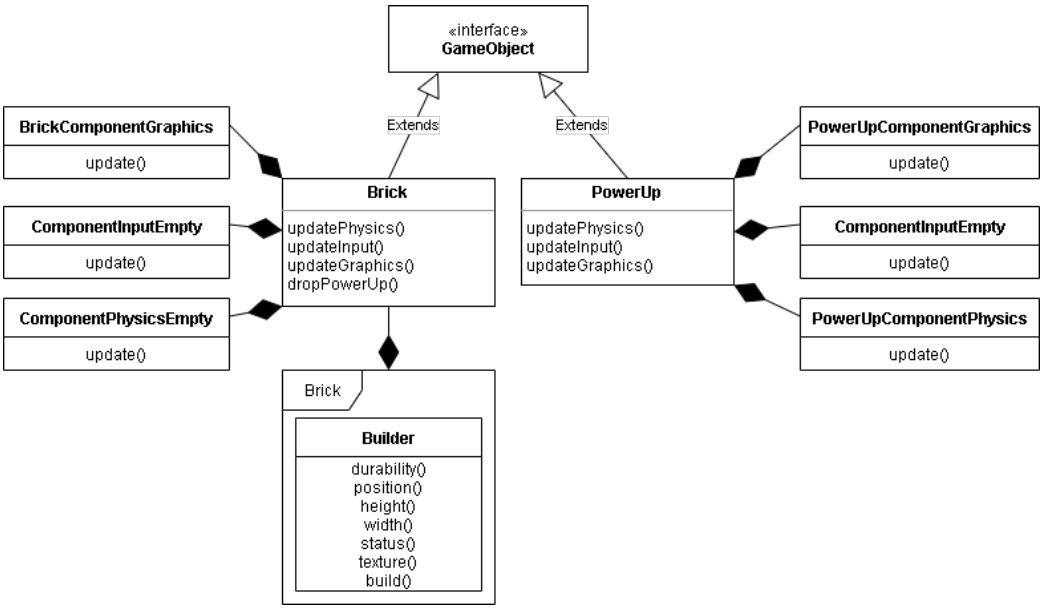


Figura 2.12: Schema UML che rappresenta le entità Brick e PowerUp

Giacomo Totaro

Per quanto riguarda la View, ho realizzato la schermata di gioco, ovvero la sezione in cui l'utente inizia la vera e propria partita, rispettando il pattern MVC. Dopodiché mi sono occupato della realizzazione del paddle e della pallina, con costruttore privato, per i quali mi sono servito del pattern Builder per una creazione veloce e dinamica da parte del Game Board, quest'ultimo gestito da Alessandro. Per entrambe le entità ho prodotto il relativo componente grafico, input e fisico seguendo il pattern Component.
Componente fisico: ogni volta che quest'ultimo viene aggiornato, questo chiede al GameBoard se è avvenuta una collisione (dato che è a conoscenza di tutte le entità presenti nel gioco) e ne calcola la direzione.
Componente input: la sola entità che può essere controllata dall'utente è il paddle mentre la pallina viene controllata tramite il rimbalzo.
Componente grafico: questo mantiene dentro sé tutte le informazioni necessarie per sapere come disegnare se stesso sulla view attraverso lo strumento fornito dal canvas che fà da "pennello".

