Flappy Bird

Gabriel Sergiu Rolnic, Arianna Zannoni

Anno Accademico 2020/2021

Indice

Capitolo 1

Analisi

L'applicazione, sviluppata come elaborato di "Programmazione ad Oggetti" del corso di Ingegneria e Scienze Informatiche dell'Università di Bologna, ha come obiettivo la realizzazione del gioco Flappy Bird in versione desktop.

1.1 Requisiti

Requisiti funzionali

- Al avvio dell'applicazione, il giocatore potrà iniziare una nuova partita o scegliere un uccello diverso da usare.
- Egli dovrà far volare un uccello attraverso una serie di tubi, evitando di farlo scontrare con essi o di farlo cadere a terra.
- L'uccello, quando non riceve comandi, subisce l'effetto della gravità e perde quota.
- L'obiettivo è quello di totalizzare il punteggio più alto possibile (il giocatore guadagna un punto per ogni tubo attraversato)
- Inserire il proprio nome al termine della partita, salvare il proprio punteggio e visualizzare la classifica.

Requisiti non funzionali

- L'applicazione nel suo utilizzo dovrà garantire fluidità.

1.2 Analisi e modello del dominio

Il dominio dell'applicazione descrive il volo di un uccellino attraverso una serie di ostacoli, e senza toccare il pavimento, con lo scopo di raggiungere un piu' alto score possibile.Esso viene rappresentato attraverso un'entità "universo" World, all'interno della quale vengono aggregate tutte le componenti che ne fanno parte. L'uccellino (Bird) e gli ostacoli(Column) sono dei GameObject mentre la collisione (Collision) dovrà gestire sia lo scontro con gli ostacoli che lo scontro con il pavimento. L'ultimo componente ad essere aggregato a World è l'entità Score. Essa dovrà aggiornarsi ogni volta che viene raggiunto un ostacolo. La complessità maggiore sarà data dall'avere un aggiornamento fedele per ogni entità in gioco.

