RELAZIONE PROGETTO DI PROGRAMMAZIONE AD OGGETTI

Margherita Balzoni, Chiara Castiglioni, Edoardo Desiderio,
Virginia Foschi, Simone Ruggeri

Febbraio 2023

Indice

Capitolo 1

Analisi

Il seguente capitolo contiene l'analisi dei requisiti e del problema, ovvero fornisce una descrizione del dominio applicativo e delle funzionalità offerte dall'applicazione.

1.1 Requisiti

Il progetto, commissionato dall'Università di Bologna[1], si pone come obbiettivo la realizzazione di un videogioco del genere Arcade, di nome Arkanoid[2]. L'obbiettivo del gioco è quello di superare i tre round per ogni livello di difficoltà cercando di totalizzare un punteggio elevato, abbattendo un certo numero di mattoncini colorati colpendoli con una sfera.

Requisiti funzionali

- Menù principale che all'avvio del software permette di scegliere il livello in base alla difficoltà, visionare la classifica e visualizzare i comandi.
- L'applicazione implementa la fisica della palla e ne gestisce il rimbalzo con i bordi dell'arena, con il pad e con i mattoncini.
- Area di gioco composta da un certo numero di mattoncini che cambiano disposizione ad ogni livello.
- Gestione dei parametri del giocatore, in particolare vita (che diminuisce ogni volta che la pallina esce dall'arena di gioco) e il punteggio.

[1] `https://www.unibo.it/it`

[2] `https://it.wikipedia.org/wiki/Arkanoid`

- Organizzazione del gioco a livelli ciascuno composto da 3 round
- Creazione di differenti bonus o malus
- Avanzamento del gioco con difficoltà incrementale (quantità di mattoncini aumentata per ogni round)
- Realizzazione e aggiornamento di una classifica basata sul punteggio del giocatore

Requisiti non funzionali

- Fluidità del movimento degli oggetti
- Grafica user-friendly

1.2 Analisi e modello del dominio

Il gioco è strutturato a livelli di difficoltà crescente, ognuno dei quali presenta una diversa disposizione dei mattonicini appartenenti a tre diverse tipologie. Ogni livello è composto da tre round dove vi è sempre un numero crescente di mattoncini. Il livello contiene anche informazioni riguardanti il punteggio, il quale incrementa man mano che i mattoncini vengono distrutti, e la vita del giocatore.
Le vite stabiliscono il numero di tentativi che l'utente ha per superare il livello al termine delle quali il giocatore potrà segliere se salvare il proprio punteggio, tornare alla schermata di gioco oppure uscire dall'applicazione; le stesse scelte saranno disponibili anche in caso di vittoria.
Il giocatore ha a disposizione un pad che si può muovere solo orrizzontalmente con il quale potrà far rimbalzare la sfera contro i mattoncini. Le tre tipologie di mattoncini sono:

- NormalBrick: mattoncini normali che se colpiti una volta dalla palla vengono distrutti.
- HardBrick: mattoncini che devono essere colpiti due volte per essere distrutti.
- Obstacle: mattoncini indistruttibili.

Ulteriori dettagli circa le entità presenti possono essere consultati attraverso la Figura 1.1 inserita di seguito.

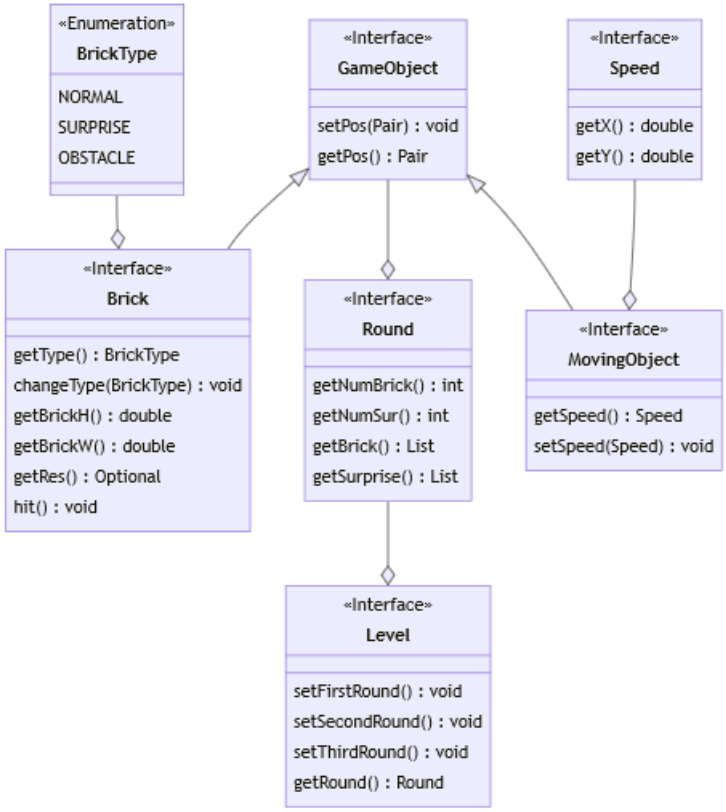

Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro.

Capitolo 2

Design

2.1 Architettura

Si è optato per un pattern architetturale ispirato a un MVC (Model , View, Controller) in cui ci siamo impegnati il più possibile a mantenere separati gli aspetti logici, di controllo e di grafica. Tuttavia non ne rispetta tutti i principi poiché nella classe GameEngine ci siamo serviti di una funzionalità di una libreria grafica (SwingWorker).
La nostra strategia ha comunque il vantaggio che qualora fosse necessario cambiare l'interfaccia grafica questo non causerebbe modifiche nelle classi di Model e Controller.

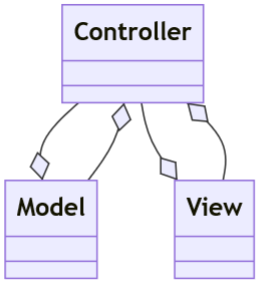

Figura 2.1: Schema UML del pattern.

Il pattern prevede comunque la suddivisione dei compiti in 3 parti:

- Model gestisce i dati, la logica e le regole dell'applicazione.
- View responsabile della visualizzazione del dominio applicativo e dell'interazione con l'utente.

- Controller è l'elemento che si interpone tra il Model e la View.

Il Model è composto da varie classi e interfaccie che rappresentano i principali elementi applicativi (Level, Round, MovingObject, Brick, Pad, Surprise, SizeCalculation). Di seguito nella Figura 2.2 seguente si può vedere una rappresentazione generale del Model.

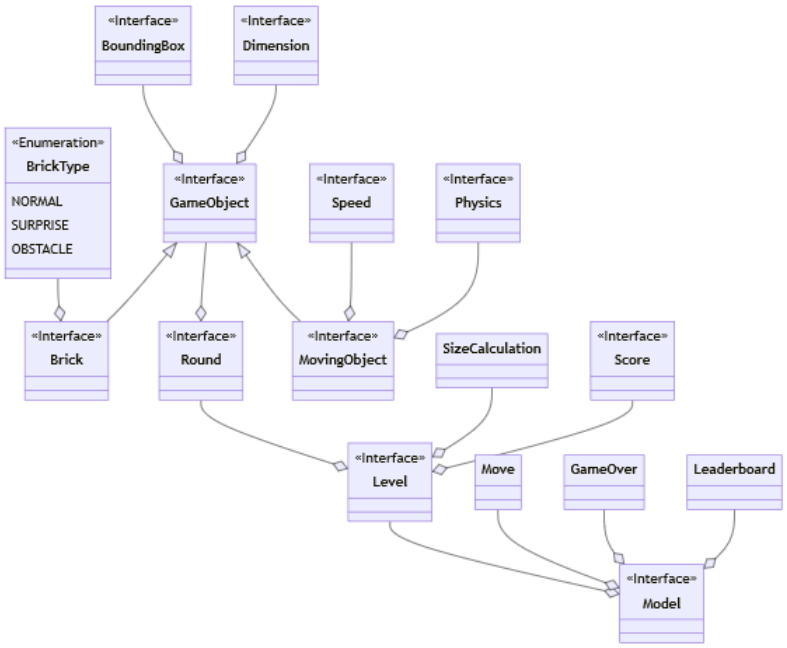

Figura 2.2: Schema UML del Model.

La View è gestita principalmente da un'interfaccia, UIController, che dirige tutti i vari JPanel (LeaderboardView, GameView, CommandsView, StartMenu) ed è colei che comunica con il Controller dell'applicazione. Vi è anche una classe astratta (AbstractView) che viene estesa dai vari menù di gioco (GameOver, PauseMenu e Victory) i quali utilizzano la classe CustomBtn creata con lo scopo di avere i bottoni nei vari menù tutti con le stesse caratteristiche. Le principali dipendenze possono essere viste di seguito nella figura Figura 2.3.

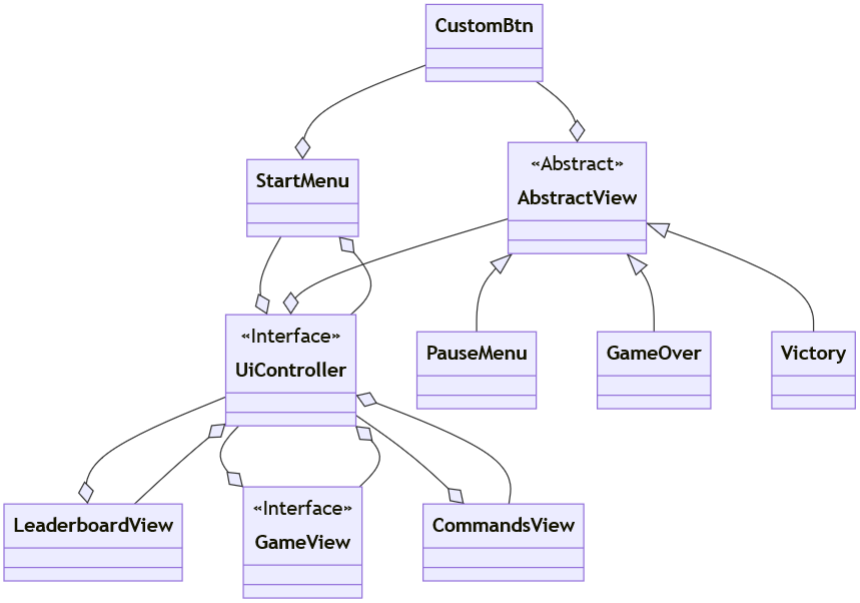

Figura 2.3: Schema UML della View.

Il Controller è costituito da una sola interfaccia che comunica con il Model e con la View ma anche con la classe GameEngine, il motore dell'applicazione. Di seguito nella Figura 2.4 l'UML del Controller.

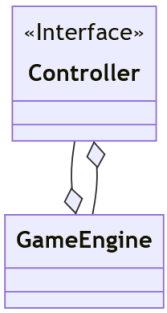

Figura 2.4: Schema UML della View.

2.2 Desing dettagliato

2.2.1 Ruggeri Simone

Le parti di lavoro da me svolte hanno riguardato: la progettazione della struttura dei vari livelli, l'impostazione del primo livello con i suoi 3 round, il controllo se il giocatore manca la pallina oppure se finisce il round, lo sviluppo di alcuni metodi per i bonus, la parte di grafica che riguarda la pausa, il game over e la vittoria.