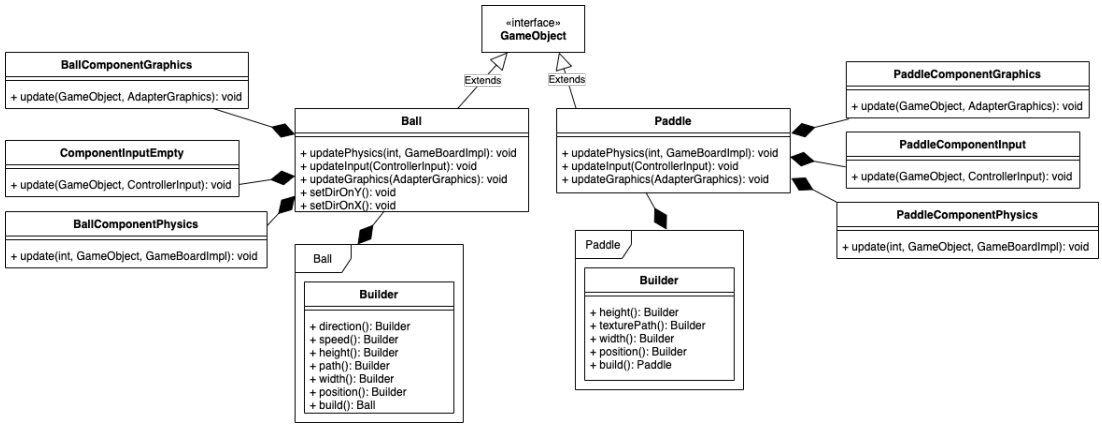


Figura 2.13: Schema UML che rappresenta l'entità Ball e Paddle

Francesco Raso, Giacomo Totaro

Insieme abbiamo collaborato per la realizzazione delle collisioni.
Esse vengono rilevate dal collision controller che calcola tutte le possibili interazioni tra due GameObject che possono presentarsi nel gioco.
Dopodiché il Game Board comunicherà l'avvenuta collisione ai Game Object che notificheranno l'evento alla Board stessa solo se c'è stato realmente un contatto. In caso affermativo, l'evento viene aggiunto in una coda e sarà un'istanza dell'interfaccia Event e in futuro potrà tornare utile aggiungere un'implementazione per portar con sé delle informazioni riguardanti l'evento specifico.
Questa coda di eventi viene risolta dal manageEvent() ad ogni update della Board, il quale funziona da controller per manipolare gli attributi dei GameObject. Per la gestione (notifica) degli eventi abbiamo utilizzato il pattern Event Queue.

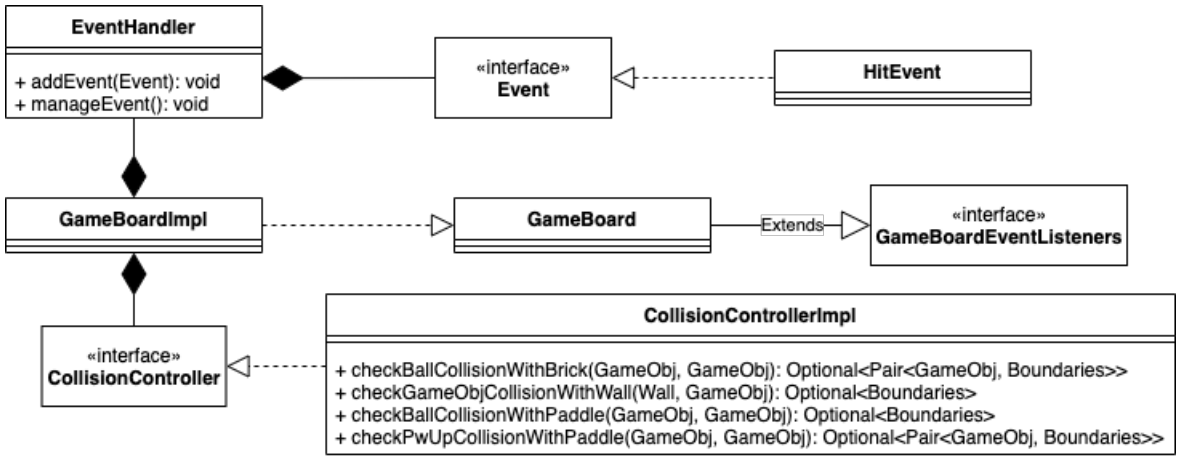


Figura 2.14: Schema UML che rappresenta le collisioni con la gestione degli eventi

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per poterci assicurare che il nostro lavoro funzionasse, ognuno di noi ha implementato test riguardanti i rispettivi compiti per ricevere feedback su parti complesse da controllare. Si è scelta una suddivisione in package stile MVC per una questione di corrispondenza delle classi.
Insieme si è deciso di utilizzare la libreria `JUnit` e `TestFX`.