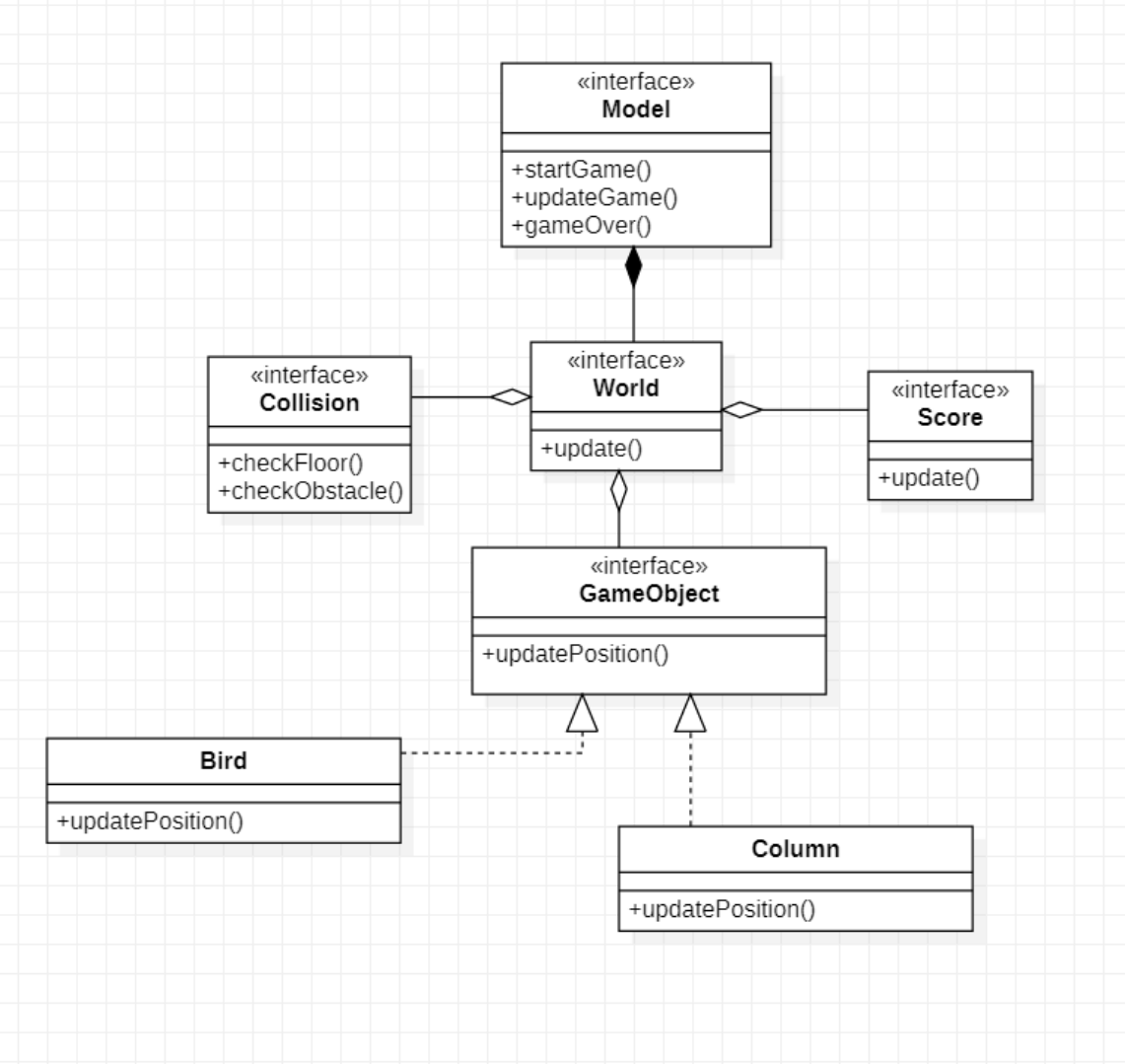

Figura 1.1: Schema UML dell'analisi del dominio

Capitolo 2

Design

2.1 Architettura

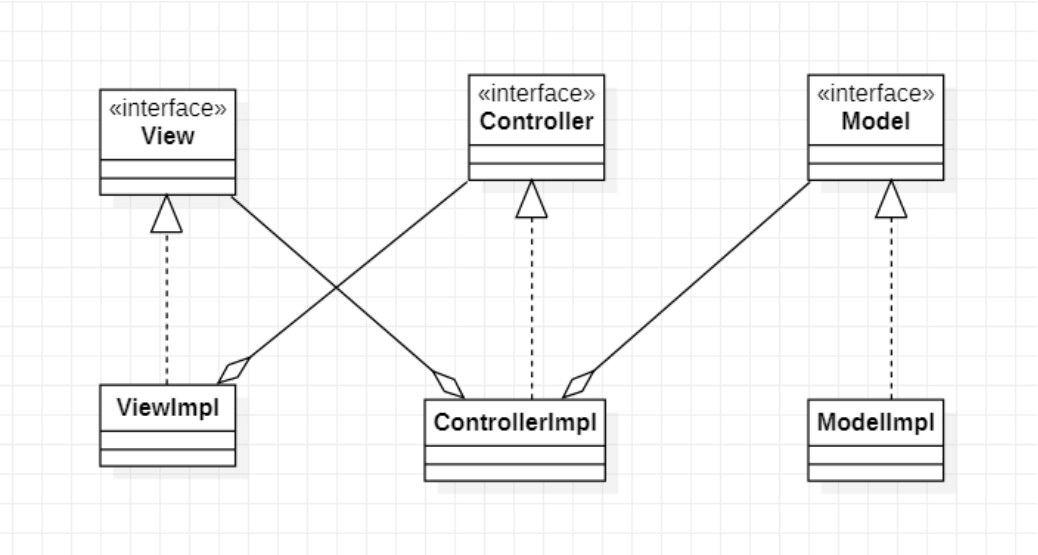

Figura 2.1: MVC

Per lo sviluppo del sistema è stato deciso di seguire il pattern architteturale MVC, in una maniera molto rigida. Il model gestisce il dominio, tutta la logica(incluso il gameloop) e l'interfaccia con il file system, la view si occupa dell'interfaccia con l'utente, mentre il controller ha **esclusivamente** il ruolo di ponte tra le altre due entità e di aggiornamento dello stato di gioco. Non esiste nessun collegamento diretto tra model e view, in modo poter sostituire o aggiungere diverse user interface senza toccare minimamente il resto del codice. La view quando viene lanciata crea il controller, il quale istanzia un

nuovo model, che farà partire il gameloop quando sarà il momento del gioco vero e proprio.

