Relazione
"Il Migglione"

Alessandro Zani
Lorenzo Zanoni
Thomas Giovannini
Anton Abramov

15 Febbraio 2026

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software si pone l'obiettivo di realizzare la versione digitale del gioco di carte inventato *Il Migglione*. Il progetto originale deriva da un'idea di poter ritrarre i vari scherzi di un gruppo di amici formatasi nel corso degli anni sotto forma di carte, essendo quello un passatempo comune tra loro. Il meccanismo di gioco è piuttosto semplice: ogni carta dispone di cinque attributi i cui valori vagliano tra uno e dieci, il giocatore di turno sceglie la categoria su cui giocare la carta e in senso orario ogni altro giocatore seleziona una delle proprie carte da giocare. Una volta che tutti i giocatori hanno fatto la loro scelta, si passa al momento della conteggia dei valori, in cui il giocatore con la carta dal valore più alto nella categoria scelta vince il round. Dopodiché, tutte le carte giocate diventano i punti di quest'ultimo, e la partita prosegue. Il giocatore di turno cambia in soli due casi: quando un altro giocatore vince il round oppure se lo stesso giocatore ha già vinto tre volte di seguito, a quel punto si passerà al giocatore dopo in senso orario. A fine di ogni round ogni giocatore, a partire dal vincitore e poi in senso orario, riceve una nuova carta dal mazzo comune, in modo da avere sempre tre carte in mano. La partita termina quando tutte le carte sono state giocate, e a quel punto si dichiara vincitore della partita il giocatore con più punti.

Requisiti funzionali

- Il software presenterà un menù con varie opzioni, esse permetteranno di iniziare la partita, rivedere il tutorial, consultare una galleria delle carte, vedere la classifica dei punteggi, mostrare i riconoscimenti e uscire dal gioco.
- Prima di iniziare la partita verrà chiesto un nome da usare, in modo tale da poter salvare il risultato e porlo nella classifica.

- Durante la partita, ogni giocatore potrà vedere solo le proprie carte, mentre lo score sarà visibile da tutti e il mazzo è completamente inaccessibile.
- La griglia di gioco sarà molto semplice e si prevede un solo tipo di modalità giocatore contro CPU, rappresentata da una mosquito IA che farà sempre la mossa migliore possibile.
- Una volta selezionata una carta il turno passerà automaticamente all'altro giocatore, quindi ogni scelta è definitiva.
- Finita la partita si decreterà il vincitore e il punteggio del giocatore verrà salvato solo se a vincere è stato questo; dopodiché, si ritornerà al menù principale.

Requisiti non funzionali

- Il software deve garantire meccanismi che permettono un gioco fluido e veloce, cosicché le partite non risultino tediose.
- La grafica utilizzata permette che possa essere adattata in maniera dinamica, ma si prefigge un utilizzo su schermi da computer.
- L'interfaccia di gioco deve essere ordinata e intuitiva, in modo che anche i giocatori che sceglieranno di ignorare il tutorial non siano soverchiati da ciò che accade a schermo.

1.2 Modello del Dominio

Il giocatore dovrà essere in grado di poter liberamente scorrere tra le carte della propria mano, in modo da consultare i loro attributi e scegliere la mossa che gli consentirebbe di vincere. Similmente dovrà fare la CPU, cosicché il giocatore potrà contare sul fatto che se questa ha una carta molto buona la giocherà immediatamente. Inoltre, il gioco non permetterà di passare il turno senza prima aver giocato una carta, consentendo all'utente di pensare alla propria mossa con calma. Sarà necessario rendere il conteggio di punti e di carte rimanenti nel deck, nonché quali carte possono ancora venir estratte, dinamico, a fronte anche di eventuali possibili altre modalità che richiederebbero un uso del mazzo particolare. Il sistema provvederà poi a fare sì che ogni partita sia diversa, in modo che il giocatore non possa sapere quali carte riceverà in seguito.

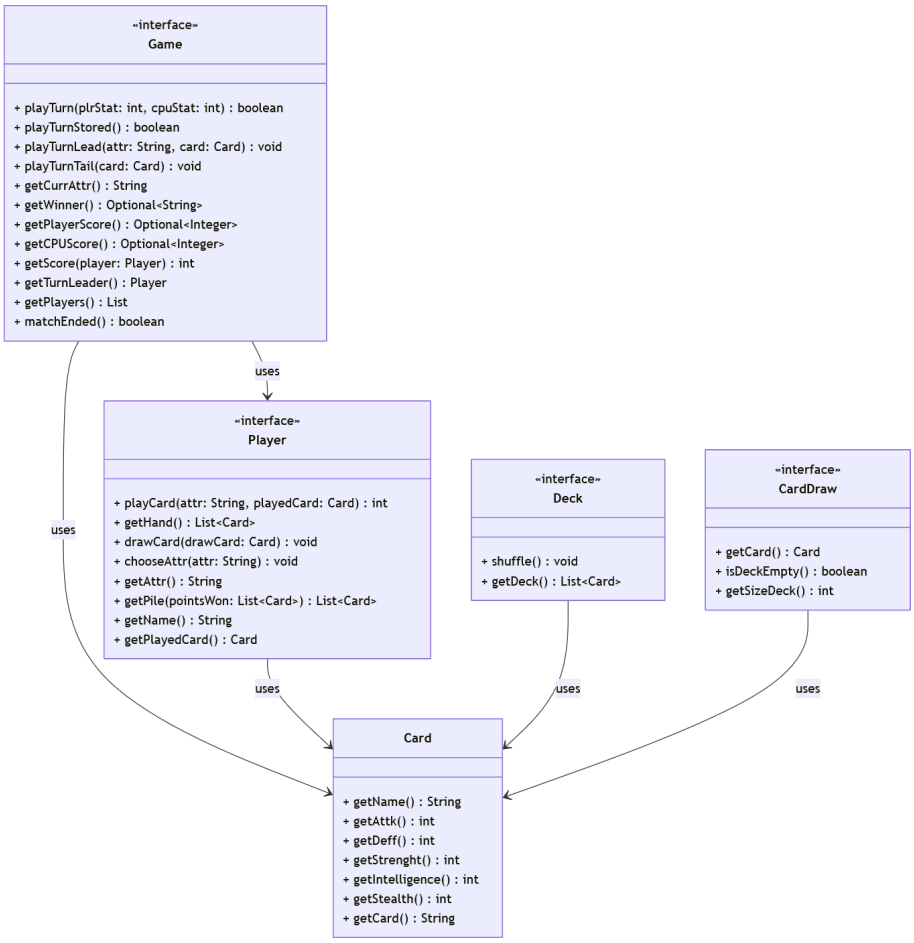

Figura 1.1: UML che mostra le principali entità che costituiscono il dominio

Capitolo 2

Design

Questa sezione è dedicata a fornire una visione generale e di alto livello di quello che costituisce il Model, la View e il Controller in questo progetto. Ci si concentrerà a esporre come questi componenti interagiscono tra loro seppur mantenendo un certo livello di indipendenza, garantendo quindi che venga rispettato il modello MVC.