Alessandro Palladino

Per quanto riguarda la mia parte, ciò che ho testato automaticamente riguarda la creazione delle entità all'interno della board affinché tutti gli elementi fossero istanziati correttamente. A seguito di questo test ho testato manualmente la giocabilità e fin da subito si è riscontrato un problema di gestione delle risorse che rallentava il gioco rendendo l'esperienza dell'utente finale sgradevole. Per ovviare al problema è stata esaminata la parte di drawing e sono state effettuate delle correzioni, oltre a questo si è anche deciso di reperire nuove risorse per poter alleggerire il più possibile il carico di lavoro del GameLoop. In parallelo ho controllato la gestione dei salvataggi dei livelli attraverso fasi di debug durante l'implementazione stessa e, essendo risultato tutto corretto fin dall'inizio, non ho ritenuto indispensabile implementare la parte di test automatizzato.

Alex Testa

Di seguito riporto i test che sono stati svolti seguiti da una breve descrizione:

Test eseguiti		
Classe	**Tipo**	**Descrizione**
TestLeaderboard	Model	Test usato per verificare il corretto funzionamento della classifica, in particolare, si verifica la corretta creazione, la corretta aggiunta di un giocatore, la corretta sovrascrizione di un giocatore, la corretta rimozione, il corretto ordinamento tramite `Strategy` pettern
TestBuilder Settings	Model	In particolare viene testato, la corretta creazione di settaggi predefiniti del gioco, la corretta creazione dei settaggi, la corretta creazione dell'oggetto che rispetti i vincoli imposti dal `Builder` (La difficoltà non può essere un valore nullo, Le due modalità di controllo della tastiera devono essere complementari).
TestPlayer	Model	Utilizzato per testare, il corretto incremento/decremento del punteggio e della vita, l'utilizzo della programmazione funzionale riguardante le operazioni di `Strategy`, l'equivalenza di due giocatori uguali, la corretta iniziallizzazione del punteggio, il corretto funzionamento del metodo che mi permette di stampare a video il giocatore.
TestPlayer Builder	Model	Utilizzata per verificare la corretta creazione dell'oggetto player, in particolare il controllo su punteggi negativi, vite negative e alias nulli.
TestLeaderboard Controller	Controller	Utilizzato per verificare la corretta aggiunta di un giocatore, la visualizzazione di un corretto podio (in ordine decrescente), il corretto salvataggio della classifica e la corretta cancellazione della classifica
TestPrint Leaderboard	Controller	Test del corretto salvataggio e lettura della classifica
TestPrintSettings	Controller	Test del corretto salvataggio e lettura dei settaggi di gioco
TestMainMenu	View	Test utile per verificare il corretto cambio di scena e le corrette interazioni (con scambio di dati) tra le varie schermate.

Inoltre sono stati condotti svariati test riguardanti la nota UX(User Experience), ciò mi consentiva di capire se un utente comune, sarebbe stato in grado, senza la lettura del manuale specifico riguardante l'applicativo,
di poter addentrarsi nel gioco compiendo azioni come :

- Avvio di una partita.
- Modalità "Fast Game" (feature descritta nel successivo paragrafo).
- Cambio dei settaggi di gioco.
- Uscita dalla schermata attuale.
- Tornare alla schermata precedente.

I test sono stati condotti attraverso una telecamera e saranno resi disponibili al docente se interessato.
Ringrazio di cuore lo studente di Ingegneria Aerospaziale Corbelli Marco, per essersi prestato come cavia.
(Non sono stati eseguiti test sulla parte dei settaggi di gioco poichè considerati come elementi secondari)

Francesco Raso

La mia parte di testing ha previsto il controllo sulla creazione dei Brick attraverso il builder, e che la loro durability venisse correttamente decrementata. Ho testato la creazione dei PowerUp controllando che inizializzasse correttamente la texture da visualizzare in funzione di quella ereditata dal Brick di provenienza. Per testare il movimento del PowerUp l'ho inserito nella GameBoard per verificare che, sfruttando la frequenza di aggiornamento, venisse correttamente aggiornato il componente fisico e quindi la sua posizione.