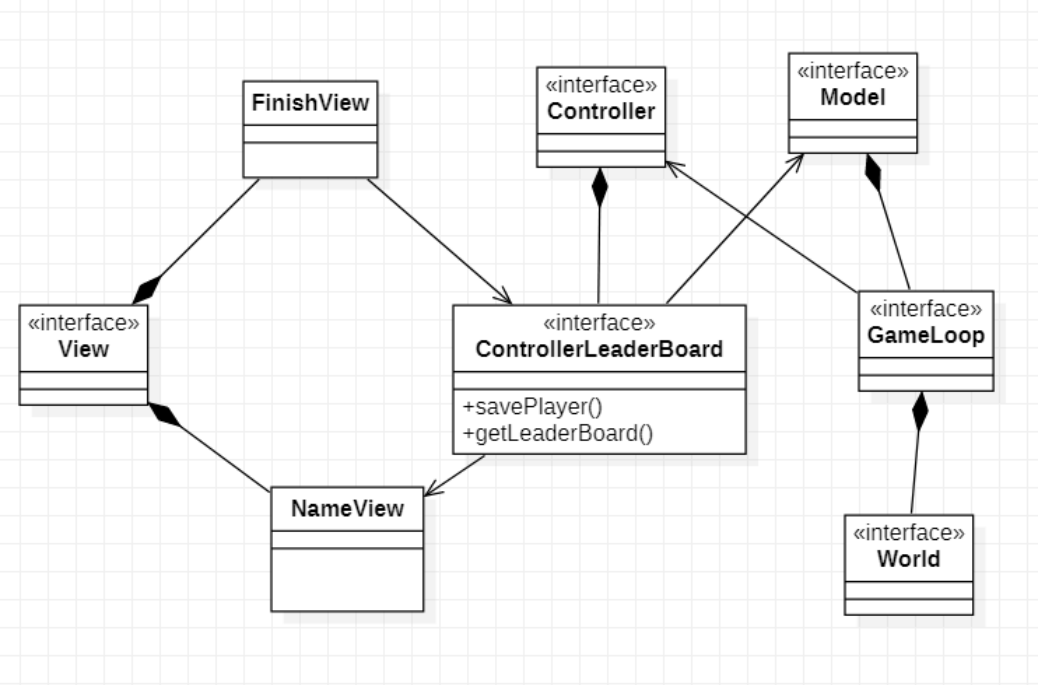

Figura 2.2: Schema dettagliato interazione componenti principali

2.2 Design dettagliato

2.2.1 Sergiu Gabriel Rolnic

Column Creation

Uno dei principali problemi che mi si è posto davanti per quanto riguardava la creazione degli ostacoli era trovare un modo per rispettare DRY e KISS,in quanto il vari oggetti avrebbero presentato leggerissime differenze, e volevo anche avere la possibilità di fare nuove aggiunte in futuro, quindi avere codice riusabile era fondamentale. Ho capito subito che la scelta procedurale sarebbe ricaduta sul uso del template method o dello strategy. A livello di funzionalità sembrava fosse piu' appropriato l'uso dello strategy, in quando le principali differenze implementative degli ostacoli erano una semplice questione algoritmica. Tuttavia, il pattern deve anche rappresentare l'essenza, la struttura e la funzionalità delle classi, perciò il **Template Method** mi è sembrato molto piu' appropriato a livello concettuale. Perciò classe

astratta AbstractColumn contiene il template method *setHeight()*, il quale va a richiamare il metodo astratto *updateHeight()*. Ogni tipologia di ostacolo implementerà in maniera specifica quest ultimo metodo.

