BECCACCINO

Relazione progetto programmazione ad oggetti

A.A. 2018/2019
Deadline B: 25/04/19

Davide Alpi	0000831238
Riccardo Berti	0000826433
Francesco Foschini	0000826200
Alessia Rocco	0000830542

Indice

1 ANALISI 3
1.1 REQUISITI 3
1.2 ANALISI E MODELLO DEL DOMINIO 4
2 DESIGN 5
2.1 ARCHITETTURA 5
2.2 DESIGN DETTAGLIATO 6
3 SVILUPPO 13
3.1 TESTING AUTOMATIZZATO 13
3.2 METODOLOGIA DI LAVORO 13
3.3 NOTE DI SVILUPPO 13
4 COMMENTI FINALI 15
4.1 AUTOVALUTAZIONE E LAVORI FUTURI 15
4.2 DIFFICOLTA' INCONTRATE E COMMENTI PER I DOCENTI 16
A GUIDA UTENTE 17

CAPITOLO 1: ANALISI

L'obiettivo è creare una versione digitale del gioco di carte romagnolo Beccaccino/Marafone.
Il Marafone/Beccaccino è un gioco di carte per quattro giocatori che utilizza un mazzo da 40 carte da gioco italiane.
I quattro giocatori sono divisi in due squadre contrapposte. La squadra vincitrice è quella che totalizza il maggior numero di punti.
I punti si accumulano vincendo i vari round. Quando un giocatore vince un round la sua squadra ottiene tutte le carte giocate in quel round, e ad ogni carta è associato un punteggio.
Ogni giocatore possiede 10 carte in mano fin dall'inizio e ad ogni round dovrà giocarne una. All'inizio di ogni match il primo giocatore battezza la briscola (seme dominante) che rimarrà tale per tutti e 10 i round. Il primo giocatore di ogni round è il vincitore del round precedente. Il primo round della partita lo inizia il giocatore che tiene in mano il quattro di denari. I giocatori successivi devono rispondere giocando carte dello stesso seme di quella giocata dal primo (se ne hanno, altrimenti una qualsiasi). Il primo giocatore del round può anche associare alla propria giocata un messaggio tra "striscio, busso o volo" in modo da fare capire al proprio compagno di squadra la propria situazione. Con il messaggio "Busso" si vuole dire al compagno di prendere e rigiocare una carta al round successivo sullo stesso seme, con il messaggio "Striscio" si indica al compagno di avere molte carte del seme giocato, con il messaggio "Volo", infine, si vuole aggiornare il compagno che la carta giocata è l'ultima di quel seme che si possiede.
Il gioco creato è una versione semplificata del gioco ufficiale, che prevede una serie di partite fino al raggiungimento di 41 punti. Per maggiori informazioni consultare le regole definite da https://it.wikipedia.org/wiki/Marafone_Beccacino.

1.1 REQUISITI

REQUISITI FUNZIONALI

- Presenza di un menù di gioco che permette di cambiare le impostazioni di base, creare nuovi profili giocatore e di avviare la partita.
- Una partita è effettuata con un singolo giocatore e 3 intelligenze artificiali molto semplici.
- Durante la partita è possibile mandare i messaggi di base del gioco ("busso", "striscio", "volo")
- Alla fine della partita sarà visualizzato il resoconto di fine partita con relativi punteggi delle squadre.

REQUISITI FUNZIONALI OPZIONALI

- Introduzione di IA più avanzate.
- Possibilità di modificare il set di regole in uso.
- Possibilità di modificare varie impostazioni del gioco, come ad esempio:
    - ordinamento delle carte nella propria mano
    - musica di sottofondo
- Possibilità di effettuare una partita tra più giocatori umani sulla stessa macchina, quindi con gestione dei turni intelligente che fa in modo di nascondere le informazioni sulla mano del giocatore precedente e possibilità di impostare un limite di tempo per il turno di ogni giocatore.
- Possibilità di creare profili utente, con relative statistiche.

REQUISITI NON FUNZIONALI

- L'esperienza dell'utente dovrà essere il più fluida e piacevole possibile.

- Le specifiche tecniche parlano della possibilità di funzionare su dispositivi con diversi sistemi operativi. Portabile quindi nei sistemi operativi Windows e Linux.

1.2 ANALISI E MODELLO DEL DOMINIO

Le entità base individuate nel dominio applicativo sono:
- le carte, che posseggono un valore e un seme;
- il mazzo di carte, formato da 40 carte;
- le mani, che possono tenere un numero di carte che muta nel proseguo della partita;
- i giocatori, che hanno un nome identificativo e si compongono di una mano;
- i team, che nel caso del beccaccino sono formati da 2 giocatori;
- le giocate, che consistono in 1 carta + un eventuale messaggio.