Giacomo Totaro

Per quanto riguarda le entità da me create, ho testato la corretta creazione del paddle usando il relativo builder, controllando che segnalasse errore in caso di informazioni sbagliate.
Ho verificato le corrette collisioni con il Wall, il corretto settaggio delle direzioni in base all'input dell'utente e che si muovesse di una quantità dipendente dalla sua velocità e dalla frequenza di aggiornamento impostata dal Game Loop.
Per il test sulla pallina ho verificato che la creazione attraverso il builder desse esito positivo nel caso di informazioni giuste e esito negativo in caso contrario. Successivamente ho inserito la pallina nel Game Board e poi ho controllato l'aggiornamento del componente fisico in modo da verificare se la posizione finale fosse quella desiderata.
Il controllo sulla velocità è stato eseguito nello stesso modo.

Francesco Raso, Giacomo Totaro

Insieme abbiamo testato il corretto funzionamento delle collisioni, e della comunicazione delle entità in causa. Ci siamo focalizzati su tutti i vari tipi di collisioni, per verificare che il lato in cui si scontrano e la direzione finale dei vari GameObject fosse giusto.

3.2 Metodologia di lavoro

A seguito di una fase di analisi e di design svolta in gruppo ci siamo resi conto di dover riorganizzare la suddivisione del lavoro come suggerita dal prof in fase di presentazione del progetto. Siamo partiti quindi dalla realizzazione dello scheletro MVC, implementando parallelamente la parte di model e controller. Avvalendoci di Teams e del DVCS siamo riusciti a implementare parti diverse di codice che, periodicamente venivano revisionate in gruppo per assicurarci una corretta implementazione. Poichè alcune parti erano strettamente connesse, si è deciso di svilupparle in contemporanea confrontandoci affinchè il tutto potesse unirsi correttamente.

Alessandro Palladino

Le classi che ho implementato appartengono a sottosezioni diverse, ciascuna delle quali svolge un ruolo diverso andando a ricoprire tutte le parti dell'MVC.

- controller.game: (Con il contributo in alcune parti seppur piccole di tutti) GameLoop e le classi correlate come GameStatus, GameState e GameStateImpl.
- model.mapeditor: Level, LevelBuilder, LevelStandard, LevelManager.
- controller.scene: ControllerCreativeMode, ControllerMapEditor, ControllerNextLevel.
- Classi accessorie come GameObjectEmpty, ComponentInputEmpty, ComponentPhysicsEmpty. ComponentGraphicsEmpty
- CheckCustomAlert (Creata per ridurre codice nel controllo dell'inserimento di campi all'interno dell'editor).
- Realizzazione del GameBoard e GameBoardImpl.
- Creazione e gestione di tutto il pacchetto texture (BackGround, BallTexture, BrickTexture, PaddleTexture, PowerUpTexture, PowerUpDropTexture) con l'aggiunta di alcuni metodi da parte degli altri del gruppo.
- TextureComboBox (Necessaria per poter visualizzare le anteprime all'interno di ogni ComboBox nella schermata Editor),
- model.settings: SettingLevel e SettingLevelManager.

Alex Testa

- View : In particolare il mio lavoro è stato incentrato sulla creazione del menù nelle relative schermate di : Menù Principale, Settaggi, Classifica, Selezione dell' alias, Selezione della difficoltà di gioco e creazione del tutorial (in collaborazione con `Francesco Raso`).
  Inoltre il mio compito era quello di creare lo stile del gioco (stile pulsanti, label, immagini di sfondo, gestione generale della scena, animazioni, effetti di transizione) e di delineare un'interfaccia comune a cui i miei colleghi avrebbero preso spunto per la creazione delle loro relative schermate rendendo uniforme il gioco. Aspetto non meno importante è dato dalla user experience al quale ho dedicato gran parte del tempo, inserendo un modalità che chiamo "Fast Game" la quale permette al giocatore di avviare una partita in maniera rapida, senza impiegare tempo nell'immissione dell'alias relativo, etichettandolo come "Guest".

- Model : Nello specifico mi è stato assegnato l'obbiettivo di gestire le entità che avrebbero definito i settaggi di gioco, la gestione del giocatore e la gestione della classifica portandomi sopratutto a definire vincoli ben precisi per la creazione e gestione di essi.

- Controller : La parte di controller che mi è stata assegnata riguarda la particolare gestione del model sopra riportato, più dettagliatamente ho gestito il salvataggio, la corretta lettura e la corretta manipolazione dei dati. Inoltre ho gestito la parte inerente alla fonica di gioco.