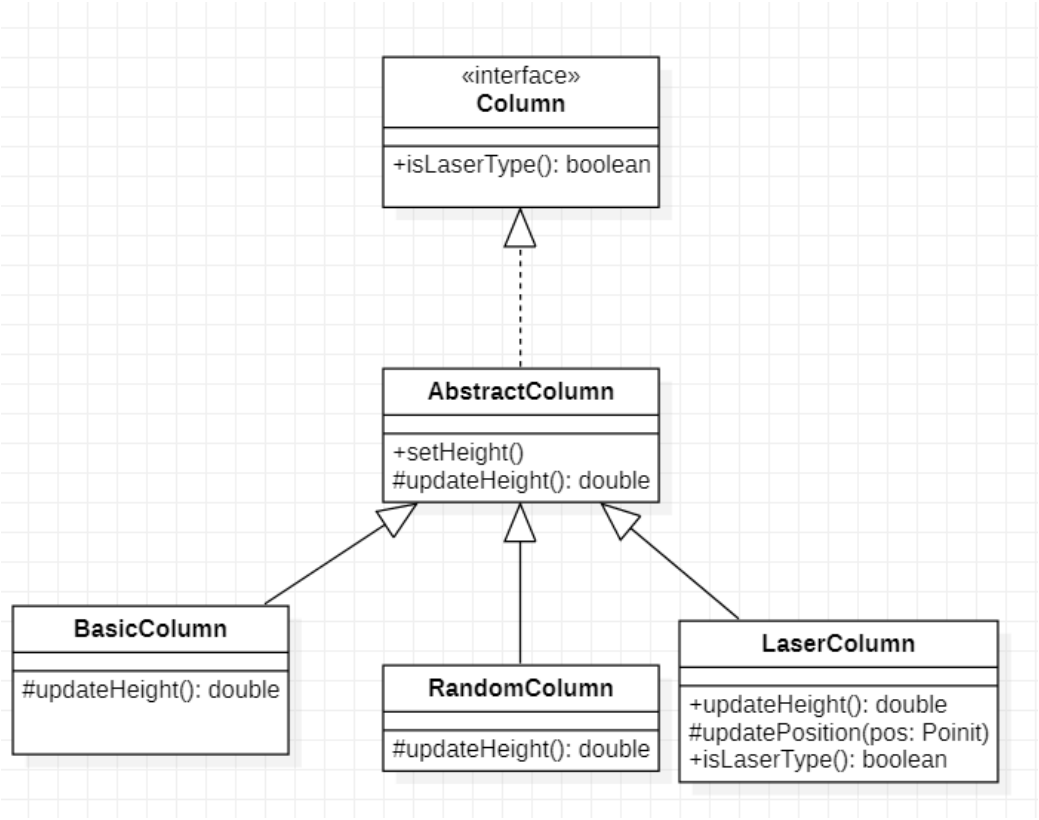

Figura 2.3: Template Method

Column Manager

Successivamente, si è reso il bisogno di trovare una maniera efficace per la gestione degli oggetti, pensando anche di tornare indietro e usare un pattern creazionale che desse piu' libertà di gestione, ma nessuno di quelli conosciuti a lezione riusciva a soddisfare in maniera totale o quantomeno sufficiente, le esigenze implementative. Sarebbe risultata una forzatura che non avrebbe neanche risposto a tutti i problemi emersi. La soluzione è stata rendersi conto che effettivamente l'idea iniziale di di vedere gli ostacoli come se fossero anche degli algoritmi era corretta. Perciò, è stata creata la *funtional interface* OperationGenerate, che svolge il ruolo di *Supplier* per la classe ObstacleGenerator,la quale si occupa di istanziare gli ostacoli e posizionarli

correttamente durante il gioco. L'uso delle *lambda expressions* ha permesso di beneficiare nel migliore dei modi del **pattern Strategy**