Livelli
Il gioco è suddiviso in tre livelli. Per la progettazione di una struttura comune per i livelli, ho creato un'interfaccia 'Level' e un'abstract class 'AbstractLevel'. In Level vengono dichiarati i metodi che ogni livello dovrà possedere, mentre in AbstractLevel vengono implemetati. I metodi che però definiscono caratteristiche differenti per ogni round del livello, sono dichiarati abstract e dovranno essere implementati nelle classi dei livelli. I tre livelli dunque, 'FirstLevel','SecondLevel' e 'ThirdLevel', dovranno estendere la classe astratta AbstractLevel. Level e AbstractLevel sono state create fin da subito, proprio per evitare di bloccare l'avanzamento dello sviluppo complessivo del progetto. Inizialmente sono stati creati e implementati i metodi che avrebbero svolto le azioni principali, mentre successivamente a man mano che il progetto progrediva sono stati aggiunti metodi e variabili che si sono dimostrati necessari. Ogni livello è composto da tre round. Per quanto riguarda la realizzazione del primo livello, questi 3 round sono generati tramite delle costanti, che indicano il numero di blocchi normali e blocchi sorpresa, da passare come argomento a 'RoundEasy'. RoundEasy definisce come questi blocchi debbano essere posizionati nella schermata di gioco.

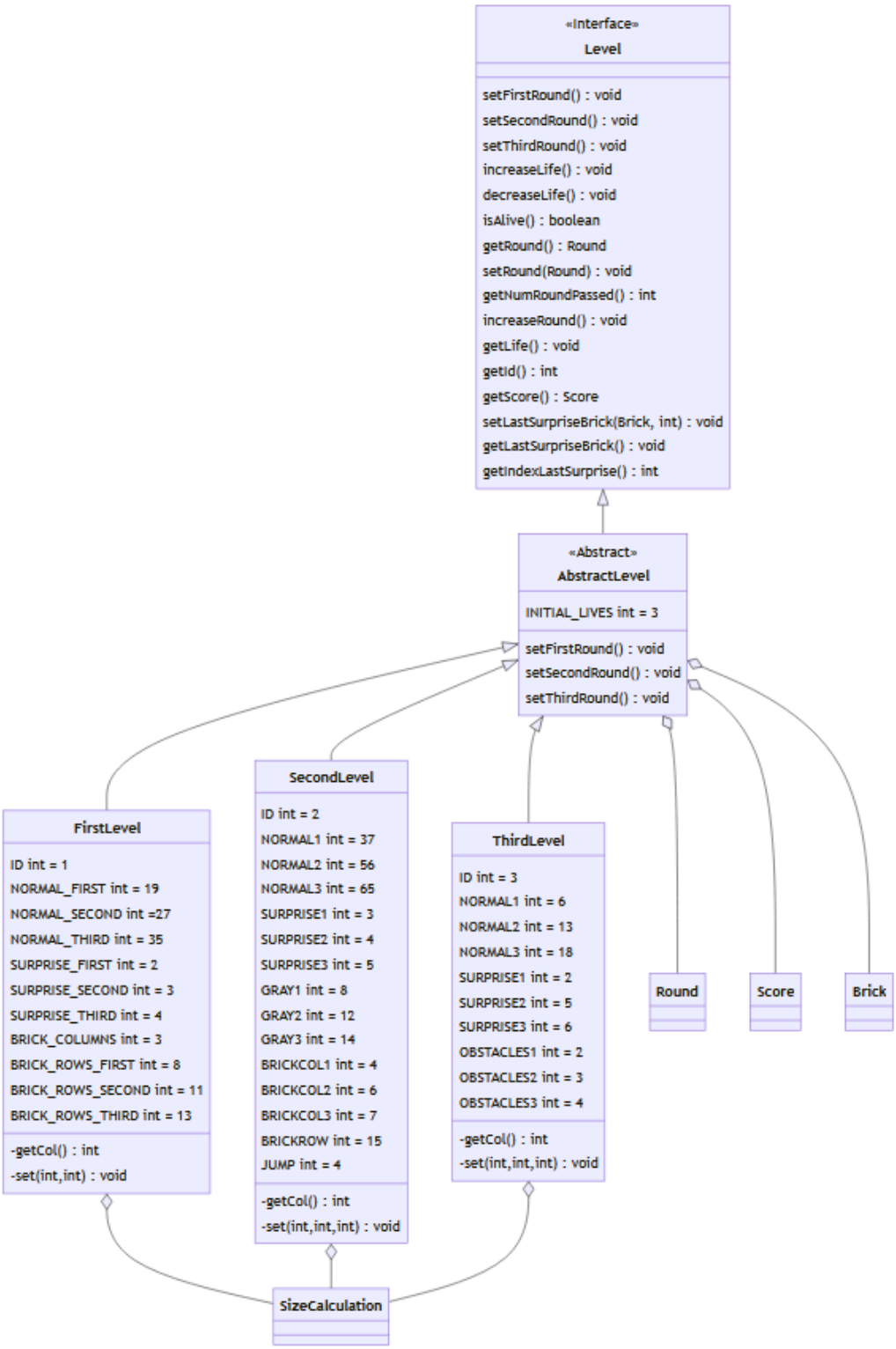

Figura 2.5: Schema UML dell'organizzazione dei livelli.

Controlli sul Round
Il decremento della vita all'interno del round di gioco e il passaggio all'eventuale round successivo sono condizionati dai controlli fatti nella classe GameOver (del model). All'interno di questa classe sono presenti due metodi che restituiscono al Model un valore booleano, che indicano se il player, tramite il controllo del pad, ha mancato la pallina oppure se il round è terminato.

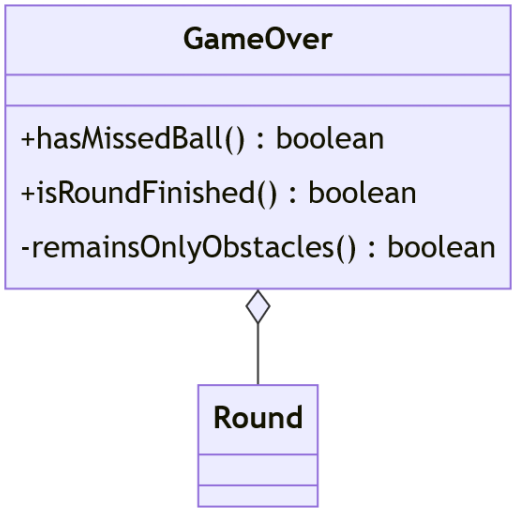

Figura 2.6: Schema UML della classe GameOver.

View di Pausa, Game Over e Vittoria
La parte grafica del progetto prevede l'alternarsi di JPanels (che rappresentano menù inziale, menù di pausa, gameover, vittoria, classifica) con la game view vera e propria. Per la realizzazione della parte grafica di mia responsabilità (menù di pausa, game-over, vittoria), ho creato una classe astratta 'AbstractView' che definsce delle impostazioni generali a cui fanno riferimento le classi che la estendono. Queste caratteristiche prevedono uno sfondo nero, un titolo, e tre pulsanti. Due di questi pulsanti, menuBtn e quitBtn, sono già definiti all'interno di AbstractView poichè sono in comune con tutti e tre i pannelli da visualizzare. Il terzo pulsante viene impostato in AbstractView (a livello di posizionamento) ma definito nelle classi che estendono poichè questo pulsante e la relativa azione che svolge cambia in base a quale panel l'untente si interfaccia nell'avanzamento del gioco. L'istanza UIController permette ai pannelli di poter svolgere azioni come cambiare view (da tutti e tre i panels è possibile tramite il pulsante mentuBtn andare alla schermata iniziale di gioco).

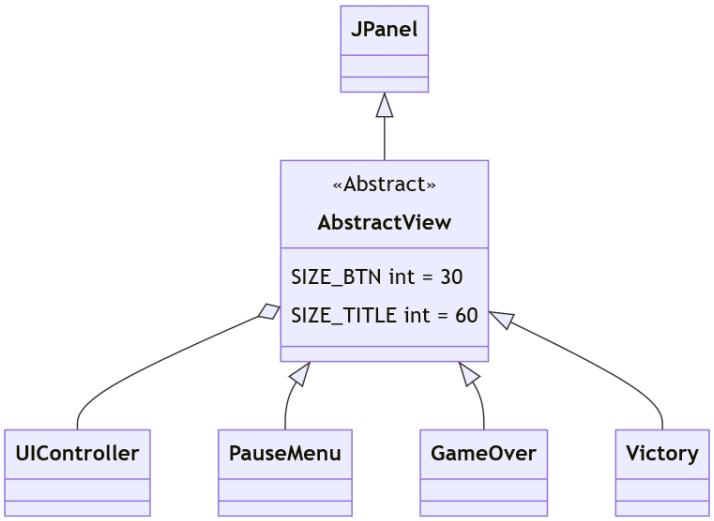

Figura 2.7: Schema UML dell'implementazione della View di Pausa, Game Over e Vittoria.

2.2.2 Desiderio Edoardo

Dal momento in cui si è formato il gruppo il mio primo obbiettivo è stato quello preoccuparmi di creare una buona impostazione, in particolare grazie al colloquio con il professor Pianini ho potuto chiarire al gruppo in maniera definitiva come fosse meglio lavorare sul repository del progetto e come doveva funzionare il pattern dell'applicazione da noi scelto. In oltre, ho studiato come funzionano i file json di configurazione di vscode in maniera da fornire una configurazione user friendly a tutti i miei colleghi per un' esperienza "out of the box" dell' IDE come la configurazione di un debugger meno invasivo rispetto quello di default. Ho riservato una buona attenzione a questa configurazione per limitare al massimo lo stress causato dalla poca intuittività di alcune automattizzazioni ed essere sereni e tranquilli.

Elementi di gioco.