Francesco Raso

- Creazione del Brick e i relativi componenti.

- Creazione dei PowerUp e suoi componenti, settaggi, e controller.

- Creazione della schermata di gioco finale e di GameOver con punteggio del giocatore e miglior punteggio in classifica.

Giacomo Totaro

Partendo dalla view ho realizzato parte del package view.graphics e ho utilizzato un'interfaccia chiamata AdapterGraphics. Quest'ultima prende in input le dimensioni e le posizioni delle entità, inclusa la loro immagine, le converte nelle dimensioni giuste e le disegna.
Inoltre ho avuto la possibilità di creare la view di gioco, ovvero la finestra in cui l'utente interagisce con i GameObject e vede il riepilogo delle vite, dello score e del tipo di powerUp attivo.
Passando al model mi sono occupato dell'implementazione del paddle e della pallina, che risiedono in model.entities, inclusi i loro relativi componenti: fisici (package model.physics), grafici (quello di cui parlavo sopra) e di input (package controller.input).

Francesco Raso, Giacomo Totaro

Insieme abbiamo realizzato l'intero package model.collision che, appunto gestisce tutte le collisioni all'interno del mondo di gioco.
Inoltre, anche il package controller.event è stato implementato da noi, il quale, sfruttando una coda di eventi, gestisce questi ultimi.

3.3 Note di sviluppo

Alessandro Palladino

- Ho utilizzato diverse fonti per capire come poter impostare vari aspetti del progetto e a tal scopo cito: seminario Game-as-a-Lab tenuto dal prof Ricci, diversi progetti su github riguardanti ambito videoludico, diversi post su StackOverflow riguardo svariate implementazioni, tra le quali quelle relative alle anteprime del CheckBox, diverse pagine sul pattern GameLoop.
- Uso di stream
- Uso di lambda expression
- Uso di method reference
- Uso di Optional
- Uso di librerie JavaFX per creare delle logiche applicative.
- Uso di file FXML per creare la presentazione delle schermate.

Alex Testa

- Per la realizzazione delle schermate principali sopra elencate mi sono servito della libreria `JavaFx` in quanto mi ha facilitato una corretta scissione tra model view e controller, inoltre mi ha permesso la costruzione di schermate moderne più accattivanti.
- Per la stampa tramite formato `JSON` ho utilizzato la libreria esterna concessa da Google (`GSON`), in quanto mi forniva la serializzazione di un oggetto Java in JSON, e mi consentiva di poterlo deserializzare una volta letto dal file.
- Per la realizzazione dei test grafici mi sono servito della libreria esterna `TestFx` la quale mi permetteva, in maniera semplificata, di testare approfonditamente le componenti grafiche.
- Per ottimizzare l'utilizzo della libreria esterna `JavaFx`, ho utilizzato la documentazione oracle riportata : Implementing JavaFX Best Practices, dalla quale ho appreso l'ottimizzazione riguardante i fogli di stile, la corretta gestione dell'MVC, e la scelta dell'inutilizzo del componente `PreLoader` non necessario nella nostra realtà.

- L'aspetto più importante è stato l'utilizzo di `Stream` e della programmazione funzionale, consentendo di semplificare e rifattorizzare il codice, rendendolo più efficiente e comprensibile, sopratutto nella gestione della classifica e delle varie operazioni gestite nel `Player`.
- Ultimo riferimento va nella gestione della User Experience citando un testo molto utile per rendere intuitive le componenti grafiche : Don't Make Me Think

Francesco Raso

Data la semplicità del progetto, probabilmente anche dovuta ad una sbilanciata ripartizione del lavoro tra i componenti del gruppo, e a parte la sezione sviluppata in collaborazione con Giacomo, non vi è stata necessità di utilizzare algoritmi di grande rilievo nella mia parte.

Per questo motivo ho cercato di gestire al meglio le entità da me create e renderle maggiormente compatibili e complementari al lavoro dei miei colleghi, in particolare quello di Alessandro nella creazione dei livelli di gioco. Da qui la scelta di utilizzare il pattern Builder per la costruzione dei Brick, contro la scelta di mantenere un costruttore semplice e meno dinamico per la creazione dei PowerUp. Questo deriva in parte da una scelta di progettazione iniziale che prevedeva l'entità PowerUp come estensione del Brick, e che avrebbe creato dei problemi di incompatibilità con le componenti previste per i Brick stessi.