2.1 Architettura

L'architettura del *Migglione* utilizza il pattern architetturale MVC, ossia Model-View-Controller, per garantire indipendenza ed estensibilità al codice. In particolare, nel nostro caso sarà possibile sostituire la View liberamente e senza impattare in nessun modo Controller o Model, denotando eventuali aggiunte o, in generale, cambiamenti futuri delle funzionalità esposte all'utente. L'elevata flessibilità e modularità delle sue componenti può essere osservata nel diagramma UML sottostante, mentre possiamo osservare nel dettaglio:

- **Model:** la sua classe entry point è Game, e costituisce la parte logica dell'applicazione, gestendo tutto ciò che riguarda la partita e i dati dei giocatori. In particolare, i suoi compiti riguardano: inizializzazione e uso di carte, deck e mano, nonché la partita stessa e l'utilizzo dei punteggi ottenuti.
- **View:** attraverso SwingView come classe di entry point, si è stato scelto Swing come meccanismo per presentare al giocatore il menù con tutte le sue componenti e successivamente, iniziata la partita, il campo di gioco e la propria mano di carte. Avrà infatti il compito di gestire le varie scene e provvedere alla transizione tra queste.
- **Controller:** arbitro tra il Model e la View, è definito dalla classe di entry point sua omonima; grazie alla sua indipendenza dalle altre componenti

anche un'ipotetica aggiunta di funzionalità al Controller non le comprometterà in alcun modo e permette che sono certe parti del Model possano essere utilizzate dalla View, sincronizzando il loro stato.

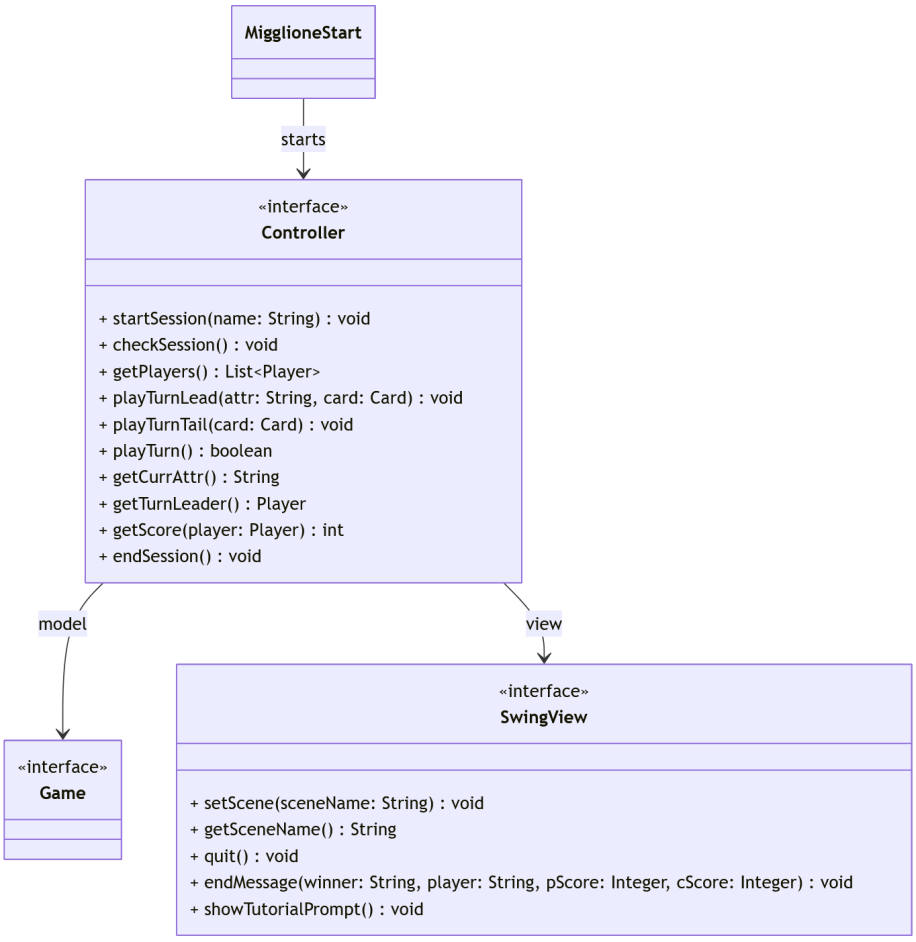

Figura 2.1: UML che dimostra l'utilizzo del Controller. Una volta inizializzata la View, potrà all'occorrenza chiamare il Model attraverso il Controller stesso

2.2 Design dettagliato

In questa parte del capitolo si esplorerà nel dettaglio gli aspetti chiave del *Miglione*, divisa in base ai contributi singoli dei partecipanti del progetto e che espone i vari problemi riscontrati durante la progettazione, la loro risoluzione e un diagramma UML che ne illustri le classi coinvolte.

2.2.1 Alessandro Zani

Creazione e cambio di scena

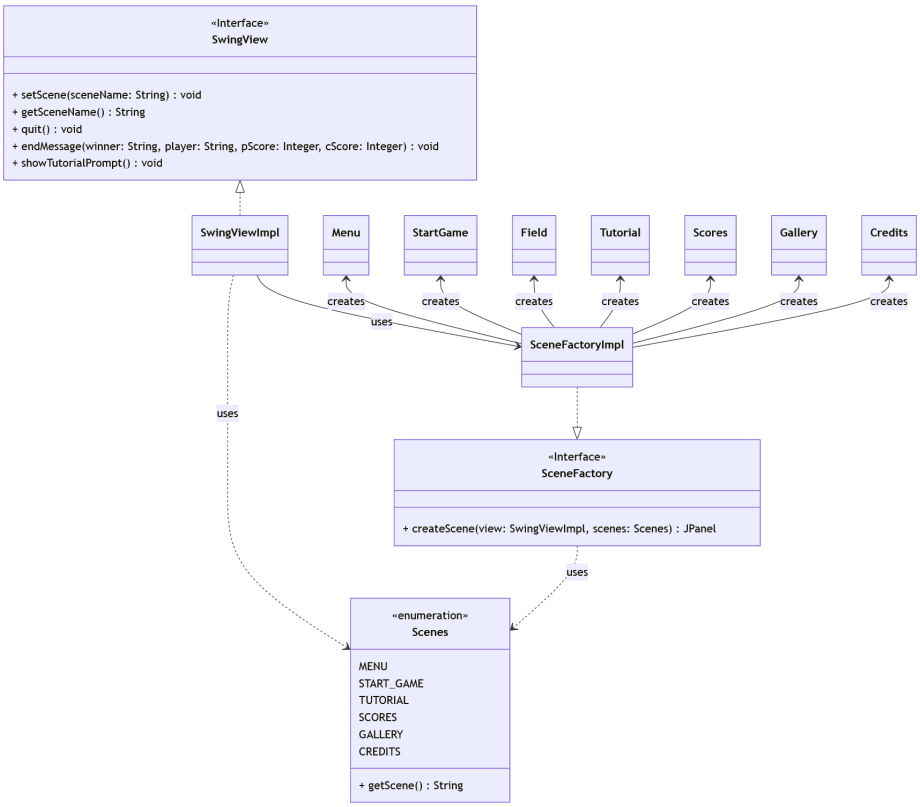

Figura 2.2: Schema UML per la creazione delle scene

Problema: Durante la costruzione della implementazione di SwingView, si è rivelato necessario far sì che le "scene" possano cambiare, o più semplicemente poter vedere altre schermate oltre a quella principale. Essendo che queste devono dipendere tutte dalla classe entry point, è necessario collegarle in maniera efficiente e precisa.