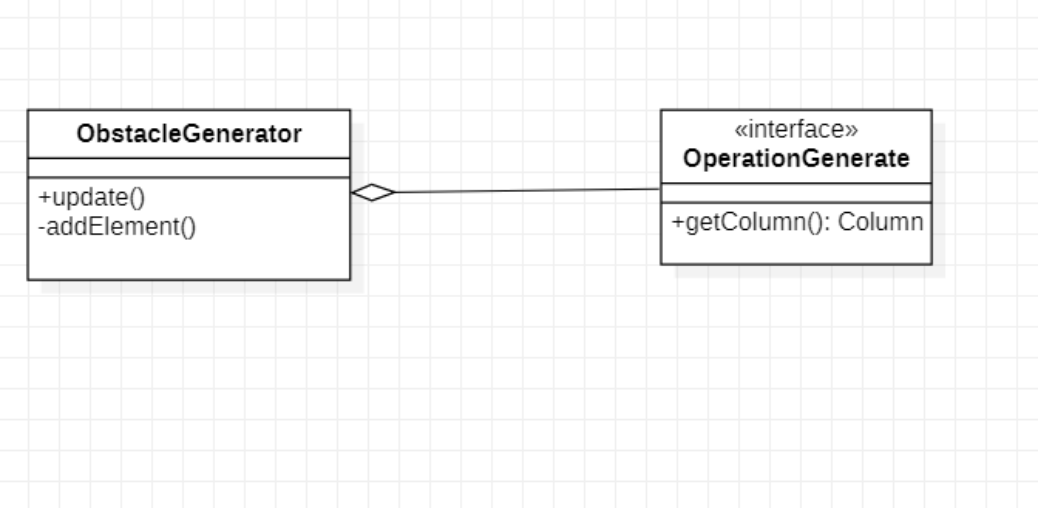

Figura 2.4: Strategy Pattern

File Manager

Per quanto riguarda il salvataggio delle statistiche sui giocatori, il problema principale era trovare un modo efficace di trasmettere i dati al File System, e contemporanemente gestire la loro interazione con le altre entità del sistema. La soluzione è stata la libreria Gson, la quale mi ha permesso, una volta creata la classe Gamer, di traformare in formato json, tutti gli oggetti di tipo Gamer, ed effettuare in maniera semplice ed efficace le operazioni di lettura e scrittura su file. E' bastato creare una lista di Gamer e passarla alla libreria per permettere l'interazione su file.

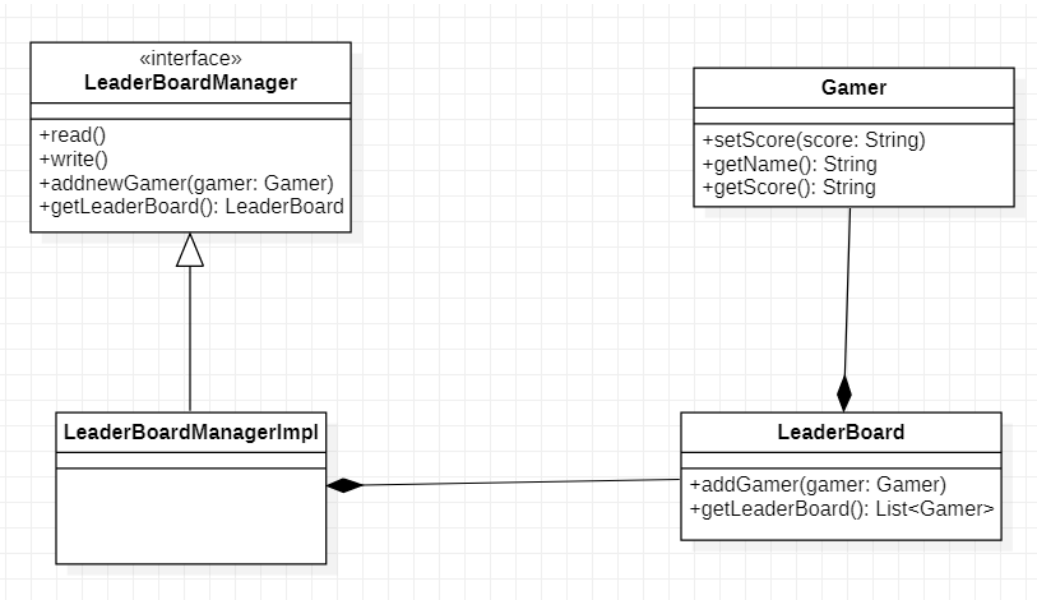

Figura 2.5: Schema sulla gestione della LeaderBoard

2.2.2 Arianna Zannoni

In questa sezione l'attenzione è focalizzata sull'implementazione del menù principale, della logica del Bird e sulla gestione delle collisioni.

Menù Principale

Il menù principale è composto da uno sfondo sopra il quale sono posti 3 bottoni:

- START: Inizio di una nuova partita.
- SELECT PLAYER: Apertura di una nuova schermata dove poter selezionare l'immagine del giocatore da voler utilizzare per la partita.
- EXIT: Chiusura della schermata di gioco.