Dalla descrizione del dominio risulta chiaro che il gioco dovrà mostrare e gestire due tipi di entità diverse:

- entità statiche
- entità dinamiche

L'idea iniziale é stata di creare due intefacce che avessero proprietá specifice per ogni tipo di queste entità. Da specificare che un'entitá mobile del gioco estende anche tutte le proprieta di un entitá statica. Proprietá di un GameObj:

- dimensione
- posizione
- riferimento al suo bordo

Proprietá di un MovingObj:

- fisica
- velocitá

con questa struttura iniziale ogni classe estendeva l'implementazione di un gameObj e per gli oggetti che avrebbero dovuto muoversi veniva implementata l'interfaccia MovingObj

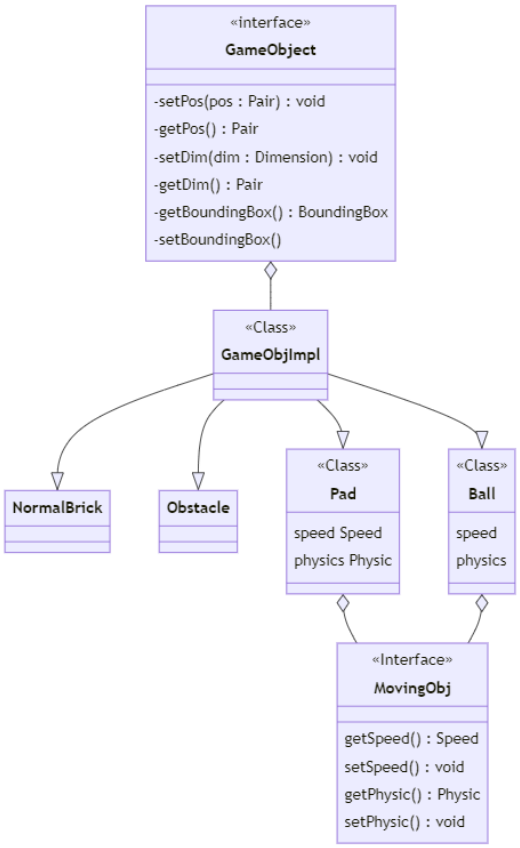

Figura 2.8: Schema UML del desing iniziale delle entitá di gioco.

questo design delle classi si é presto dimostrato fallace, per via della poca manutentibilità ed estenbilitá[1], quindi é stata necessaria una riorganizzazione strutturale della gestione delle entitá di gioco.

riorganizzazione Partendo dall'interfaccia gameObject ho lasciato a chi dovesse occuparsene la possibilitá di implementarla a piacimento. Per quanto riguarda gli oggetti mobili ho definito una classe astratta che permettesse la gestione di tutte le parti comuni a un qualsiasi oggetto di gioco piú la definizione della sua velocitá e eventuale logica di spostamento. In questo modo garantisco una maggiore estendibilitá e riuso del codice (la possibilitá di dover aggiungere un nuovo oggetto mobile non imparanoia nessuno).

[1]risultava dover fare tantissimo copia-incolla per le implementazioni

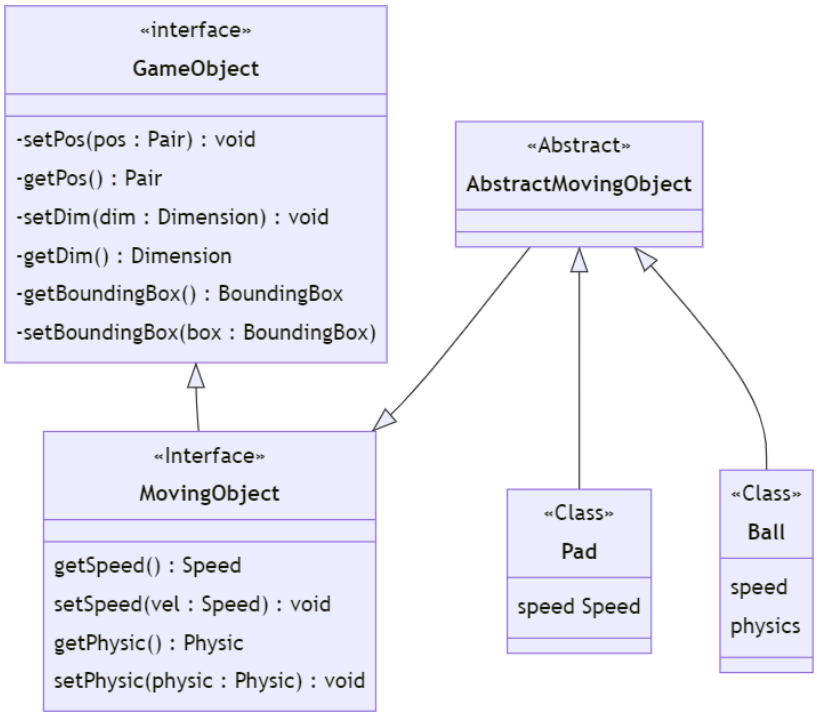

Figura 2.9: Schema UML ristrutturato.

si puó notare che il pad é differemte dalla pallina poiché il suo spostamento viene gestito dall'utente e non dall' implementazione di logiche interne all'applicazione.

implementazione e movimento del pad

Oltre al desing delle entità di gioco ho implementato la logica del movimento del pad, ritengo questa una parte molto delicata poiché la facilitá con cui si puó violare il pattern MVC era abbastanza elevata[2]. Di fatto mi sono occupato di far veicolare l'imput catturato dalla view cercando di scindere il piú possibile la logica di spostamento dall'attuazione dello stesso delegandola al model. Gli step che compie il messaggio di imput per muovere il pad sono i seguenti:

- l'utente preme una direzione che vuole assegnare al pad

[2]potrebbe sembrare molto piú facile e banale inserire i calcoli dentro il jpannel

- il view Controller si accorge di questo e comunica la cosa al controller specificando solo in maniera logica la nuova direzione da intraprendere ma senza preoccuparsi di implementarla
- il controller conosce i vari tipi di direzioni che un GameObj puó assumere e in base alla logica catturata dalla view la concretizza passandone il messaggio all'entryPoint del model
- il messaggio é entrato nel model e viene gestito aggiornando la poizione del pad in base alla sua velocitá

per evitare che il pad uscisse dalla scena di gioco infine ho valutato se la nuova posizione genrasse una collisone coi le deathline del mondo imposte dal model e nel caso accadesse questo evitavo di aggiornare la posizione richiesta ma mantenedo quella in collisione con il bordo. Ecco una vista porzionata rispetto allo scambio di messaggi che vengono fatti quando si vuole muovere il pad

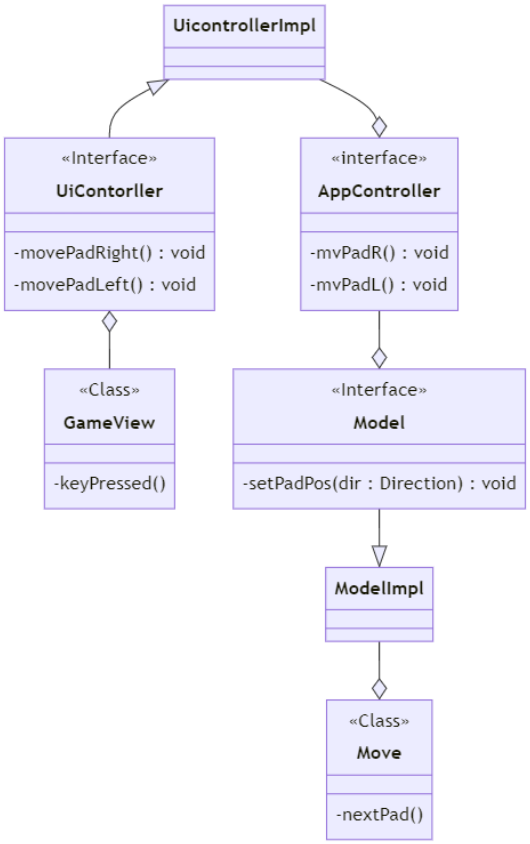

Figura 2.10: da leggere da sinistra verso destra

UIController, la scelta.

ho ritenuto vantaggioso avere un controller dedicato per la vista che comunicasse con il controller principale dell' applicazione.

Pagando con una maggiore ripetizione dei metodi "messaggeri" che si occupano semplicemente di trasportare informazioni dall' applicazione alla sua vista si ottiene una maggiore manutentibilità e scalabilità dell'applicazione. Di fatto si è verificato che modifice come:

- modifica alle classi delle view
- aggiunta metodi o relativa modifica
- aggiunta di intere view

si sono rilevate poco costose sia in termini logici, guardando l'interfaccia UIController è lampante come muoversi, che in termini pratici, per aggiungere una nuova vista basta aggiungere la sua dichiarazione e il metodo che la invochi.

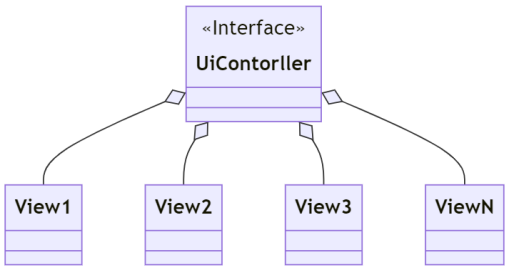

Figura 2.11: proiezione UML della comunicazione fra le viste con UIController

UIController, l'implementazione.

Per gestire le varie viste ho scelto di utilizzare un layout di jswing (ne approffondiró l'utilizzo nelle note di sviluppo). Questo layout è funzionale a quello

che intendevo far fare al View-controller poichè gestisce due o più componenti che condividono lo stesso spazio di visualizzazione (il jframe nel nostro caso).

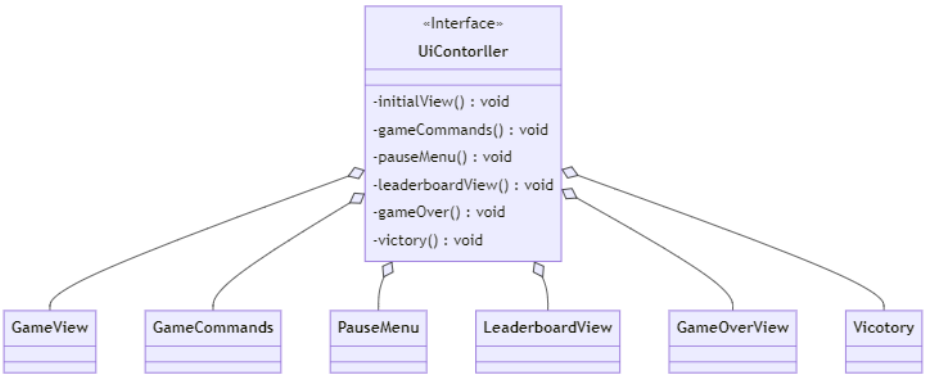

Figura 2.12: proiezione UML dei metodi predisposti al cambio interfaccia

Come sicuramente descritto anche dai miei colleghi, ogni classe View ha un riferimento all'interfaccia del loro controller, di fatto il controller quindi "osserva" il comportamento della view e quando accade una determinata interazione della stessa si scatenerà l'evento desiderato. Per questo tipo di desing mi sono ispirato alle soluzioni viste in laboratorio e come mi è stato detto a ricevimento questo modo di fare non è un pattern observer puro ma posso affermare che sia di simile comportamento.

Window resizable.

l'idea nasce con la logica di mantenere sempre fisse le dimensioni del model e di renderizzare la view in maniera tale da crearne semplicemente le loro proiezioni su schermo.