Soluzione: Per poter permettere una centralizzazione delle scene, si è utilizzato il metodo Factory con classe SceneFactory e la sua omonima implementazione, cosicché all'interno della classe SwingViewImpl si possa semplicemente richiamarla e utilizzarne il metodo createScene. Successivamente, per poter effettivamente cambiare scena, basterà usufruire del metodo setScene, che, utilizzando il nome della scena da applicare a schermo, questa verrà effettivamente mostrata all'utente, con le eventuali modifiche. L'utilizzo della enum Scenes aggiunge un grado di separazione e modularità non indefferente.

Utilizzo di una immagine di background

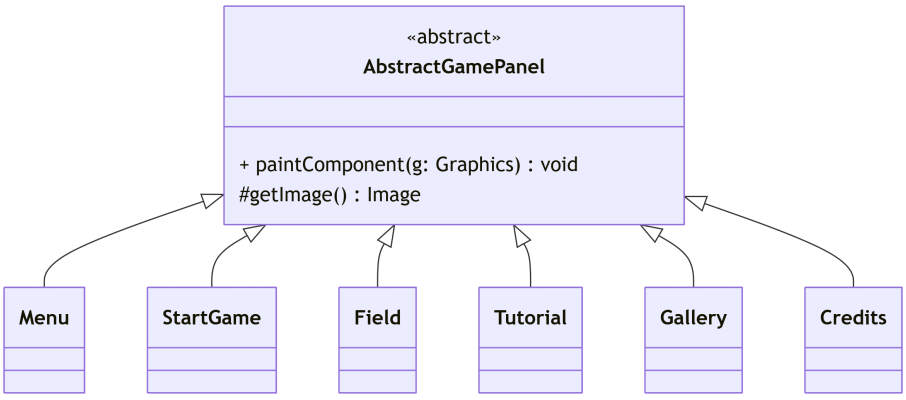

Figura 2.3: UML per la gestione delle immagini nelle scene

Problema: In fase di sviluppo, si è potuto osservare che tutte le scene condividevano uno stesso metodo per definire e adattare le immagini al background, generando quindi casi di duplicazione e ridondanza del codice.

Soluzione: Facendo sì che le scene estendessero la classe astratta GamePanel, abbiamo potuto definire in questa il metodo paintComponent responsabile del draw delle immagini; dato che questo illustra lo schema fisso è corretto definirlo il metodo Template. Il passo variabile, cioè il metodo astratto getImage, verrà invece implementato in ciascuna scena permettendo la massimizzazione del riuso di parti in comune.

Scelta e riproduzione di musica

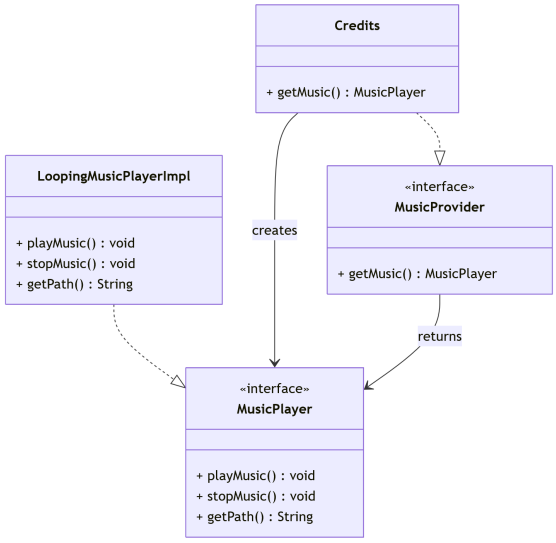

Figura 2.4: Schema UML che mostra l'utilizzo della musica nel progetto

Problema: Dato che ogni scena avrebbe utilizzato della musica in sottofondo, ci si è presto trovati a ragionare sulla possibilità di avere nel futuro molteplici modi di riprodurre le soundtrack, per cui era d'obbligo trovare un modo per offrire intercambiabilità a runtime, senza cadere in casi di ripetitività di codice.

Soluzione: Il metodo più conveniente per gestire tale situazione è sicuramente quello dato dallo Strategy method. Utilizzando infatti come interfaccia che rappresenta la strategia MusicPlayer si potrà cambiare a piacimento le specifiche delle musiche in ogni scena, essendo che queste implementano il provider di Strategy MusicProvider e ritornano un'istanza di MusicPlayer. Nel caso di questo UML si è deciso di mostrare Credits per pulizia, ma appunto tutte le scene seguono questo pattern.

Mazzo e pescata dinamici

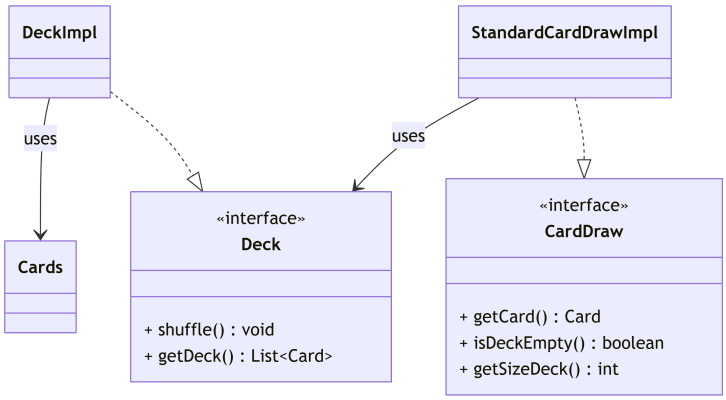

Figura 2.5: UML per la possibile scelta tra mazzi e metodi di pescata diversi

Problema: Essendo un gioco di carte per natura ampiamente modificabile e disponibile all'aggiunta o rimozione di proprietà e regole, si è resa necessaria la capacità di cambiare i possibili mazzi che si possono usare e i metodi di pescata a partire da tali mazzi.

Soluzione: Anche in questo caso la risposta più ovvia a questo tipo di problema è indubbiamente l'utilizzo di metodi Strategy. Come osservabile in figura infatti sia la classe Deck che quella CardDraw offrono la capacità di essere implementate in infiniti modi diversi, a patto che si aderisca all'utilizzo e definizione dei loro metodi principali. DeckImpl si limiterà a creare una lista a partire da ciò contenuto in Cards, ma non vieta in futuro l'introduzione di altri tipi di Deck che possono integrare Card diverse o in maniera differente. Similmente, StandardCardDrawImpl fa sì che venga eseguita la normale pescata dal mazzo, ma grazie all'interazione di CardDraw con Deck non vieta la creazione di pescate particolari per altre modalità di gioco.

2.2.2 Lorenzo Zanoni

Creazione dei metodi di un player generico

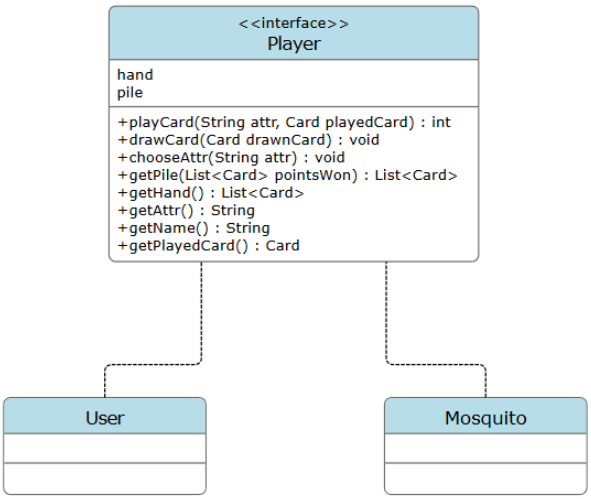

Figura 2.6: UML per i metodi di Player e User

Problema: Per fare la CPU, a monte c'era la necessità di creare dei metodi comuni sia all'utente che alla CPU, in modo tale che potessi poi usarli già per l'implementazione della effettiva CPU e creare effettivi metodi per far giocare l'utente stesso.