- **JavaFx**: utilizzata per l'interfaccia grafica.
- **Lambda expression**: utilizzate nella gestione degli eventi e di controllo dei parametri modificati dai PowerUp.
- **Optional**: anche queste utilizzate nei controlli delle collisioni e degli eventi.

Giacomo Totaro

Per la definizione del vettore, la classe DirVector è stata presa dal seminario game-as-a-lab.

- **Stream**: sono una delle feature più interessanti di Java perchè fa risparmiare tante righe di codice.
- **Generici**: usate nella classe Pair per renderla adatta ad una gestione uniforme.
- **Lambda expression**: utilizzate per avere codice leggibile e pulito.
- **Optional**: utili nella chiarezza del codice, in più per evitare null come valore di ritorno.
- **JavaFx**: utilizzata per la creazione dell'interfaccia grafica.

Francesco Raso, Giacomo Totaro

In collaborazione abbiamo deciso di implementare le collisioni nel collision controller e siamo riusciti a trovare su Youtube, dopo varie ricerche, l'algoritmo AABB(Axis-Aligned-Bounding-Box). E' stato scelto questo per due motivi: innazitutto abbiamo a che fare con oggetti rettangolari che rimangono statici per l'intera durata della partita (ovvero non ruotano mai), poi i loro spigoli sono sempre paralleli agli assi di riferimento e questo algoritmo è meno complesso di altri da calcolare, perché fa semplici controlli sulle posizioni delle entità coinvolte.
La gestione degli eventi è stata appresa dal seminario game-as-a-lab e poi riadattata alla nostra applicazione.
Le feature avanzate utilizzate sono:

- **Stream**: usati nella gestione degli eventi per semplificare e rendere leggibile il codice.
- **Optional**: utilizzati per verificare l'eventuale collisione.
- **Lambda expressions**: usati per il controllo delle collisioni.
- **Generici**: usati nella classe Pair per differenziare il passaggio di informazioni delle collisioni.

Capitolo 4

Commenti finali

Considerando l'inesperienza del gruppo nel generare un progetto così esteso, ci riteniamo soddisfatti del lavoro eseguito anche se ci siamo resi conto che è ulteriormente migliorabile. Tra tutte le difficoltà incontrate nello sviluppo del codice, riteniamo opportuno includere anche il fattore distanza che ha influito parecchio a livello organizzativo.
Tuttavia, questo elaborato ci ha fornito una concreta visione nel mondo del lavoro.