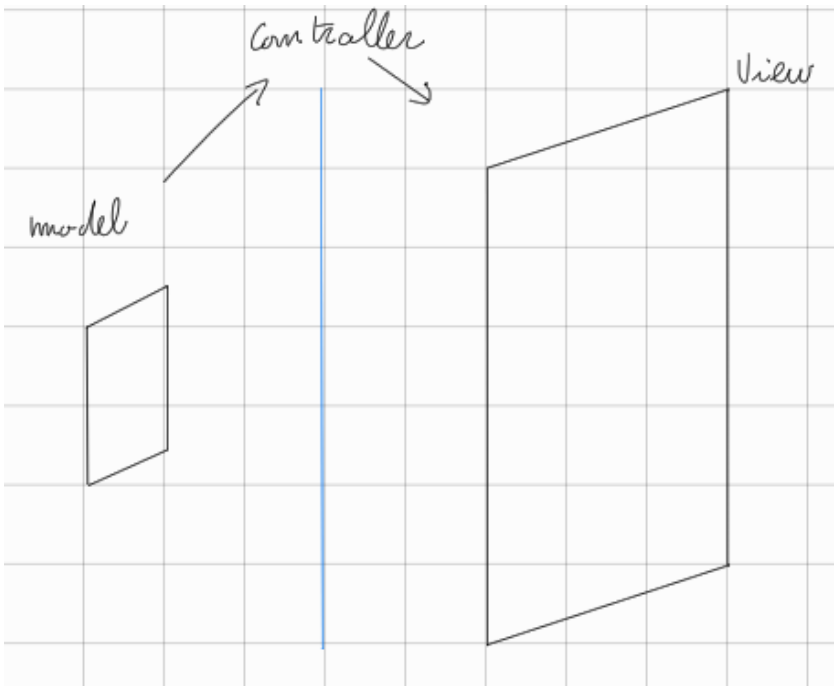

Figura 2.13: bozza della proiezione di un mattoncino fatta durante lo studio

l'implementazione segue la logica descritta in figura: il model passa gli oggetti da disegnare con le relative dimensioni e posizioni sul paino, é compito poi del controller filtrarle e restituirle alla view, ricarlcolate in maniera opportuna.

Window resizable, la proporzione

per proiettare correttamente gli oggettti di dimensione del model il controller deve moltiplicare ognuno di questi per un delta di proiezione.

Questo Delta é calcolato facendo il rapporto fra quelle che sono le dimensioni correnti della View , rispetto a quelle del model.

esempio:

assumiamo ipoteticamente che le dimensioni del model siano un mondo 400x400

nel quale sia disposto un mattoncino con coordinate (2,3) e lunghezza e altezza (5,2). A run time la view ha dimensioni 800x800 e quindi il controller posizionerá il mattoncino seguendo la logica sopra descritta.

- calcolo $\Delta W \rightarrow 800/400 = 2$
- calcolo $\Delta H \rightarrow 800/400 = 2$
- calcolo coordinate $\rightarrow P(2 \times \Delta W, 3 \times \Delta H)$
- calcolo dimensioni $\rightarrow Dim(5 \times \Delta W, 2 \times \Delta H)$

in questa maniera viene assicurato un giusto mantenimento delle proporzioni delgi spazi decisi nel model. Utilizzando questo tipo di proiezione assicuro anche il fatto che qualsiasi evento generato nel model, come una collisione fra oggetti, venga riprodotto fedelmente sulla view.

Surprise: remove Random Brick

questo metodo posso considerarlo come un Malus poiché ho deciso di rimuovere brick senza aggiornare il punteggio, di fatto per il giocatore si ridurrá la possibilitá di aumentare il suo punteggio nel determinato round, é un metodo molto semplice di cui l'implemetazione é praticamente descritta dal suo nominativo.

Surprise: reduce Size & enlarge size pad

questi sono due metodi su cui posso spendere qualche parola piú interssante. Per mantere una buona predisposizione alle modifiche ho creato un metodo di Utiliy che in base a una determinato valore percentuale calcoli un fattore da moltiplicare per la lunghezza a runtime del pad. Il fattore moltiplicativo é calcolato nel seguente modo: **(100-PercentualeDecisa)/100** in questa maniera potrei evolvere l'applicazione aumentando o diminuendo di percentuali variabili il pad generando cosí un gioco piú accattivante dato dall'imprevedibilitá.

Per garantite che il pad rimanga sempre all'interno della scena di gioco nel caso in cui ingrandendolo uscisse dalla stessa ho previsto anche un ricalcolo della sua posizione in base alle nuove dimensioni.
that's it

2.2.3 Castiglioni Chiara

Nel corso dello sviluppo del progetto mi sono occupata principalmente della parte riguardante la struttura dei vari round, il game loop del videogioco, la parte di calcoli per il posizionamento delle varie entità e della loro diminsione, lo sviluppo di due metodi per i bonus, la creazione dell'entità Brick mentre nella parte della View l'apparizione della scritta che indica il bonus preso.
Inoltre ho collaborato con i miei compagni nell'implementazione delle parti comuni come il Controller oppure il Model.
Gestione del game loop
Il game loop è il vero e proprio motore del gioco cioè un ciclo che viene attivato quando:

- L'utente seleziona uno dei tre livello messi a disposizione nello Start-Menu.
- Si riprende il gioco dal menù di pausa.
- Si inizia un nuovo round.

mentre invece viene interrotto nel momento in cui:

- Viene aperto il menù di pausa all'interno della partita.
- Il giocatore perde le sue tre vite messe a disposizione.
- L'utente vince il round e deve passare a quello successivo.
- Il giocatore vince la partita.

Per la realizzazione del game loop mi sono servita di una classe della libreria Swing cioè SwingWorker che consente di eseguire un thread in background in quanto Swing utilizza un singolo thread cioè Event dispatch thread.
All'interno del game loop richiamo alcuni metodi privati che richiamano a loro volta metodi del Controller in quanto questa classe deve comunicare con il Model e con la View aggiornandoli continuamente.
Per quanto riguarda l'aggiornamento del Model il game loop si occupa continuamente di:

- Richiamare l'update delle entità in movimento.
- Controllare se vi è la terminazione del round e in tal caso richiamare l'inizializzazione del round successivo se vi è, altrimenti, viene impostata la vittoria.

- Controllare il meccanismo di perdita di vita dell'utente, nel caso avesse ancora vite viene riposizionata la pallina alla posizione iniziale altrimenti viene dichiarato game over.

In riferimento alla View il game loop si occupa di richiamare la repaint del Frame.
All'interno della classe ho voluto anche implementare un metodo che viene richiamato come ultimo all'interno del ciclo che si occupa di mantenere la stessa velocità di esecuzione del ciclo su diversi PC in modo tale da evitare che su dispositivi molto prestanti l'applicazione venga eseguita troppo velocemente.

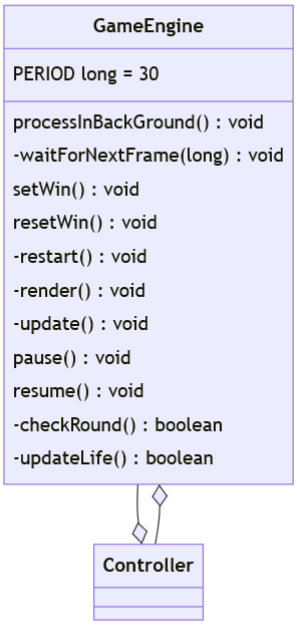

Figura 2.14: Schema UML del GameEngine.

Entità Brick

Fin da subito mi sono occupata dell'entità brick facendo un'interfaccia che estende quella già implementata dai miei colleghi per gli oggetti di gioco in modo tale da avere già i metodi per la posizione dei brick, la loro boundingBox e la loro dimensione.
Per questa entità ho poi implementato una classe astratta che implementa

l'interfaccia in modo tale che le due diverse tipologie di blocchi (normali e ostacoli) potessero estenderla ereditando dei medoti già implementati come per esempio il tipo di blocco e tutti i metodi dell'interfaccia scritta dai miei colleghi sugli oggetti, al fine di poter un giorno estendere i tipi di blocchi, mentre altri, invece, da implementare in quanto differiscono dalle diverse tipologie di blocchi. In particolare mi sono occupata dell'entità di brick normale, che estende la classe astratta, la quale si occupa delle creazione di blocchi normali oppure hard a seconda della resistenza datagli.
Questa scelta strutturale è dovuta dal fatto che i brick normali e quelli hard differiscono solamente a livello della quantità di resistenza quindi, ho ritenuto inutile creare un'altra classe e aggiungere un tipo diverso di brick.

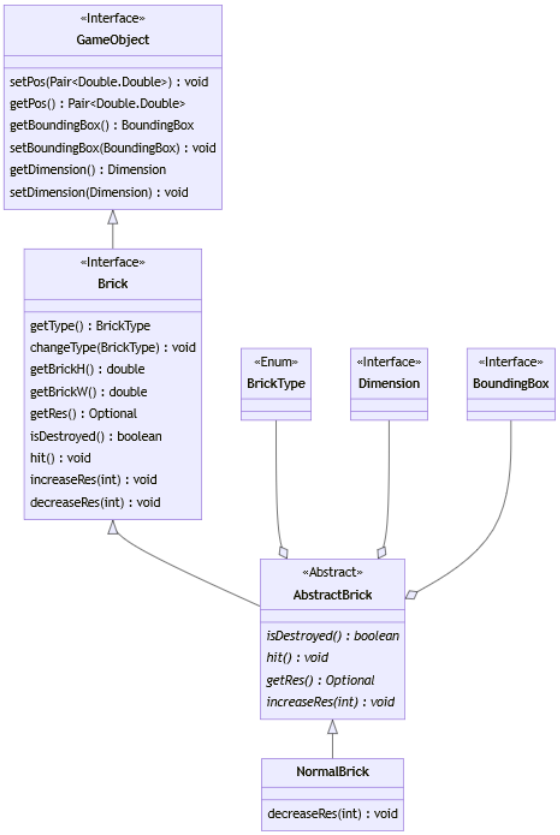

Figura 2.15: Schema UML dell'entità brick.

Secondo Livello

In questa classe mi sono occupata principalmente della scelta sulla quantità di blocchi da disporre all'interno dei tre round. La struttura della mia classe può essere visualizzata alla Figura 2.5.

SizeCalculation
Questa classe si occupa di eseguire calcoli riguardanti il posizionamento delle varie entità e la loro dimensione all'interno del mondo di gioco al quale abbiamo deciso di dargli una dimensione fissa.
L'idea che mi era venuta in mente era quella di calcolare le dimensioni del brick in base al numero di brick da posizionare, mentre di lasciare mezza lunghezza di brick dai rispettivi lati del mondo per calcolare il punto di partenza e di fine del posizionamento dei blocchi lungo l'asse delle ascisse. Per il calcolo dell'ordinata invece viene calcolata in base al numero di brick per colonna, questo perché ogni round ha sempre più blocchi quindi necessita di spazio maggiore.
Vi sono anche due metodi che calcolano le dimensioni del pad e della pallina rispetto alle dimensioni del mondo nel model e un metodo che serve alla view per sapere qual è la coordinata y di ogni riga in modo da riuscire a colorare in modo diverso ogni riga.

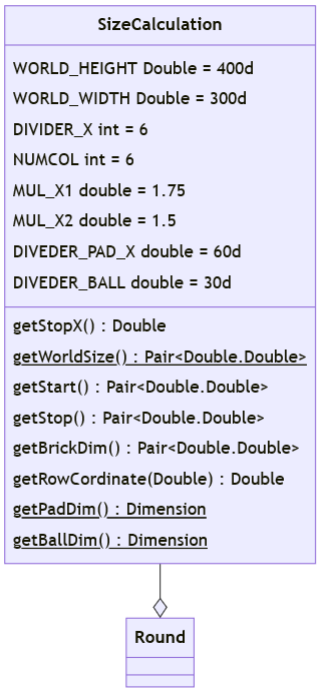

Figura 2.16: Schema UML di SizeCalculation.