Soluzione: Ho creato un interfaccia Player, in cui sono descritti tutti i metodi comuni sia a Mosquito(la CPU) che allo User, in particolare ogni Player ha una mano(hand), una pila dei punti vinti(pile) e poi si usano alcune variabili per la sincronizzazione durante la partita: chosenAttr per l'ultimo attributo scelto dal player e lastPlayed per l'ultima carta scelta dal player. I metodi comuni sono i getter e drawCard, mentre per playCard e getPile, Mosquito attua specifiche operazione per giocare autonomamente.

Creazione di metodi per rendere la CPU dinamica

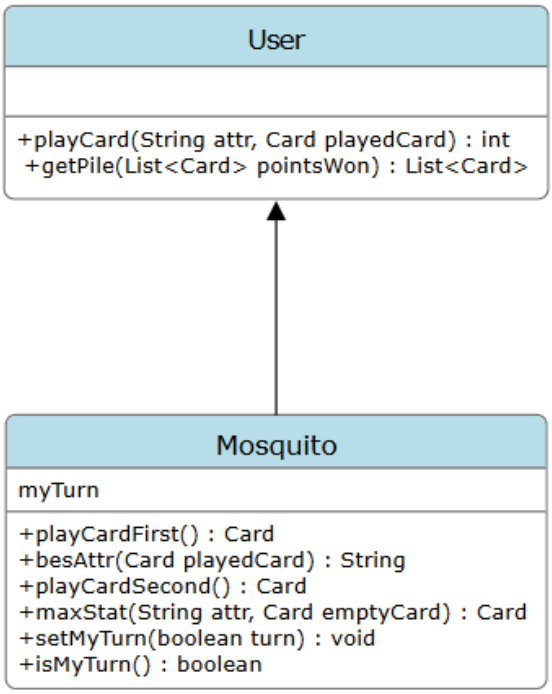

Figura 2.7: UML per i metodi di Mosquito

Problema: I metodi playCard e getPile in Mosquito sono leggermente diversi: se la CPU deve giocare seconda, dovrà usare l'attributo selezionato dall'utente, viceversa se gioca per prima dovrà autonomamente scegliere l'attributo che più avrà possibilità di vincere(il maggiore). In più dovrà, in primis, capire se giocherà per prima o per seconda nel turno.

Soluzione: Mosquito ha 2 variabili in più, che sono un boolean, che rappresenta se gioca prima o dopo, e due int che contano se ha vinto/perso 3+ volte consecutive per cambiare turno. Il metodo playCard si divide in 2 casi: playFirst e playSecond. Se gioca per prima, la CPU sceglie l'attributo migliore che ha in mano e poi gioca la sua carta passando per il metodo in User con super(). Se gioca per seconda, chiede che attributo ha usato il primo giocatore e gioca la carta con la statistica più alta di quell'attributo. Il metodo getPile, invece, viene usato non solo per salvare la pila delle carte vinte, ma anche per comprendere se giocare ancora per primi, per secondi o cambiare turno tramite la size della List di carte vinte e alle già memorizzate consecWins.

Hovering delle carte

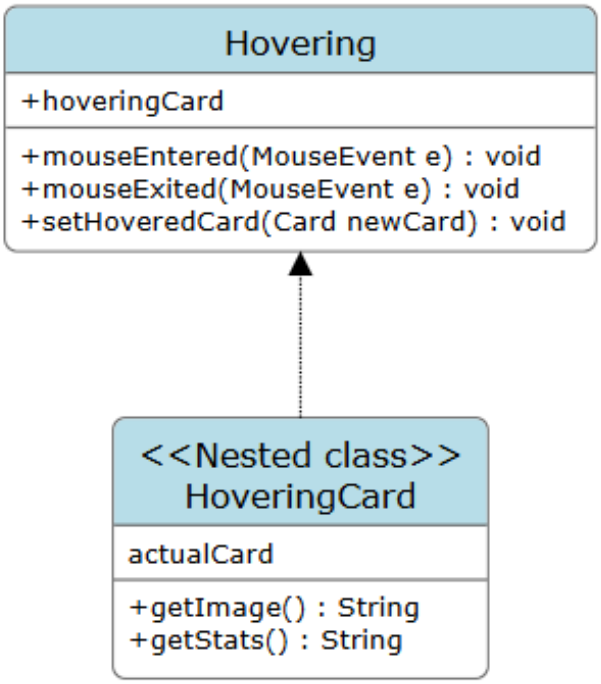

Figura 2.8: UML per la classe Hovering e la sua classe innestata

Problema: In partita, Il giocatore per comprendere le statistiche delle carte che ha in mano, deve poter usufruire dell'Hovering, cioè di un modo per vedere le statistiche della carta scorrendo il cursore su di essa, cosi da avere potere decisionale nel giocare quest'ultima e su che attributo se sta giocando per primo.

Soluzione: Ho creato una classe che implementi l'interfaccia MouseListener, che al suo interno prevede i metodi mosueEntered e mouseExited, i 2 casi importanti per implementare l'Hovering delle carte. In questa nuova classe ho usato anche una classe innestata statica per memorizzare le 3 carte in mano, una diversa dall'altra, e anche un metodo che cambiasse la "Hovering Card" ogni volta che se ne pesca un'altra. Dopodichè, utilizzando questa classe e dei metodi da cui ho ricavato i percorsi delle immagini delle carte, su mouseEntered ho fatto mostrare al gamePanel la carta e le sue statistiche, la cui dimensione dell'immagine varia in base alla dimensione della finestra, e in mouseExited annullo il cambiamento fatto al gamePanel, facendolo ritornare con lo sfondo del background dove precedentemente si veniva mostrata la carta.

2.2.3 Thomas Giovannini

Creazione della logica di gioco

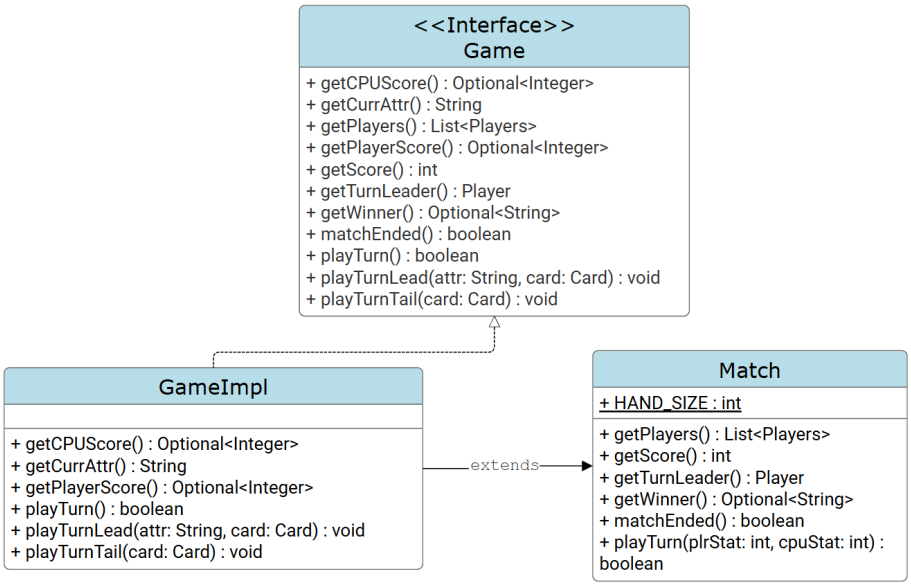

Figura 2.9: UML per la divisione dei compiti tra Game e Match

Problema: Durante la costruzione dell'implementazione della classe GameImpl, se è rivelato necessario separare la logica di decretazione del vincitore di turno e conseguenze dai metodi di gestione delle giocate dei giocatori per poter aderire al principio SRP.