Bird

Per la gestione dell'uccellino ho creato la classe BirdImpl che implementa l'interfaccia Bird e si occupa di definire la posizione iniziale dell'oggetto. Per quanto riguarda invece la logica dell'uccellino, ho definito un package a parte di model *'model.manager'* composto da 4 classi e le rispettive interfacce:

- **ManagerBirdImpl**: questa è la classe principale che si occupa di richiamare le altre classi per definire il comportamento dell'uccellino.
- **ManagerGravityImpl**: si occupa della gravità applicata all'uccellino. Tramite il metodo *fallBird()* viene controllata la coordinata Y dell'uccellino ed in base al suo valore viene incrementata per fargli perdere quota.
- **ManagerJumpImpl**: viene decrementata la coordinata Y dell'uccellino ogni volta che c'è un input da tastiera.
- **ManagerCollisionImpl**: vengono definiti due metodi che controllano rispettivamente la collisione con le colonne (*'checkColumnCollision'*) e la collisione a terra (*'checkFloorCollision'*).

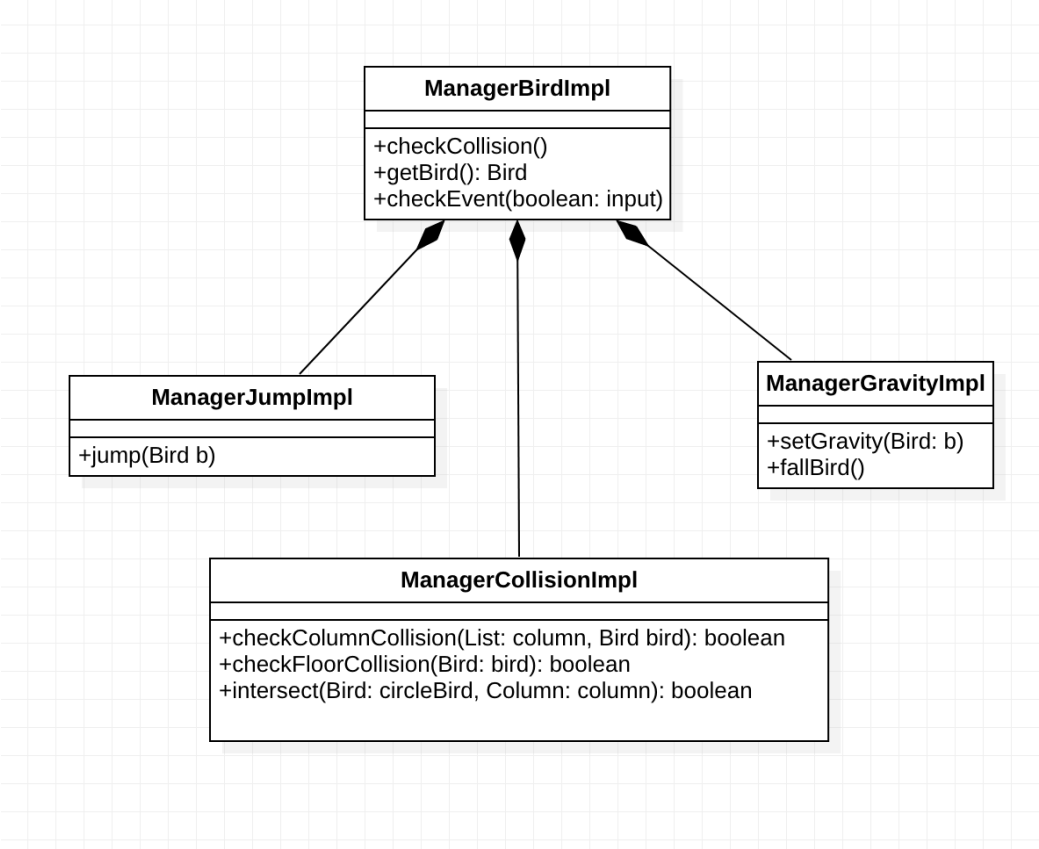

Figura 2.6: Pattern Observer

Dopodichè mi si è posto il problema di dover gestire in qualche modo la comunicazione tra View e Controller per per permettere alla prima di

avvisare il Controller ogni volta che intercetta un input dell'utente. Per questo ho ritenuto utile da utillizzare il **Pattern Observer** che è risultato particolarmente efficacie per gestire l'input della barra spaziatrice per far saltare l'uccellino.