Round
Per quanto riguarda l'organizzazione dei vari round è stato utile fin da subito creare una classe astratta in quanto ha permesso ai miei colleghi di estenderla

e creare il proprio round e permetterà in futuro di aggiungerne altri.
All'interno di questa classe ho scelto di implementare i metodi utili per rimuovere i brick, per settare i brick surprise in modo randomico, in quanto comune a tutti i tipi di round, il metodo per aggiungere le palline surprise e altri metodi get e set.
L'unico metodo che ho deciso di mettere abstract è quello per il posizionamento dei brick in quanto ogni livello ha un proprio round con una diversa disposizione.

Round Medium

Il round al quale mi sono dedicata io stessa è quello medio dove la disposizione dei brick è composta da tre colonne separate ognuna con quattro blocchi ciascuna e vi è una tipologia di blocco in più rispetto agli altri cioè quello hard. La differenza tra i tre round all'interno del mio livello è che il numero di righe e il numero di brick hard e brick surprise aumentano.
All'interno di questa classe ovviamente ho implementato il metodo che era definito astratto dove all'interno richiamo anche i metodi per posizionare i blocchi surprise e quelli hard in modo randomico. Per posizionare i brick ovviamente mi sono servita della classe SizeCalculation già descritta per sapere dove partire a inserirli.

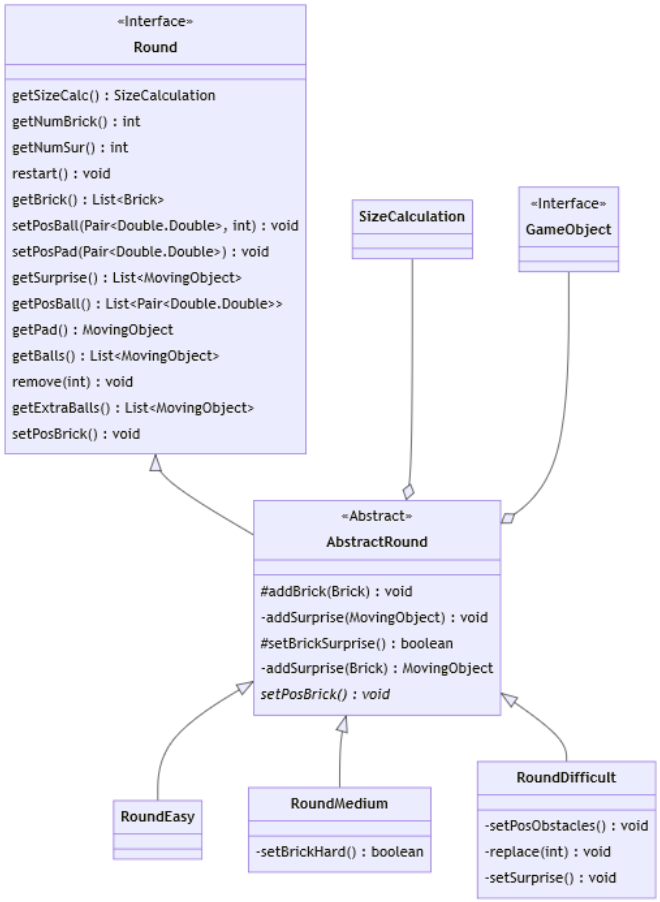

Figura 2.17: Schema UML della struttura Round.

Bonus

In questo progetto mi sono anche dedicata a creare due bonus che fossero correlati con quello che ho spiegato fino ad ora, cioè i blocchi hard.
Infatti i miei due surprise sono uno un bonus in quanto cambia per 10 secondi i blocchi hard con quelli normali mentre l'altro è un malus perché aggiungo una riga di blocchi hard. Per l'implementazione del primo mi sono servita di due classi di libreria Timer e TimerTask, sovvrascrivendo il metodo run che è colui che scaduto il Timer esegue le istruzioni che ha al suo interno.

View

La parte da me svolta all'interno della view è stata quella principalmente di: implementare il metodo per la colorazione dei brick a seconda della riga in cui sono situati utilizzando il metodo implementato all'interno di SizeCalculation e l'apparizione della scritta del bonus preso implementato attraverso l'uso di un Timer come fatto già all'interno di un mio bonus.

2.2.4 Foschi Virginia

Nel corso dello sviluppo del progetto il mio lavoro si è principalmente concentrato sulla gestione e implementazione della classifica di gioco, del salvataggio del punteggio, del livello difficile con i corrispondenti round e della visualizzazione dei comandi principali del gioco.
Ho collaborato anche alla creazione dei bonus e alla visualizzazione degli oggetti del gioco.

Classifica di gioco
Per quanto riguarda la realizzazione della classifica ho optato per la creazione, all'avvio dell'applicazione, di un file di testo nella home directory dell'utente in cui verrranno inizialmente caricati i punteggi conseguiti da me e i miei compagni che si trovano memorizzati all'interno di un file di testo nella cartella resources del progetto.
Ogni giocatore ha la possibilità, una volta completato il livello o dopo aver perso, di memorizzare il punteggio complessivo conseguito inserendo nome e password. Si è pensato infatti all'utilizzo di una password in modo da permettere al giocatore, una volta riaperta l'applicazione, di aggiornare il punteggio conseguito nelle partite precedenti.
La memorizzazione del punteggio avviene nel seguente modo:

- Ogni volta che il giocatore decide di salvare il punteggio conseguito in un determinato livello, il vecchio punteggio, relativo a quel livello, sarà sempre rimpiazzato dal nuovo.

- Il punteggio complessivo, cioè quello visualizzato nella classifica, sarà dato dalla somma dei punteggi conseguiti nei 3 livelli.

La classifica aggiornata è in ogni momento visualizzabile cliccando l'apposito tasto presente nel menù di gioco.

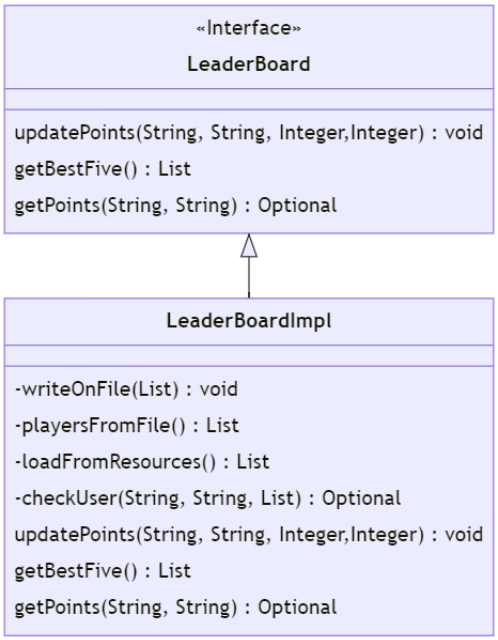

Figura 2.18: Schema UML della leaderboard.

Livello e round difficile

Per quanto riguarda invece la struttura dei livelli e dei round si è cercato di organizzare il tutto in modo tale da creare codice facilmente espandibile in caso di aggiunta di ulteriori livelli di gioco e round.
Ogni livello mantiene informazione sul round corrente al cui interno sono contenute tutte le informazioni riguardanti gli oggetti in gioco con le corrispondenti caratteristiche (posizione, velocità, dimensione,...). In particolare il livello difficile è caratterizzato dalla presenza di ostacoli, cioè mattoncini indistruttibili, e dalla disposizione dei blocchi in maniera piramidale.
Mi sono basata sulla strutturazione e organizzazione delle classi realizzata dai miei compagni, che è possibile osservare in maniera dettagliata alla Figura 2.5 e Figura 2.17, e mi sono occupata del collegamento tra le varie classi in modo da permettere la visualizzazione a video degli oggetti del gioco tramite la classe GameView.

Gestione palline bonus

Mi sono occupata inoltre della gestione delle palline bonus ovvero oggetti che vengono rilasciati nel momento in cui viene distrutto un blocco sorpresa. Ciascuna pallina corrisponde ad un bonus o malus che viene scelto ogni volta in modo casuale e di cui il giocatore potrà usufruire solo in caso di contatto

tra la pallina e il pad. In particolare mi sono occupata dell'implementazione di 2 bonus ovvero diminuzione velocità pallina e rimpiazzo ostacoli con blocchi normali e un malus corrispondente all'aumento della velocità della pallina.

View
Dal punto di vista della view mi sono occupata principalmente dell'implementazione di 3 interfacce grafiche:

- LeaderBoardView: interfaccia che permette di visualizzare a video la classifica costituita dai migliori 5 giocatori.
- CommandView: insieme ai miei compagni abbiamo pensato di realizzare un'interfaccia per permettere al giocatore di visualizzare i comandi principali del gioco.
- SaveScore: JDialog che permette al giocatore di inserire i suoi dati (nome e password) e salvare il proprio punteggio.

2.2.5 Balzoni Margherita

Il mio contributo all'interno del progetto ha riguardato principalmente la gestione della fisica della pallina, in particolare gli aspetti di creazione delle BoundigBox, controllo delle collisioni, movimento e gestione della direzione della palla.
Mi sono occupata poi di realizzare il menù iniziale di gioco e implementare la classe che memorizza e increamenta il punteggio durante la partita.
Ho inoltre collaborato alla realizzazione di alcuni bouns, nello specifico l'aggiunta di una pallina extra e il raddoppio del punteggio per un determinato periodo di tempo.

BoundigBox
Per delimitare l'area occupata dai vari oggetti di gioco ho utilizzato una Bounding Box, ovvero il più piccolo rettangolo capace di contenere una determinta figura al suo interno.
Sono partita realizzando un'intefaccia BoundigBox la quale viene implementata da una classe astratta AbstractBoundingBox. Successivamente ho creato due classi, CircleBoundingBox e RectBoundingBox, che estendono la classe astratta. In particolare la prima permette la creazione di Bounding Box per gli oggetti circolari come le palline e gli oggetti surprise, mentre la seconda le realizza per il pad e i brick.

La box viene creata utilizzando la posizione di ogni oggetto (che fa riferimento all'angolo in alto a sinistra, permettendo così di calcolare i restanti tre) e viene memorizzata in una mappa all'interno della quale si trovano le coordinate di tutti e quattro gli angoli della "scatola".
Il metodo collideWith implementato all'interno della classe astratta AbstractBoundigBox verifica l'intersezione tra due bounding box, se non si intersecano verrà restituito un opzionale vuoto, in caso contrario restituirà il punto in cui è avvenuta la collisione:

- Su uno dei due lati orizzontali della box.
- Su uno dei due lati verticati della box .
- Tra due angoli della box .

Questa informazine ci sarà utile in seguito per verificare le collisioni e gestire il cambio di direzione della palla.

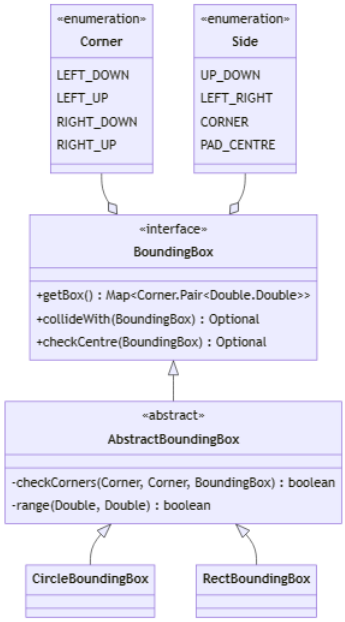

Figura 2.19: Schema UML della BoundigBox.