Soluzione: Utilizzando il pattern *Class Adapter*, si è potuto dividere l'intero codice originale di Game in una classe di logica (Match, l'*Adaptee*), e una di gestione delle giocate (GameImpl, l'*Adapter*) che estende quella di logica. Così facendo, si è poi rivelato molto più semplice gestire sia il Controller che gli elementi di View relativi a Game. Inoltre, l'estensione ha permesso sia di evitare ridondanza nella scrittura di metodi necessari ad entrambe le parti o che necessitavano di campi privati in Match, sia di semplificare eventuali modifiche future, in quanto potranno essere fatte direttamente su GameImpl.

Scelta di una modalità di gioco

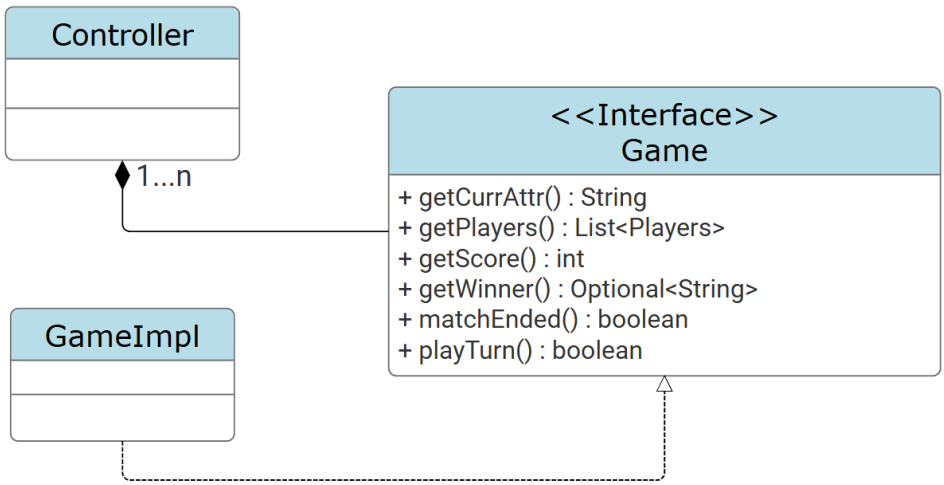

Figura 2.10: UML per la divisione dei compiti tra Game e Match

Problema: Una volta implementata la classe Game, si è notato come in futuro si potrebbero aggiungere diverse modalità di gioco, che avrebbero richiesto multiple classi con metodi simili, quindi si è trovato necessario pensare ad una soluzione per semplificare una nuova implementazione.

Soluzione: La soluzione consisteva chiaramente nel pattern *Strategy*. Creando l'interfaccia Game, si è reso possibile utilizzarla per applicare il pattern in futuro, in particolare nei Controller: ciò permette di implementare un Controller nella stessa maniera per diverse modalità, che potranno essere scelte a piacimento.

2.2.4 Anton Abramov

Creazione Carte e pila punti

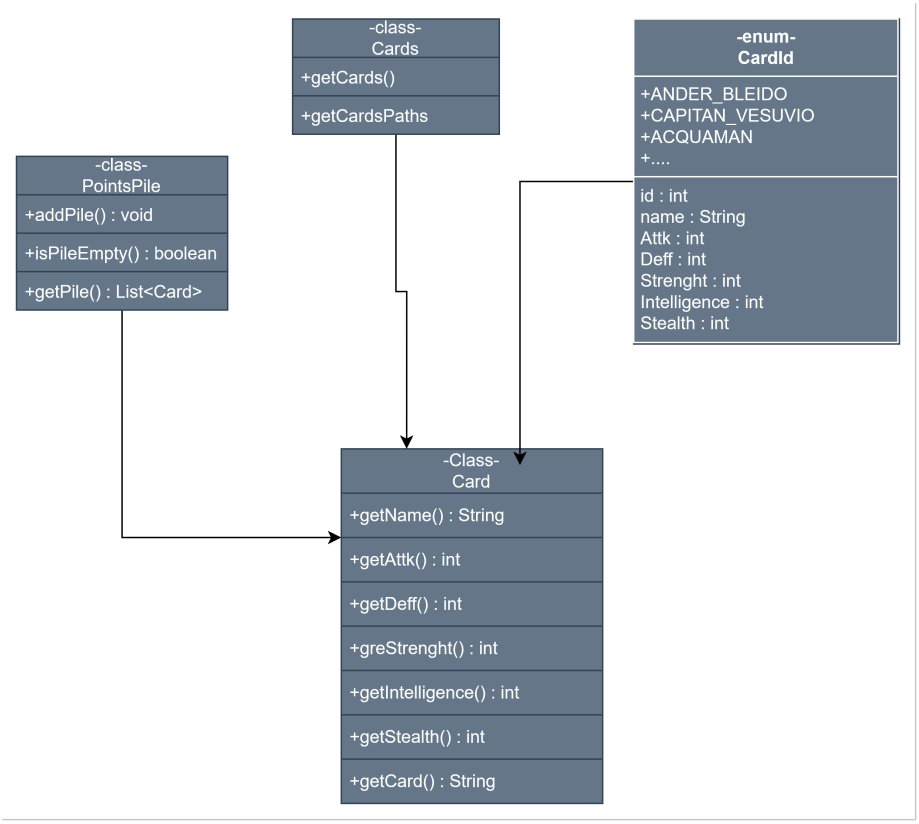

Figura 2.11: UML per la creazione e inizializzazione carte, pila punti

Problema: Per il gioco bisogna creare 24 carte con 5 atributi tutte diverse e una pila dove andranno le carte giocate per diventare in seguito punti assegnati al giocatore, per determinare il vincitore e il totale dei punti ottenuti durante la partita.

Soluzione: Ogni carta è diversa e quindi le sue carateristche vengono segnate in un enum, per semplificare la randoizazione e la gestione delle carte si è optato per una mappa di array, dove vengono definite le carte e cosi da poter richiamare la carta voluta con una chiave che va da 1 a 24 asegnata insieme alle sue caratteristiche nel enum. Le liste sono anche utili per gestire la pila dove vengono messe le carte.