4.1 Autovalutazione e lavori futuri

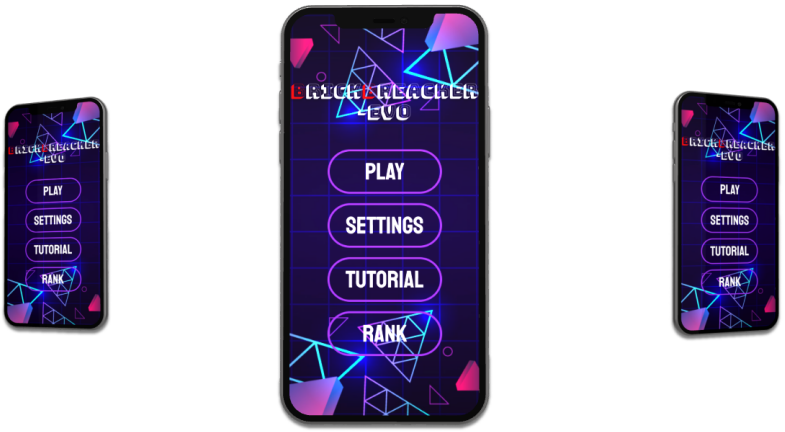
Alessandro Palladino

Innanzitutto sono fiero di essere riuscito a portare a termine un compito così grande in quanto non avendo mai affrontato nulla del genere mi sono sentito all'inizio un po' perso. Fortunatamente sono riuscito insieme al mio gruppo a redigere un piano lavorativo e questo ha fatto si che ognuno avesse il suo ruolo. Sono state presenti delle divergenze ma le abbiamo sempre risolte in modo molto logico, andando a confrontarci e a comprendere i nostri errori. Nonostante tutto credo che sia uscito un bel progetto, sicuramente migliorabile sotto ogni aspetto. Durante le fasi di sviluppo tutti noi abbiamo avuto delle idee, alcune siamo riusciti ad implementarle subito, altre purtroppo richiedendo più tempo del monte ore previsto non siamo riusciti a svilupparle. Sicuramente parlando dei miei aspetti ho previsto un futuro cambiamento del salvataggio dei file in un formato leggibile, questo perché cosi risulta più comodo anche all'utente verificare l'interno del suo file. Per quanto riguarda l'autocritica posso dire che questo progetto mi ha aperto molte vie sotto parecchi punti di vista e grazie ad esso mi sono addentrato nel mondo della

programmazione e del lavoro. Posso dire che ho cercato di mettere il massimo impegno, sicuramente parecchie cose potevano essere implementate in maniera più smart ma comunque non potendo dedicarci troppo tempo per non trascurare le altre materie mi è toccato rinunciare in qualche miglioria che sarebbe stata possibile.

Alex Testa

Mi ritengo molto soddisfatto riguardo all'applicativo prodotto, in particolare la fase in cui ho dovuto consultare la documentazione (JavaFx, Gson, TestFx, Javadoc) per poter creare il prodotto finito, lo ritengo un aspetto altamente fondamentale ma sopratutto molto utile per lo sviluppo di software e di crescita personale. Autovalutandomi ritengo di aver sviluppato un ottimo software, ma con qualche miglioria, sicuramente vorrei eliminare i suppress Warning, i quali mi hanno destabilizzato ma, per motivi legati alla tempistica, non sono riuscito ad eliminare. Inoltre grazie all' utlizzo del pattern mvc sono riuscito a rendere il software totalmente riutilizzabile dando la possibilità di non "reinventare la ruota" ma di poterla migliorare. Il punto di forza sul quale si è basato la maggior parte del mio lavoro, è stato quello di rendere la rappresentazione del gioco il più intuitiva possibile, utilizzando regole di design, per uniformarmi agli standard grafici moderni, in aggiunta, l'aspetto fondamentale, è stato quello di rendere i componenti del software (Settaggi, Classifica, Giocatore) altamente riutilizzabili anche per scopi basati su altre interpretazioni come piattaforme web. Un' altro aspetto di miglioramento è dato dalla possibilità di cambiare più stili all'interno del videogame, dando la possibilità all'utente di personalizzare le schermate secondo i propri gusti. (In programma vi è anche lo sviluppo della Guida Utente, in formato libretto, come in un videogame realistico).



Coming Soon

Francesco Raso

Questo progetto è stato un chiaro esempio di come dovrebbe essere l'approccio di noi studenti al mondo del lavoro, e mi ha fatto rendere conto di come la collaborazione e le scelte progettuali influiscano nella creazione di un software che si possa definire ben fatto. Inoltre ho trovato questo progetto una buona occasione per misurare le mie capacità con gli altri componenti del gruppo e capire quanto le competenze in relazione ad una corretta ripartizione dei compiti siano indispensabili per approcciare questo lavoro in un ottica strettamente professionale.

Guardando al futuro, molte implementazioni potrebbero subire modifiche e migliorie. Ad esempio si sarebbero potuti specializzare i PowerUp in due sottocategorie per distinguere quelli da cui il giocatore trae beneficio da quelli che rendono la partita più impegnativa. Lo stesso ragionamento potrebbe essere fatto per specializzare gli eventi di gioco che per questioni di tempo sono stati limitati al controllo delle collisioni, ma che potrebbero venire adattati per aggiungere nuove sfide per il giocatore.

Giacomo Totaro

Nel complesso sono contento del lavoro svolto. Giustamente ho avuto alcune difficoltà e solo ora ho capito quanto sia importante fare una scelta progettuale e un analisi ben dettagliata prima di iniziare il lavoro. Ci sono alcune cose che avrei potuto gestire in maniera diversa e forse migliore, come ad esempio generalizzare le entità scomponendoli in interfacce così da avere una migliore separazione tra interfaccia e la sua classe.
Vorrei fare un accenno all'utilizzo del DVCS che, inizialmente non piaceva, poi invece è risultato fondamentale.