Gestione della direzione

Per gestire la direzione mi sono serivta di due interfacce Direction e Physics le quali vengono successivamenteimplementate dalle Classi DirectionImpl e BallPhysicsImpl.
La prima si occupa di tener traccia della direzione corrente degli oggetti

mentre la seconda gestisce l'aspetto più dinamico, grazie al metodo changeDirection è infatti possibile cambiare la direzione della palla a seconda del punto di collisione che si è ottenuto.
La pallina può muoversi in cinque direzioni, quattro sulla diagonale e una verso l'alto (che si può ottenere solo colpendo il pad al centro).
Per quanto riguarda le prime quattro ho deciso di gestirle utilizzando una coppia di versori che assumono i segenti valori:

- Direzione SUD = 1
- Direzione NORD = -1
- Direzione EST = 1
- Direzione OVEST = -1

(Dove SUD e NORD si riferiscono all'asse y e EST e OVEST all'asse x)
Per la direzione verticale non ho usato lo stesso approccio per un motivo puramente estetico. Non avendo una componente orizzontale infatti , la pallina subiva visivamente un notevole "decremento" della velocità. Ho decisio così di assegnare alla componente vericale valore -2 in modo da raddoppiare la velocità sull'asse y.

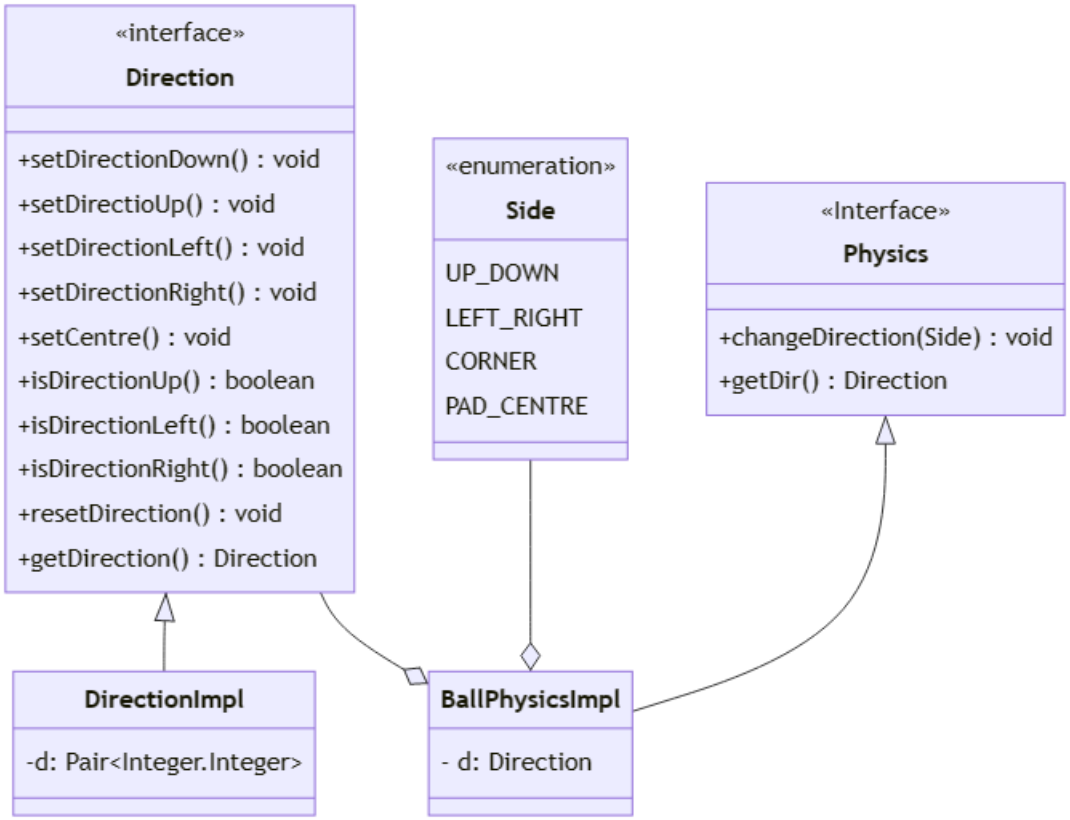

Figura 2.20: Schema UML della gestione della direzione.

Collisioni

Per quanto riguarda le collisioni, io mi sono occupata principalmente di gestire quelle riguardanti la pallina. Infatti sono stati realizzati da me tre dei quattro metodi che si trovano all'interno della classe Collision (collideWithEdges, collideWithBrick, collideWithPad) .

Il controllo delle collsioni l'ho strutturato nel seguende modo:
come prima cosa viene creata una BoundigBox specifica per ogni oggetto di gioco da controllare , essa viene creata ogni volta che viene richiamato il metodo, successivamente avviene la verifica della collisione sfruttando i metodi della classe bounding box (che ho descritto precendentemente), infine se la collisione è avvenuta verrà aggiornata opportunamente la direzione e in caso di distruzione di uno o più brick sarà aggiornato il punteggio.

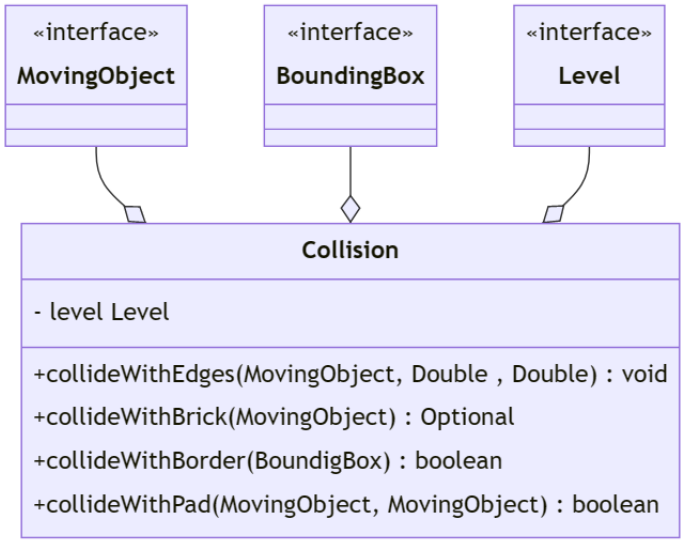

Figura 2.21: Schema UML della gestione delle Collisioni.

Movimento

Il movimento degli oggetti in gioco è gestito dalla classe Move la quale, attraverso il metodo updat aggiorna la posizione di tutti i MovingObject. Per quanto riguarda il movimento delle palline verrà richiamato il metodo nextBall.

All'interno di questo metodo dopo aver controllato se sono avvenute tutte le possibili collisioni della palla ed eventualmente dopo aver modificato la direzione, viene aggiornata la posizione della pallina.

La nuova posizione si ottiene sommando a quella attuale la direzione moltiplicata per la velocità.

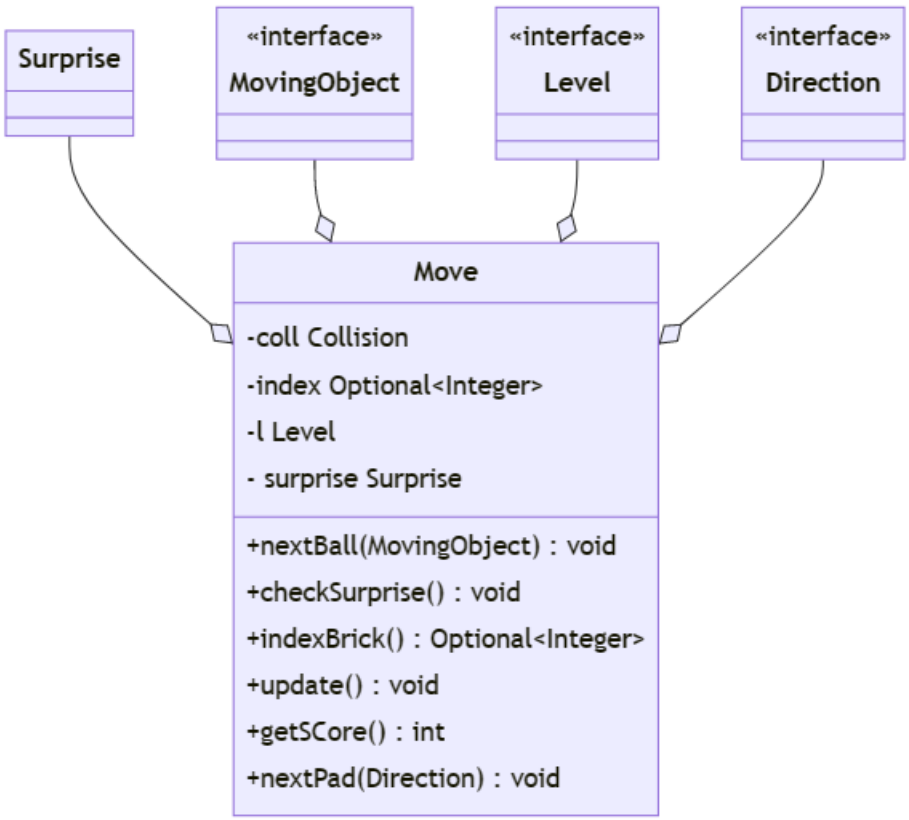

Figura 2.22: Schema UML del Movimento.

Punteggio

Dopo aver discusso insieme il valore da assegnare ad ogni brick distrutto ,abbiamo optato per il seguente metodo:
Ad ogni brick distrutto viene assegnato un punteggio pari ad una costante all'interno della classe Score (che in questo caso ha valore uno), se vengono distrutti più brick consecutivamente questi hanno tutti valore doppio rispetto alla costante, ciò permette di ottenere punteggi ogni volta differenti completando gli stessi livelli.
Inizialmente si era pensato di raddoppiare il punteggio per ogni brick colpito consecutivamente ma ciò postava ad un incremento eccessivo del punteggio al termine del terzo round, poichè il punteggio è unico per un intero livello.
Per realizzare la struttura adatta a gestire il punteggio sono partita realizzado un'interfaccia score contenente metodi implementati dalla classe ScoreImpl che permetono di svolgere azioni utili: Ad esempio resetPoints che segnala di non dover raddoppiare il punteggio per il blocco successivo (ciò avviene quando la pallina colpisce il pad o un ostacolo) o enableBonus.
Questi metodi grazie all'utilizzo dei campi della classe permettono di increamentare il punteggio in modi diversi a seconda della situazione di gioco in cui ci si trova.