Tabella dei punti

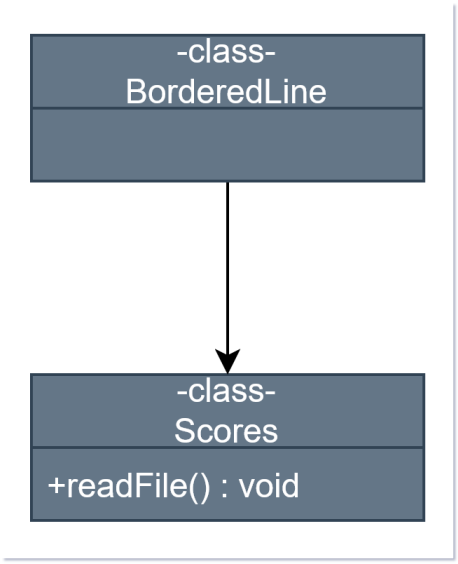

Figura 2.12: UML per la Creare la tabella dei Punti

Problema: Alla fine della partita i punteggi fatti vengono salvati in un fie esterno, che contengono il nome e il numero di punti dei giocatore fatti durante le parttite passate che poi si vuole visualizare nel menu principale.

Soluzione: la classe scores implementa un metodo per poter leggere il contenuto del file esterno, che poi viene modificato per avere un contorno, colori e dimensine personalizate.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per questo progetto si è deciso di implementare JUnit per fare i test delle sue componenti Model principale. In particolare, ciascun membro del gruppo si è dedicato a inserire dei test consoni alle parti del Model di cui erano responsabili e il cui funzionamento era cruciale per la riuscita dell'applicazione. I test, completamente automatici, sono i seguenti:

- **DeckImplTest** - Per testare le funzionalità principali dell'implementazione DeckImpl dell'interfaccia Deck, verifica che il deck non sia vuoto appena inizializzato, che due deck siano uguali prima di fare shuffle, che la size del deck non cambi se mischiato e che il deck sia effettivamente mischiato.
- **StandardCardDrawTest** - Dimostra che l'implementazione StandardCardDrawImpl di CardDraw sia valida, cioè che risulta diversa da un deck solo se rimuove una carta, che questa sia effettivamente rimossa e che l'oggetto rimosso sia una Card.
- **UserTest** - Per testare le funzionalità di User. Viene testato se la mano si riempie troppo, se vengono effettivamente salvate le carte giuste nelle posizioni giuste(ultima carta pescata in ultima posizione) e se vengono giocate le carte giuste con l'attributo scelto.
- **MosquitoTest** - Per testare la correttezza del funzionamento di Mosquito. Avendo già testato le funzioni di base di User, viene testato come la CPU gioca se va per prima, se va per seconda e se effettivamente riesce a riconoscere il suo turno e a giocare seguende le regole.
- **MatchTest** - Per accertare l'integrità di Match. Essendo StandardCardDraw testato a parte, qui ci si concentra sulla capacità di Match di comparare correttamente i valori in entrata, aggiornare in maniera coeren-

te i punteggi, gestire casi di pareggio e cambio del giocatore iniziale, e riconoscere se la partita è finita.

- **GameImplTest** - Per testare le funzionalità principali di GameImpl. Essendo Match, User e Mosquito testati a parte, qui vengono testati l'integrità della costruzione, la capacità di effettuare correttamente le giocate dei Player e utilizzare i loro ritorni per giocare un turno, e la coerenza dell'attributo per la durata di un turno.
- **ControllerImplTest** - Per testare la correttezza del Controller, e in particolare le sue funzioni uniche, si è optato per l'utilizzo della libreria Mockito per utilizzare i mock di View e Model e verificare che le chiamate dei loro metodi non avrebbero generato errori.

3.2 Note di sviluppo

3.2.1 Alessandro Zani

Utilizzo di `Stream` e lambda expressions

Utilizzati parsivamente. Il seguente è un singolo esempio. Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d2684cc93652/src/main/java/migglione/model/impl/Cards.java#L38C4-L40C6`

Utilizzo di `Reflection`

Usata parsivamente. Esempio: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d2684cc93652/src/main/java/migglione/view/impl/scenesimpl/Gallery.java#L59C5-L75C6`

Utilizzo di `Method reference`

Usata in vari punti. Permalink di esempio: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d2684cc93652/src/test/java/migglione/model/DeckImplTest.java#L42C5-L62C6`

Utilizzo di `@FunctionalInterface`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d26src/main/java/migglione/view/api/music/MusicProvider.java#L17-L26C2`

Utilizzo della libreria `Sound` e suoi delegati

Principalmente usata in questo frangente: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d2684cc93652/src/main/java/migglione/view/impl/musicimpl/LoopingMusicPlayerImpl.java#L41C5-L57`

Utilizzo della libreria `Mockito`

Usata per testare Controller: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d2684cc93652/src/test/java/migglione/controller/impl/ControllerImplTest.java#L38C1-L94C2`

Utilizzo di `Optional`

Usato parsivamente: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d2684cc93652/src/main/java/migglione/controller/impl/ControllerImpl.java#L145C5-L159C6`

Utilizzo di `Logger` **di java**

Usato parsivamente: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d2684cc93652/src/main/java/migglione/persistence/impl/ScoreRepositoryImpl.java#L23C1-L71C2`

3.2.2 Lorenzo Zanoni

Utilizzo di `Reflection`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d268`
`src/main/java/migglione/view/impl/scenesimpl/Field.java#L256C9-L257C105`

Utilizzo di `Inheritance`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/08034ab84e09349253ca9448d870d268`
`src/main/java/migglione/model/impl/Mosquito.java#L26-L33`

Utilizzo di `Method reference`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/feb442902baad5b3d1e28e3aa99f2956`
`src/main/java/migglione/view/impl/scenesimpl/Tutorial.java#L46`

3.2.3 Thomas Giovannini

Utilizzo di `Stream` **e** `lambda expressions`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/dc4b7ff46c34132ae325ca07bd70a940`
`src/main/java/migglione/view/impl/Field.java#L99`

Utilizzo di `Optional`

Utilizzata in multiple occasioni. Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/dc4b7ff46c34132ae325ca07bd70a940c882d036/src/main/java/migglione/view/impl/Field.java#L214`

Utilizzo di `Reflection`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blob/dc4b7ff46c34132ae325ca07bd70a940`
`src/main/java/migglione/view/impl/Field.java#L99`

Utilizzo di `instanceof`

Utilizzato parsivamente. Permalink: `https://github.com/Megabrodo/OOP25-migglione/`
`blob/dc4b7ff46c34132ae325ca07bd70a940c882d036/src/main/java/migglione/`
`view/impl/Field.java#L399`

3.2.4 Anton Abramov

Utilizzo di `Rendering`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d26`
`src/main/java/migglione/view/impl/scenesimpl/BorderedLine.java#L57C9-L58C112`

Utilizzo di `Reflection`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d26`
`src/main/java/migglione/view/impl/scenesimpl/Scores.java#L57C9-L58C93`

Utilizzo di `lambda expressions`

Permalink: `https://github.com/Megabrodo/OOP25-migglione/blame/08034ab84e09349253ca9448d870d26`
`src/main/java/migglione/view/impl/scenesimpl/Scores.java#L61C9-L63C26`

Note generali

Data la difficoltà e l'uso di numerosi costrutti non visti a lezione, si è ritenuto necessario consultare varie fonti online e in particolar modo forum e API, tra questi ritroviamo: StackOverflow, CodeRanch, TutorialsPoint, GeeksForGeeks, Mockito.org e ovviamente Oracle.com per le API di java. Nonostante si sono visitati questi siti come spunto per il nostro lavoro, è giusto avvertire che certe implementazioni sono per questo motivo simili ad alcune che si potrebbero trovare in tali siti, anche se, come è giusto che sia, queste sono state reinventate e riutilizzate per il nostro progetto. In conclusione, nonostante non si sia fatto in nessun caso "copia e incolla", potrebbero esserci delle somiglianze non volute.