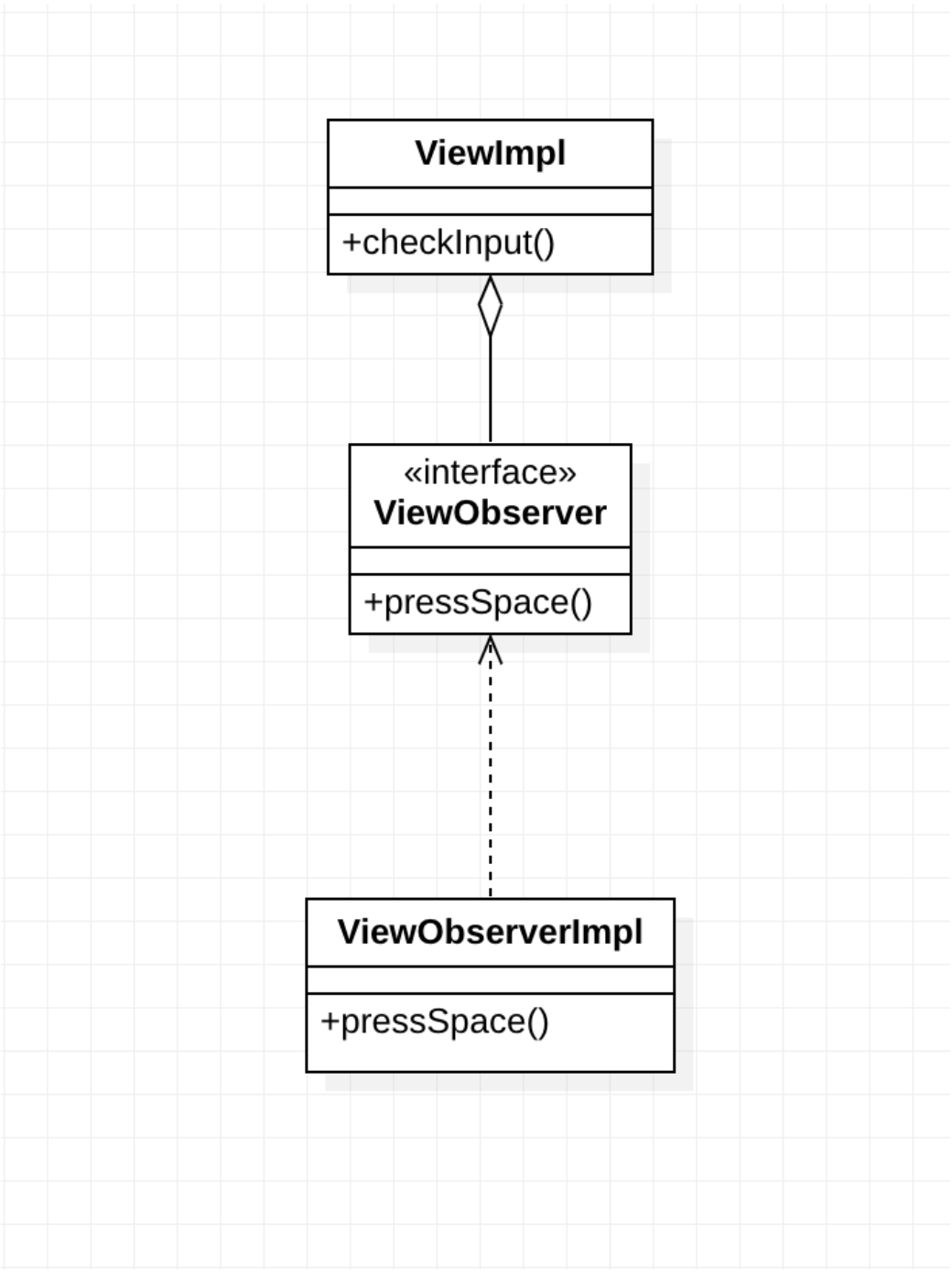

Figura 2.7: Pattern Observer

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per quanto riguarda il testing, si è fatto uso della suite JUnit 5 per verificare se la seguenza di ostacoli prodotta fosse coerente con lo step di gioco,se le dimensioni rispettassero le regole di creazione e se l'aggiornamento delle posizioni fosse corretto. Allo stesso modo si è proceduto con Bird

3.2 Metodologia di lavoro

3.2.1 Gestione del lavoro

Il progetto è stato sviluppato attraverso una fase di analisi dove entrambi abbiamo posto le basi per il dominio del applicativo. Successivamente, grazie al DVCS GIT, a github, e al template gradle fornito a lezione, abbiamo creato la repo del progetto, assieme ai 4 packages che sarebbero stati il punto di partenza per lo sviluppo: model, view, controller con le loro rispettive inferfacce, e flappybird, da dove verrà lanciata la nostra applicazione. Nel master abbiamo settato anche le impostazioni generali per Eclipse dei vari plug-in, tra cui PMD, Checkstyle e FindBugs. Successivamente abbiamo creato il branch develop, che sarebbe stato il punto di giunzione delle varie modifiche, e da esso ognuno si è spostato a lavorare su un branch proprio per la parte di model.