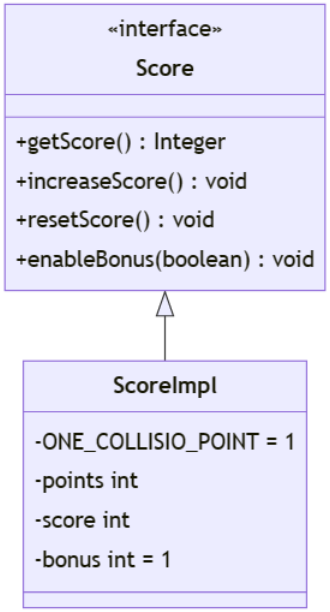

Figura 2.23: Schema UML del Punteggio.

Menù iniziale

Per quanto riguarda la parte grafica, mi sono occupata di realizzare il menù d'avvio dell'applicazione utilizzando la libreria Java swing.
La classe StartMenu estende JPanel e contiene al suo interno la Jlabel con il nome del gioco e un buttonContainer (ovvero un JPanel che ha come layout un GridBagLayout) che contiene e organizza i vari bottoni della schermata.
Per velocizzare la scrittura e evitare inutili ripetizioni ho creato anche una classe CustomBtn che si occupa di creare bottoni personalizzati per tutti i menù del gioco.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In parallelo allo sviluppo del progetto sono stati effettuati anche diversi test automatici utilizzando la libreria JUnit. Sulle componenti principali sono stati eseguiti i seguenti test automatici:

- Test sui bonus: abbiamo scelto di testare ognuno i propri bonus all'interno dei vari livelli.
- Test sulla entità brick: testato la resistenza di un blocco hard e degli ostacoli.
- Test sulle modalità di game over.
- Test sulla classifica: si verifica la correttezza del salvataggio del punteggio del giocatore.
- Test sulla dinamica: sulle collisioni e sul movimento dei MovingObject.

Si è anche dedicato del tempo per fare dei test manuali dedicati principalmente al ridimensionamento della finestra di gioco. Il gioco è stato collaudato su più sistemi operativi:

- Windows 10 e 11.
- Light Ubuntu 22.04.1.
- macOS.

3.2 Metodologia di lavoro

Per lavorare a questo progetto abbiamo cercato di strutturare il lavoro seguendo queste macro domande:

Qual'è la missione del progetto e con quali risorse si intende perseguirla?

La missione del progetto è ampiamente descritta nei paragrafi precedenti. Le risorse da noi scelte in fase di pianificazione e progettazione sono varie tra cui:

- materiale didattico fornito dai professori
- plugin gradle
- versione di java 17
- git e github[1]
- strumenti di pianificazione, comunicazione e in generale multimediali forniti dall'account aziendale (unibo).

Come verrà condotto il progetto?

Si può riassumere la conduzione del progetto con un'organigramma orizzontale. Ognuno ha piena responsabilità dei compiti che ci si è auto affidato al tempo di decisione e assegnazione delle task. Questo implica il fatto che l'auto organizzazione di ogni singolo componente è fondamentale per la buona riuscita del progetto. Ovviamente in caso di difficoltà il gruppo stimolerà il singolo a una soluzione senza però intaccarne la creatività.

Come ne verrà controllato l'avanzamento?

Organizzazione DVCS

Nelle varie risorse scelte sono indicati git e github, git è un software DVCS presentato e spiegato a lezione mentre github.com è un servizio hub di repository. La produzione è organizzata su più branch.
Il branch principale (master nel nostro caso) raccoglie tutto il codice di progetto approvato da tutti, su github è settata una policy di sicurezza affinchè sia obbligatorio aprire una pull request per poter eseguire il merging del codice. Il branch develop raccoglie la linea di sviluppo in fase di pre-relies in

[1] `github.com`

cui è ammissibile trovare eventuali bug o errori di desing.
Ogni elemento del teamWork, ad ogni step evolutivo, si occupa di creare un branch locale che non è necessario venga pubblicato poichè finito il lavoro si occuperà di eseguire il merging in develop.

Ciclo di vita del progetto

Ogni pull request (develop - master) verrà eseguita quando il codice develop soddisfa i requisiti che l'applicazione deve avere rispetto all'obbiettivo incrementale che ci siamo posti. In linea di massima la nostra scelta è di procedere a piccoli passi fino alla finalizzazione del progetto.

La parte di lavoro sviluppata in comune è stata quella inanzitutto della parte di analisi iniziale e poi successivamente durante il corso dello sviluppo sono state implementate classi comuni a tutti come quelle del Controller e quelle del Model, dove ognuno ha inserito dei metodi che servivano per collagare le parti di codice già create singolarmente.
Si riporta di seguito la divisione dei compiti di ogni componente del gruppo.

3.2.1 Castiglioni Chiara

Le attività da me svolte sono state:

- Creazione della struttura dei round e il relativo round medio.
- Gestione del game loop.
- Implementazione della parte di calcoli riguardante la dimensione e posizione delle entità.
- Implementazione di due metodi all'interno della classe Surprise.
- Implementazione del livello medio.
- Creazione dell'entità brick.
- Apparizione della scritta al centro del frame che indica il bonus preso.
- Colorazione delle righe di brick.

3.2.2 Foschi Virginia

I compiti da me sviluppati sono stati:

- Implementazione del livello difficile con i corrispondenti 3 round.
- Gestione classifica e salvataggio punteggio.
- Implementazione di 3 metodi all'interno della classe surprise.
- Visualizzazione degli oggetti (pad, pallina e brick) nella scena.
- Creazione e gestione delle palline bonus.
- Creazione e gestione degli ostacoli.
- Realizzazione di tre interfacce grafiche.
- Creazione e gestione della velocità degli oggetti.

3.2.3 Desiderio Edoardo

i problemi da me risolti principalmente sono:

- Configurazione progetto gradle
- garantire piú possibile l'integritá del pattern architetturale dell'applicazione
- Desing oggetti di gioco e movimento del pad
- Desing e implementazione della gestione delle viste
- studio, desing e implementazione delle figure in maniera che fossero ridimensionabili a runtime
- riprogettazione e suddivisione in pkg del progetto in maniera da poter suddividere le classi in base al loro concetto
- Configurazione e personalizzazione di vscode con formatter javastyle in modo da automatizzare piú possibile il lavoro
- implementazione di 3 metodi dentro it.unibo.game.app.model.Surprise
- gestione di avanzamento della produzione, a metá progetto si é rotta ma grazie alla buona organizzazione descritta nel paragrafo prima siamo riusciti a recuperare la linea di progetto

3.2.4 Balzoni Margherita

I compiti da me svolti sono stati:

- Gestione della direzione della pallina.
- Creazione delle BoundigBox.
- Gestione delle collisioni della pallina con i bordi, il pad e i brick.
- Gestione del movimento della palla.
- Implementazione di due metodi della classe Surprise.
- realizzazione del menù iniziale dell'applicazione.
- Gestione del punteggio.

3.2.5 Ruggeri Simone

Le attività da me svolte sono state:

- Creazione della struttura dei livelli.
- Implementazione del primo livello con i corrispondenti 3 round.
- Realizzazione dei controlli nella classe GameOver.
- Creazione della struttura e implementazione dei menù di pausa, gameover e vittoria.
- Definizione nella classe Surprise della mappa e del metodo che la inzializza.
- Implementazione di due metodi bonus della classe Surprise.

3.3 Note di sviluppo

In questa sezione vengono elencati gli aspetti avanzati del linguaggio utilizzato e di librerie.

3.3.1 Desiderio Edoardo

- Ispirato dalla programmazione funzionale ho cercato di evitare il piú possibile la scrittura di switch statement all'interno del progetto, in questo snippet[2] ho cercato di fare un esempio per tutti i compagni del gruppo. Adottare questo metodo garantisce di mantenere in ordine il codice e averne una manutentibilità e flessibilitá maggiore (la classe Surprise.java con uno switch sarebbe stata incomprensibile a mio avviso)

- utilizzo di CompletableFuture per rendere asicrone operazioni dei surprise in modo che impattino il meno possibile sull'esecuzione dell'intera applicazione

- tutte le collezioni che dovevo ricalcolare nel controller per rendere le figure resizable le ho modificate utilizzando stream combinate con lambda expression, l'utilizzo di lambda é stato fatto anche in assenza di stream

- Per la gestione e lo scambio delle viste all'interno di un unico frame ho utilizzato CardLayout. Come in un mazzo da gioco, ogni carta[3] all'avvio dell' applicazione viene aggiunta al "deck"[4] a questo punto, con i metodi preposti al selezionamento di una carta basterà richiamarli per una visualizzazione corretta. Senza un'accurata fase di debugging non avrei mai capito che CardLayout non sposta in automatico il focus sulla carta corrente. É stato fondamentale capire come aggiornare quest'ultimo poiché ci avrebbe costretto a una soluzione diversa ma sicuramente meno efficace e pulita

- java.util.TimerTask java.util.Timer per realizzare l'attivazione di processi in backround in un determinato asso di tempo

3.3.2 Foschi Virginia

Nelle parti di codice da me curate, ho cercato, dove possibile, di fare uso delle seguenti funzionalità del linguaggio Java:

- uso di stream e lambda expression per rendere il codice più leggibile.

[2] `https://pastecode.io/s/18uxzd1a`
[3] nel nostro caso, classi che estendevano JPanel in modo da personalizzarne i contenuti
[4] il JPanel che riempie il frame principale

- Optional che mi hanno permesso di memorizzare tutti i mattoncini, indipendentemente dal tipo, in un'unica lista all'interno della classe round.
- Iterator per scorrere la lista delle palline bonus e contemporaneamente eliminarle in caso di contatto con il pad o fuoriuscita dal frame.
- java.util.Serializable per memorizzare l'oggetto User nel file.

3.3.3 Castiglioni Chiara

Nelle mia parte di progetto ho cercato di utilizzare il più possibile:

- stream e lambda expression.
- Optional utilizzati per la resistenza dei brick.
- javax.swing.SwingWorker[5] per la realizzazione del game loop.
- java.util.Timer[6] e java.util.TimerTask[7] per l'implementazione di un metodo bonus e per l'apparizione della scritta del bonus preso.

L'unica fonte che ho utilizzato è stata `https://www.youtube.com/watch?v=HUuiGW6TyfI` per comprendere il funzionamento della classe SwingWorker, oltre alla documentazione Oracle.

3.3.4 Ruggeri Simone

Nelle parti di codice di mia responsabilità ho cercato di utilizzare le lambda expression dove possibile.
Per risolvere il problema riscontrato nell'inizializzazione della mappa in Surprise, che usando Map.of non potevo inizializzare più di 10 istanze, ho trovato la soluzione di utilizzare Map.ofEntries online. `https://typeofnan.dev/how-to-create-a-map-of-more-than-10-in-java/`

3.3.5 Balzoni Margherita

Nelle parti di codice da me realizzate ho crecato di utilizzare dove possibile lambda expression, è risultato molto utile inoltre l'uso di Optional, utilizzati in diversi punti del mio codice sopratutto per quanto rigarda il controllo delle

[5] `https://docs.oracle.com/javase/7/docs/api/javax/swing/SwingWorker.html`
[6] `https://docs.oracle.com/javase/7/docs/api/java/util/Timer.html`
[7] `https://docs.oracle.com/javase/7/docs/api/java/util/TimerTask.html`

collsioni.
Per l'implementazione di uno dei due bonus a me assegnati, ho inoltre sfruttato java.util.Timer e java.util.TimerTask, che mi ha permesso di raddoppiare il punteggio per un determinato periodo di tempo.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Desiderio Edoardo

Inizio chiedendo scusa a tutto il gruppo, visto il mio compito gestionale dell'avanzamento mi sento responsabile di qualsiasi errore rilevato dal plugin gradle. Ho chiuso un' occhio accontentandomi di una compilazione andata a buon fine per la fretta di veder maturare il progetto. Non mi sento responsabile per l'errore in se, ma avrei dovuto insistere nel trovare e stimolare tutti a una risoluzione incrementale anche di errori non direttamente connessi al dominio del problema. A parte questa questa mia inadempienza gestionale sono molto felice del contributo dato al progetto e di aver potuto partecipare con questo gruppo. Ho notato durante il lavoro che ogni componente ha messo a disposizione di tutti i suoi punti di forza mentre tutto il gruppo ha spalleggiato il singolo sui punti piú deboli. Nella sfera privata invece l'esperienza mi ha impattato positivamente poiché dover gestire il mio tempo per il progetto ha avuto riflessi positivi anche nelle mie task personali.