Capitolo 4

Commenti Finali

4.1 Autovalutazione e lavori futuri

4.1.1 Alessandro Zani

Trovo che questo progetto mi abbia fatto capire quanto la difficoltà di questa professione scala all'aumentare del numero di persone che ci stanno dietro, e ho trovato che anche con il supporto dei miei colleghi, che nonostante le tempistiche dovute al periodo d'esami hanno saputo accettare la mia visione su certe parti del progetto e applicare le modifiche necessarie nel minor tempo possibile, il risultato finale non mi convince al massimo. Sarà proprio perché la maggior parte di noi ha dovuto iniziare a lavorarci tardi, ma sono sicuro che se avessimo avuto un monte ore massimo individuale più alto avremmo potuto portare migliorie non indifferenti.
Essendo uno dei creatori originali del gioco mi sono sentito in dovere di provvedere a tutto il comparto grafico, stilistico e progettistico dell'applicazione, dopotutto mi sentivo come di aver imposto la mia idea sugli altri; per cui, mentre i miei colleghi non erano disponibili ho cominciato con lo strutturare la GUI e mi sono messo in contatto con un altro amico creatore della prima versione del gioco per revisionare gli sprite delle carte, mi sono anche predisposto a discutere con tutti l'assegnamento di parti eque del progetto nella sua interezza. Quindi, anche se certe persone hanno fatto più elementi di GUI che di Model, nel complesso spero di aver saputo affidare a tutti una mole di lavoro imparziale e giusta. Già da tempo avevo vagliato l'idea di trasformare Il Migglione in una applicazione, quindi finito questo progetto mi piacerebbe utilizzare parte del mio tempo libero, e magari software e linguaggi più adeguati, per portarlo avanti e, forse, un giorno rilasciarlo come gioco indipendente, giusto per concludere una vecchia idea di tre studenti delle superiori.
Nel complesso, giudico il mio lavoro e quello dei miei colleghi molto più che sufficiente e conforme al monte di ore indicato.

4.1.2 Lorenzo Zanoni

Inizio col dire che analizzo il mio lavoro sotto una lente specifica: purtroppo questo semestre è stato quantomeno provante a livello fisico e mentale, dunque il lavoro è stato iniziato(a mio parere) tardi e con vari imprevisti e complicazioni date dai numerosi esami da dover passare. Detto questo, trovo che sia stata comunque una esperienza formativa molto positiva: lavorare in sincronia con gli altri membri mi ha fatto sentire effettivamente parte di una squadra con un obiettivo chiaro e preciso sin dai primi passi, e questo mi ha motivato molto nella scrittura del programma. In più le discussioni sulle specifiche del gioco e su come implementarle mi hanno veramente fatto capire quanto, in un ambiente sano con collaboratori seri e precisi, si possa creare con l'aiuto delle altre persone rispetto che da soli.
Personalmente, ritengo di essermi messo in gioco sin dal primo momento, mettendomi sempre a disposizione per attuare modifiche al codice per aiutare i miei collaboratori e per dare consigli e discutere costruttivamente sul progetto. Oltretutto ho anche accettato un incarico ulteriore alla mia parte obbligatoria, cioè la creazione del tutorial che grazie ad Alessandro Zani è stata implementata anche con un pop-up al primo start dell'applicazione.
Detto questo, in futuro vorrei cimentarmi nella creazioni di altri giochi(sono molto appassionato di RPG d'avventura), magari proprio concentrandomi sul lato CPU di essi, essendo il singleplayer una componente che nei giochi moderni vedo andare a scemare e io che sono cresciuto con giochi principalmente a giocatore singolo vorrei riportarli a come me li ricordo, almeno per quanto possa fare con chi deciderà di supportarmi.
In definitiva, reputo il lavoro mio e dei miei colleghi più che sufficiente, oltre che la spartizione del lavoro equa e un monte ore utilizzato saggiamente e non sforato di quantità proibitive.

4.1.3 Thomas Giovannini

Non avendo mai partecipato ad un progetto di questa portata, non sapevo bene cosa aspettarmi, però mi sono presto reso conto di aver sottovalutato la mole di lavoro necessaria a completare tutto il gioco nel tempo stabilito. Avendo scelto di consegnare a fine sessione invernale, ci siamo ritrovati in delle situazioni poco comode a causa della sovrapposizione tra progetto ed esami, che credo abbia influito (almeno, per me) in maniera non inidfferente. A livello educativo, però, ritengo che questa sia stata un'esperienza molto produttiva: mi ha permesso di imparare a collaborare con più persone per periodi lunghi di tempo, e, nonostante varie discussioni su alcuni punti delle varie implementazioni, mantenere rapporti sani con i miei colleghi, oltre che ad applicare in maniera efficace le mie conoscenze su Java.
Non avendo conosciuto il gioco prima di essere stato proposto, mi sono concentrato più su quello che effettivamente sapevo del gioco, anche se ciò ha causato più di un litigio a causa di visioni diverse sulla stessa implementazione. Inoltre, ho notato che la pressione dovuta all'avvicinarsi della deadline ha causato

più distrazioni del solito, obbligandomi a lavorare sullo stesso segmento per più tempo del necessario. In generale, però ritengo di aver svolto un buon lavoro, anche se con un gran margine di miglioramento.
Nonostante le varie peripezie durante la costruzione del progetto, ritengo che il risultato finale sia coerente con le aspettative, anche se non prefettame come me lo aspettavo. Se si potesse proseguire, con un monte ore più alto, sarei interessato ad aiutare a portare avanti questo progetto, in quanto, anche se prima non conoscevo l'idea alla base, è un gioco con un grande potenziale, e credo possa essere molto interessante produrre nuove modalità di gioco.

4.1.4 Anton Abramov

Il proggetto in gruppo mi ha fatto solo capire quanto realmento non so nulla, la mia lentezza e poca capacità di gestire esami/studio con il lavoro in squadra è stato deleterio e poco efficace, non posso essere sodisffatto del mio lavoro perché ritengo che quel poco che sono riuscito a fare, per colpa mia, non sia stato per nulla d'impatto.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Alessandro Zani

Nell'avvicinarsi alla deadline del compito mi sono reso conto che questa parte di esame ha dei problemi non indifferenti, alcuni che sono da individuare nel corso e altri nell'esame stesso. Per iniziare, penso che la quantità di elementi sviluppati e approfonditi durante il corso non è in nessun modo adeguata ad un progetto di questo tipo; cioè, la base della programmazione e tutto ciò che la riguarda è stata affrontata in maniera giusta e non ho lamentele al riguardo, ma tutto quello che dai professori è stato ritenuto **necessario** applicare nel progetto è stato visto di sfuggita e non gli è stata data in nessun modo la giusta importanza. Per essere specifici, come creare UML soddisfacenti e appropriati, i Methods e come si applicano in un progetto, la struttura MVC e cosa comporta su applicazioni non di cinque classi in totale e l'uso di librerie esterne è stato tutto approfondito in poche lezioni, e in laboratorio non sono state praticamente applicate.