3.2.2 Sergiu Gabriel Rolnic

All'interno del progetto mi sono occupato della gestione degli ostacoli, dello score, del salvataggio della leaderboard sul FileSystem, del gameloop, e ov-

viamente delle rispettive parti di view. L'integrazione delle parti e quindi la necessità di lavoro comune c'è stata nelle interfacce View, Controller, Model, GameLoop, World e rispettive implementazioni.

3.2.3 Arianna Zannoni

In questo progetto mi sono occupata della gestione della logica dell'uccellino e della sua parte di view e della creazione del menù principale.

3.3 Note di sviluppo

3.3.1 Sergiu Gabriel Rolnic

- **Stream** Usati nella classe LeaderBoardManagerImpl
- **Optional** Usato per gestire la presenza o meno di un TopScore
- **Lambda expressions** Usate in maniera sparsa
- **Gradle**
- **JavaFX**
- **Gson** Libreria di Google usata per la trasformazione di oggetti java in formato Json, in modo da agevolare notevolmente il processo di salvataggio su FileSystem.

3.3.2 Arianna Zannoni

- **Lambda expressions**
- **Gradle**
- **JavaFx** : utilizzato per gli oggetti di tipo Circle (Bird).

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Sergiu Gabriel Rolnic

La principale soddisfazione è sicuramente quella di essere riuscito a portare a termine lo sviluppo del progetto, nonostante fosse la prima esperienza in ambito gaming e java, e nonostante il numero limitato di membri nel gruppo. Questo fatto ha fortemente penalizzato la creazione di codice di qualità, dovendo mantenere una visione a 360° su ogni aspetto dello sviluppo, ma ha portato anche una base molto solida riguardo le principali tecnologie e tecniche di progettazione o implementazione, che altrimenti sicuramente non sarebbe stato possibile avere. Avendo rispettato i tempi richiesti per l'attività di progetto, e considerando le premesse elencate poco sopra, ritengo senza ombra di dubbio che si potesse fare ben poco di piu' di quel fatto.

4.1.2 Arianna Zannoni

Lo sviluppo del progetto è stato per me molto impegnativo, ma allo stesso tempo soddisfacente al termine del lavoro. Sviluppare questo progetto mi ha portato sicuramente molte conoscenze in più sulla materia e una visione più ampia in generale. Fin da subito,avendo analizzato il progetto e la sua realizzazione con il mio collega,mi sono accorta che essendo in due sarebbe stato sicuramente più faticoso. A questo , per quanto mi riguarda,si aggiunge anche il minor tempo avuto a disposizione per via del lavoro. Per questo ringrazio il mio compagno di progetto per la assoluta disponibilità e collaborazione.

4.2 Difficoltà incontrate e commenti per i docenti

Appendice A

Guida utente

Il gioco si apre con la schermata principale del menu, che mostra tre semplici bottoni:

- START: cliccando su questo pulsante inizia una nuova partita.
- SELECT PLAYER: cliccando si apre una schermata in cui è possibile scegliere quale immagine utilizzare per il personaggio 'Bird'.
- EXIT: cliccando la schermata si chiude e si esce dal gioco.

  Appena inizia la partita il giocatore deve premere la barra spaziatrice per permettere all'uccellino di saltare attraverso i tubi.

  La partita termina quando l'uccellino si scontra con un tubo o cade a terra.

  Una volta terminata la partita l'utente dovrà inserire il proprio nome e premere il tasto 'Save' per salvare il proprio punteggio e inviarlo allo storico. Nella schermata finale sono presenti due bottoni:

- TASTO PLAY: cliccando si ritorna alla schermata prinipale.
- SCOREBOARD: cliccando si apre una schermata che visualizza lo storico dei punteggi.

Appendice B

Esercitazioni di laboratorio

Bibliografia