4.1.2 Foschi Virginia

Non avendo mai svolto un progetto di gruppo prima d'ora devo dire che mi ritengo abbastanza soddisfatta del lavoro svolto. Lavorare in gruppo non è semplice ma credo che ciascuno di noi abbia contribuito con le proprie capacità alla realizzazione complessiva del videogioco. Fondamentali sono sicuramente stati gli incontri iniziali per stabilire la suddivisione del lavoro e la capacità dei vari membri del gruppo di sapersi ascoltare in modo tale da trarre, dalle varie opinioni, pensieri, idee, una soluzione ottimale all'organizzazione del progetto e alla strautturazione delle varie classi. Nel mio

piccolo penso di essermi data da fare nella realizzazione del progetto e di essermi resa disponibile nell'aiutare gli altri in caso di difficoltà. Lavorare in gruppo mi ha posto dinanzi ad una serie di sfide come il sapersi fidare del lavoro altrui e sapersi confrontare su questioni su cui si hanno idee/pareri discordanti, sfide per nulla banali per un caratterino come il mio, però devo dire che complessivamente sono soddisfatta dello sviluppo finale del gioco e dell'essere riusciti a rispettare le tempistiche.

4.1.3 Castiglioni Chiara

Sono soddisfatta del lavoro da me svolto. Questo proggetto è stato molto utile per colmare alcune lacune riguardo allo sviluppo di un applicativo completamente da zero, dato che non avevo mai avuto occasione. Anche lavorare in gruppo è stata una sfida che mi ritengo di aver superato in quanto siamo riusciti a organizzarci e a comunicare con costanza, senza lasciare indietro nessuno e aiutando chi era in difficoltà spronandolo a trovare una soluzione. Inoltre questo progetto mi ha aiutata a comprendere meglio gli aspetti del linguaggio da me affrontati per la prima volta e mi ha fatto crescre a livello di organizzazione del lavoro nel rispetto delle tempistiche in quanto alcuni compiti da me svolti era propedeutici allo sviluppo di alcune parti affrontate dai miei colleghi.
Per quanto riguarda il futuro, mi piacerebbe rivedere quanto già fatto e migliorarlo; con più tempo a disposizione magari potrei provare a cambiare la libreria grafica e raffinare il codice già scritto.

4.1.4 Ruggeri Simone

Questo progetto ha rappresentato la mia prima occasione per la realizzazione di un progetto di gruppo. Ho iniziato questo percorso con diverse perplessità e dubbi che alla fine del progetto mi sento di aver chiarito. Personalmente l'approccio che vedo funzionare meglio con me, che mi permette di fare chiarezza con i concetti teorici e alcune dinamiche, è capire le cose mentre si fanno. Il tempo speso in questo progetto mi ha permesso infatti di capire bene i concetti spiegati a lezione e di poterli usare con facilità. Prima di iniziare questo percorso di gruppo, Git e GitHub, a parte come concetto, non avevo capito come funzionassero e mi spaventava l'idea di usarli. Questo progetto mi ha dato l'occasione per capire bene come utilizzare questi strumenti e creare delle linee guida per poter agevolare anche i miei compagni. Per il corretto avanzamento del lavoro è stato fondamentale rimanere dopo le lezioni per potersi confrontare, valutare cambiamenti e discutere insieme dell'approccio megliore da seguire. Ho trovato i miei compagni molto prepa-

rati, professionali e predisposti al lavoro di gruppo. Nonostante i problemi dovuti all'inesperienza, sono molto felice del mio contributo e di ciò che ho imparato.

4.1.5 Balzoni Margherita

Mi ritengo soddisfatta del lavoro da me svolto e dall'organizzazione e l'impegno del gruppo. Nonostante le difficoltà iniziali causate principalmente dall'inesperienza, poichè questo è stato il primo vero progetto per la maggior parte dei componenti del gruppo (me compresa), una volta preso il via siamo riusciti a svolgere i compiti a noi assegnati collaborando e ripettando le scadenze da noi prefissate. Ciò ha permesso di avere un avanzamento costante del progetto.
Grazie agli incontri fatti su Teams siamo riusciti a creare una distribuzione equilibrata del lavoro che ha permesso di evidenziare i punti di forza di ogni componente e chiarire, grazie all'aiuto dei compagni , dubbi e lacune. Ho apprezzato moltissimo, infatti, lo scambio reciproco di idee e la disponibilità nell'aiutarci a vicenda nei momenti in cui sono insorte difficoltà.
Questo progetto mi ha permesso di mettere in pratica e compredere meglio argomenti svolti a lezione che ancora non sentivo pienamente miei e di confrontarmi con idee diverse dalle mie impegnandomi a trovare soluzioni che andassero bene a tutti i componenti del gruppo.

4.2 Guida utente

All'avvio dell'applicazione si presenta la seguente schermata di menu iniziale:

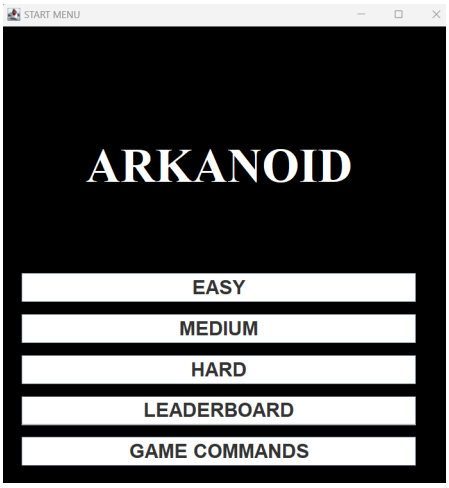

Figura 4.1: Schermata di menù iniziale.

Abbiamo 5 bottoni che hanno diverse funzionalità:

- Premendo il pulsante EASY oppure MEDIUM oppure DIFFICULT si avvierà il gioco corrispondente alla difficoltà che viene selezionata.Figura 4.3
- Premendo il pulsante LEADERBOARD si potrà visionare la classifica.
- Premendo il pulsante GAME COMMANDS si potranno visionare i comandi di gioco.Figura 4.2

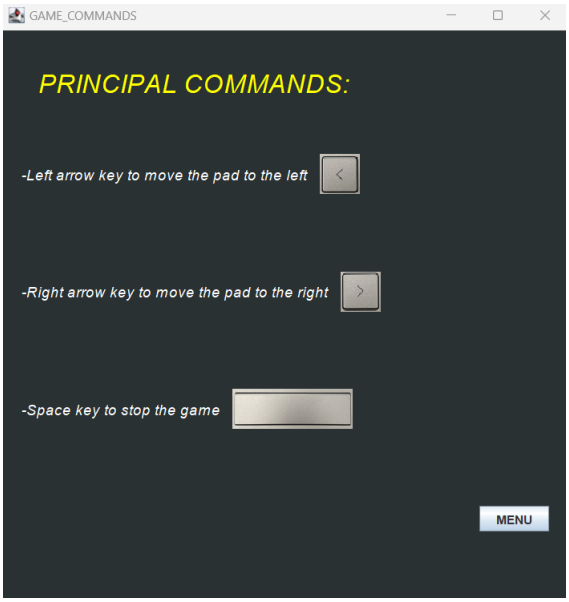

Figura 4.2: Schermata dei comandi.

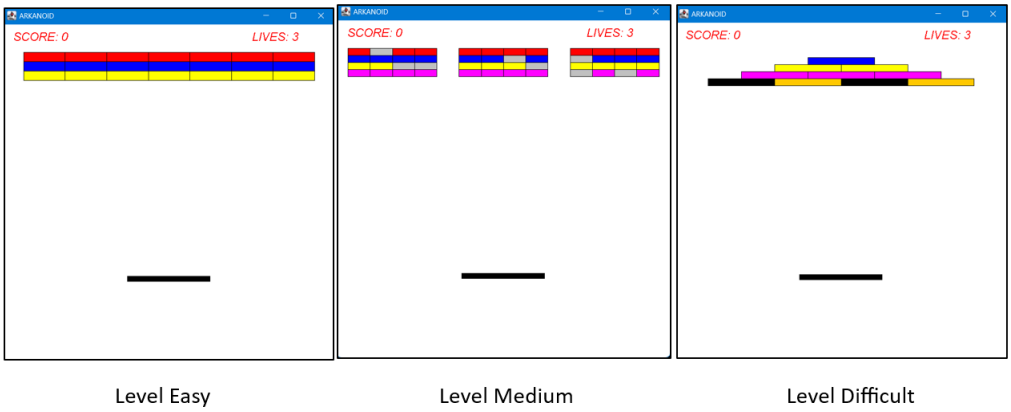

Figura 4.3: Schermata dei tre livelli.

Una volta premuto il livello desiderato, la schermata sarà una di questre tre Figura 2.5, in base al livello scelto, e la pallina inizierà subito a muoversi.

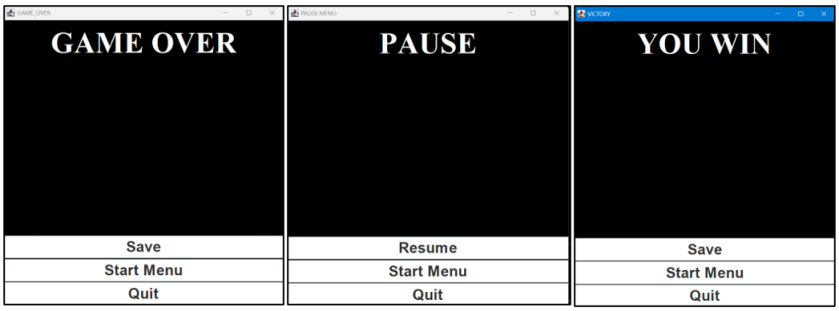

Figura 4.4: Schermate dei menù di pausa, gameover e vittoria.

Nei menù di GameOver e Victory si potrà scegliere se salvare il proprio punteggio, ritornare al menù iniziale oppure uscire del tutto dall'applicazione. Stesse azioni sono possibili farle nel menù di pausa ad eccezzione del pulsante save che viene sostituito da quello che ha il compito di far ritornare l'utente alla partita in corso prima di averla stoppata (Resume).