Gli esercizi di MVC fatti in laboratorio non sono lontanamente paragonabili al livello richiesto nel progetto e hanno generato problemi soprattutto nella comprensione dei ruoli più astratti come il Controller, di ciò che deve e non deve fare. Durante la realizzazione di questo progetto ho infatti speso circa un quarto del tempo alla sola ricerca di informazioni su come relazionare le classi tra di loro e applicare i Methods spiegati nelle due lezioni di teoria al riguardo, scoprendo che molti pattern che non abbiamo visto, come Facade, li avrei implementati per sbaglio. Ciò ovviamente non per dire che andrebbero visti tutti i

pattern, ma che far usare effettivamente quelli più importanti nelle esercitazioni permetterebbe di saper affrontare i problemi con già il pattern giusto da usare in testa. A questo proposito, l'esame pratico non è assolutamente vagliato su tutto il programma, ma si concentra a far programmare seguendo JUnit con dei Methods precisi, di cui l'unico possibile metodo di studio è rappresentato dal completare esami passati, e il rapporto tra GUI e Logic, per cui vale la stessa cosa. Quello che intendo dire, è che se uno sapesse già le basi di Java potrebbe tranquillamente fare entrambi gli esami del corso, perché per il primo basta esercitarsi su cose di anni passati e il secondo ti chiede comunque di passare la maggior parte del tempo a ricercare metodi per fare quello che ti serve.
Se questo è l'obiettivo allora va benissimo così, trattate questa parte come un elogio, ma dal punto di vista dello studente non è bellissimo finire il corso sapendo che si è rischiato più volte il burnout dalla mole di lavoro da fare. Infatti, considerando come si è pienamente in sessione nella realizzazione del progetto, è inevitabile che la possibilità di arrivare ai ferri corti con la deadline sia molto alta, dopotutto ogni studente deve preparare come **minimo** (senza contare l'orale opzionale di MDP) quattro esami nell'arco di un mese e mezzo, ponendo in tutto questo circa settanta ore di progetto di gruppo.

In conclusione, ritengo che si potrebbe utilizzare parte del tempo di laboratorio per implementare e solidificare quello fatto a lezione con compiti un po' più espansivi, soprattutto nella parte finale del corso e, similmente, far sì che a lezione si approfondiscano pienamente gli argomenti richiesti per i due esami, con esempi passati magari, giusto per comunicare correttamente allo studente la differenza tra un progetto da trenta e uno da diciotto.

4.2.2 Lorenzo Zanoni

Durante la scrittura del codice, mi sono reso conto che molti dei meccanismi fatti a lezioni non sono riuscito a implementarli: vada per la inheritance e gli advanced mechanism, che abbiamo visto ampiamente non solo a lezione ma anche in laboratorio, ma per i pattern ad esempio, avendo iniziato il lavoro tardi data la sessione molto complicata, mi sono trovato a non essere assolutamente in grado di creare un effettivo pattern come builder o factory(anche se magari non sarebbero state opzioni efficienti), il che ho pensato mi avrebbe richiesto del tempo per imparare e per scrivere efficacemente.
Questo, da uno studente che ha frequentato tutte le lezioni(a parte alcune per motivi di salute), non me lo aspettavo, perchè arrivato a voler quantomeno provare a usarli, ho consultato i miei appunti e non trovavo soluzioni, ho consultato le slide del corso e ugualmente non sono riuscito a pensare a un modo efficace di implementarli. Potrebbe essere un errore dalla mia parte, cioè che come ho detto ho iniziato assieme ai miei colleghi tardi a fare il progetto e la deadline era un obiettivo prioritario su tutto, però è anche vero a mio modo di vedere che se sono riuscito senza alcun problema ad implementare nested class e a sfruttare l'ereditarietà, cose che abbiamo visto in laboratorio, mi è sembrato strano che i pattern non sia riuscito in alcun modo ad integrarli.

Ponendo poi l'attenzione al fatto che una persona è pregata di attenersi a circa 70 ore di lavoro, diventa complesso organizzare sessione, ore effettive di lavoro e codice funzionante. Oltre a ciò, per il mio lavoro sulla GUI ho dovuto consultare più volte la documentazione di Java 21 per comprendere metodi che ritengo essere fondamentali per il progetto richiesto(ad esempio per come mostrare un immagine a schermo durante l'Hovering).
Aggiungo in ultima battuta: la sessione è un ostacolo ben più imponente di quello che può sembrare, anche solo il fatto che, rispetto ad esempio alla seconda sessione del primo anno, c'è un esame in più in 1 mese e mezzo in meno di tempo, è una complicazione non indifferente. In più un progetto di questa dimensione, fatto in gruppi da 4 in cui ogni componente ha un suo piano per la sessione che può avere intoppi e rallentarne la scrittura, è una ricetta che potrei comprendere perchè molto spesso gli studenti decidono di farlo durante le lezioni del secondo semestre per poi consegnarlo in seconda sessione, ma non credo sia un buon metodo da usare in un corso di studi come questo.

Concludo col chiedere per i futuri studenti che si implementino ulteriori esercitazioni per gli argomenti di teoria che si introducono durante le ultime settimane del corso, e che si organizzi l'esame in modo da potergli effettivamente trovare spazio in una sessione che già di per sé mette molto in difficoltà qualunque studente la affronta.

4.2.3 Thomas Giovannini

Durante la costruzione delle mie classi, ho notato che cercare di applicare i pattern di design, a parte Strategy, è risultato molto più difficile del previsto, in quanto a lezione ci si è concentrati per la maggior parte sulla struttura esterna, e non si è scesi molto in dettaglio su come convergere verso un pattern. Con Adapter sono riuscito a creare un model più pulito, ma, ad esempio, per la mia parte di View non sono riuscito a muovermi verso un qualunque pattern. Credo che, se venisse dato più tempo al riconoscimento di pattern, anche con esercizi di laboratorio dediti a portare da "pattern-less" ad un pattern specifico, il design diventerebbe molto più gestibile dagli studenti.
Anche se è vero che ho sottovalutato la mole di lavoro, ho notato che molto spesso ho dovuto cercare metodi o classi che risolvevano un problema al di fuori delle risorse del corso, il che può essere coerente con il fatto che un programmatore non avrà mai tutto a disposizione, ma dovrà cercare se altri hanno già risolto il proprio problema. Però, mi sono reso conto di non sapere nemmeno come iniziare a cercare, in quanto durante il corso ci si è concentrati più su come programmare con gli strumenti a propria disposizione. In un certo senso è un fattore positivo per me, in quanto mi piace sperimentare molto quando so quello che sto facendo, ma ritengo che fornire agli studenti almeno un metodo di ricerca possa aumentare il tempo a programmare e ridurre considerevolmente quello speso a cercare come programmare una sezione di codice.
Infine, vorrei suggerire di proporre risoluzioni di alcuni esercizi di laboratorio in presenza, in quanto mi sono trovato ad imparare la logica dietro alla maggior

parte dei concetti imparati autonomemente, ma per altri mi sono trovato più in difficoltà, avendo una soluzione davanti ma non come arrivarci.

Appendice A

Esercitazioni di laboratorio

A.0.1 alessandro.zani3@studio.unibo.it

- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=206731#p283771
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p284650
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=207921#p285866
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p286859
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=209589#p288441
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289280
- Laboratorio 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=211539#p290163

A.0.2 lorenzo.zanoni3@studio.unibo.it

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p285065
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287248
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=209589#p288515

- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289537
- Laboratorio 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=211539#p290346

A.0.3 thomas.giovannini@studio.unibo.it

- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=206731#p284114
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p284966
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=207921#p286081
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287249
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=209589#p288384
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289684
- Laboratorio 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=211539#p290667