4.2 Difficoltà incontrate e commenti per i docenti

Alessandro Palladino

Le principali difficoltà riscontrate sono relative alla gestione dei laboratori. Ripensando a come li vedevo prima del progetto e dopo il progetto il modo in cui si affrontano gli argomenti lo ritengo migliorabile.
Partendo dalla visione a fine progetto direi che le slide dei laboratori sono impostate benissimo; informazioni chiare, precise e che una volta che sai come funzionano sai subito andare a riprendere. Quando invece vengono somministrate esse sembrano trattare argomenti poco interessanti al fine del programmare, cioè sembrano essere utili ma non indispensabili (cosa che a fine progetto è l'esatto opposto). Probabilmente questo può essere dovuto al fatto che vengono spiegate molto velocemente perchè comunque i concetti che vengono introdotti sono molti, molto complessi e con importanti approfondimenti.
Suggerisco quindi magari di trattarli meglio integrandoli con le prove pratiche, ma cambiando approccio, almeno per le prime settimane. Le istruzioni degli elaborati spesso sono risultate difficili da comprendere, probabilmente con un pdf per ogni esercitazione dove si spiega cosa si vuole e come bisogna fare potrebbe essere una buona soluzione per impostare almeno il mind-set. Dopodichè credo che abituandoci ogni settimana si possa progredire con uno step di difficoltà successivo fino ad arrivare ad usare gli strumenti che citate durante i laboratori.
Detto ciò ho ritenuto comunque molto interessante il corso che mi ha aperto nuovi orizzonti e ringrazio l'opportunità di aver potuto sviluppare un progetto di tale dimensione, essendo il mio primo progetto non è stato facile cimentarsi in questa impresa, e una volta che si è finito ci si accorge di essere

migliorati ancora di più rispetto alla fine delle lezioni, probabilmente perchè ci si immerge di persona in ciò che è stato spiegato per mesi.

Alex Testa

Personalmente ritengo che il corso sia il più valido all'interno di questa facoltà, mi sono innamorato della programmazione con paradigma ad oggetti, e mi sono ritrovato a rivalutare, in maniera positiva, il linguaggio Java, un aspetto che ha influito negativamente, è stata la presenza di poche ore di corso dedicate all'utilizzo di software come Gradle e Git e JavaFx, i quali reputo strumenti altamente complessi ma molto efficaci. In conclusione ritengo le ore di lezione altamente valide ma, noto un' abisso tra gli esercizi presentati, a lezione e in laboratorio, rispetto a quelli presentati durante le prove d'esame.(Ringrazio calorosamente i docenti del corso, i quali mi hanno suscitato curiosità ed interesse per la materia).

Francesco Raso

Le difficoltà principali sono state la ripartizione dei compiti e il conseguente crearsi di "code di attesa" sul lavoro che risultava complementare a quello degli altri membri del gruppo. Ho trovato il progetto tanto stimolante quanto impegnativo, anche se in maniera sbilanciata in confronto al monte ore previsto per il suo stesso completamento.

Giacomo Totaro

In primis vorrei dire che sono molto felice dell'approccio serio, organizzato e specifico del corso e dei docenti. Ho imparato tantissime cose utili che prima, nonostante l'esperienza alle scuole superiori, ignoravo. Un difetto di questo corso, per me, è il vasto numero di contenuti, tutti molto essenziali per la vita professionale di un ingegnere informatico ma di cui bisognerebbe dedicare altri corsi aggiuntivi.

Appendice A

Esercitazioni di laboratorio

Alessandro Palladino

- Laboratorio 04: https://virtuale.unibo.it/mod/forum/discuss.php?d=62685#p101241
- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=62684#p101164
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p105681
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p102529
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p120714
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p103956
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=66753#p106744

Alex Testa

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p100901
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p102837
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p104083

Giacomo Totaro

- Laboratorio 04: https://virtuale.unibo.it/mod/forum/discuss.php?d=62685#p101584
- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=62684#p101542
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p101488
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p100925
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p103037
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p103907
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=66753#p106667
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=66463#p108394