La partita deve modificare le precedenti entità al susseguirsi dei turni.
Ad ogni giocatore può essere associata un'intelligenza artificiale, che produrrà le giocate necessarie al proseguo della partita, o un utente umano.
L'applicazione dovrà avere un menù che gestisca le impostazioni e l'avvio di nuove partite.

Gli elementi che costituiscono il problema sono sintetizzati in Figura 1.1.

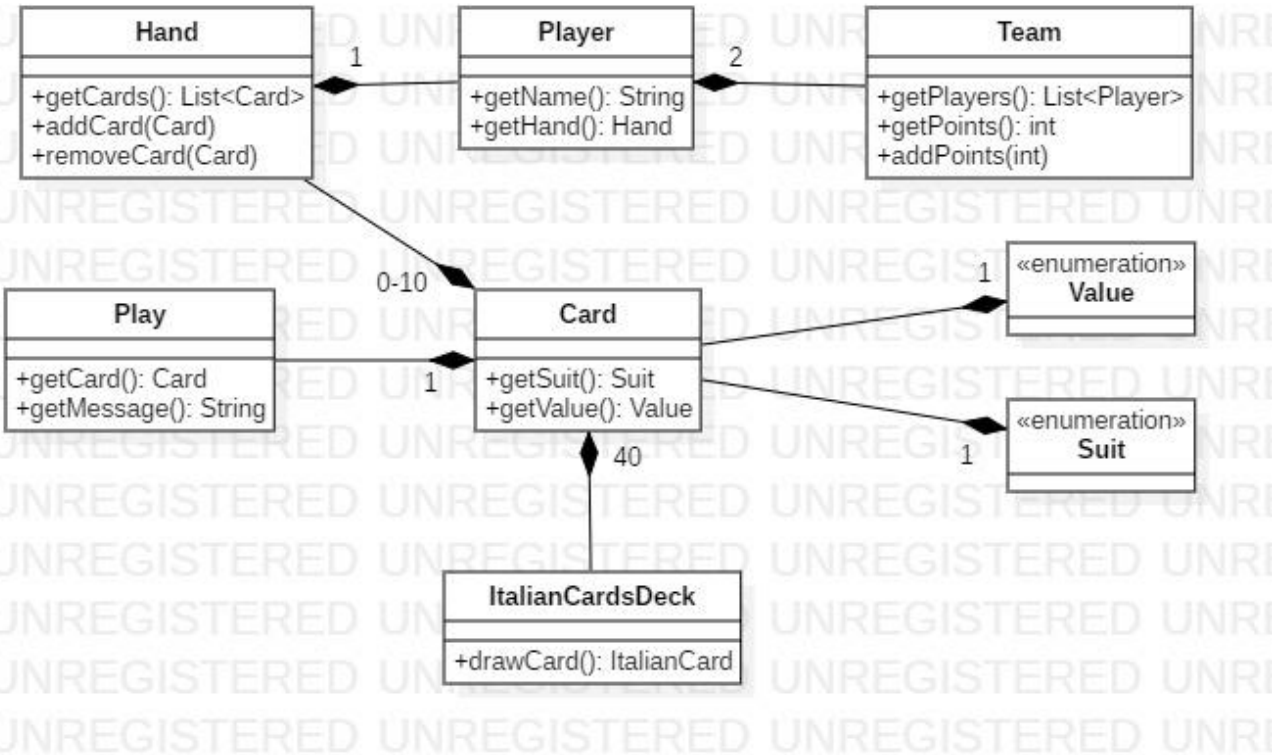

Figura 1.1: schema UML dell'analisi del problema, con rappresentate le entità principali e i rapporti fra loro.

Nel precedente e nei successivi schemi UML verranno riportati solamente i metodi e le interfacce più rilevanti alla comprensione del progetto.

CAPITOLO 2: DESIGN

2.1 ARCHITETTURA

L'architettura di Beccaccino segue il pattern MVC. Ciò consente di suddividere la logica funzionale in:

- **Model**: che gestisce la logica relativa al dominio dell'applicazione.
- **View**: che gestisce l'interfaccia grafica e la relativa interazione con l'utente.
- **Controller**: che gestisce l'interazione tra gli avvenimenti della View, intercettandoli, e mandando le necessarie modifiche al Model, nel caso in cui sia necessario il Controller notificherà nuovamente la View.

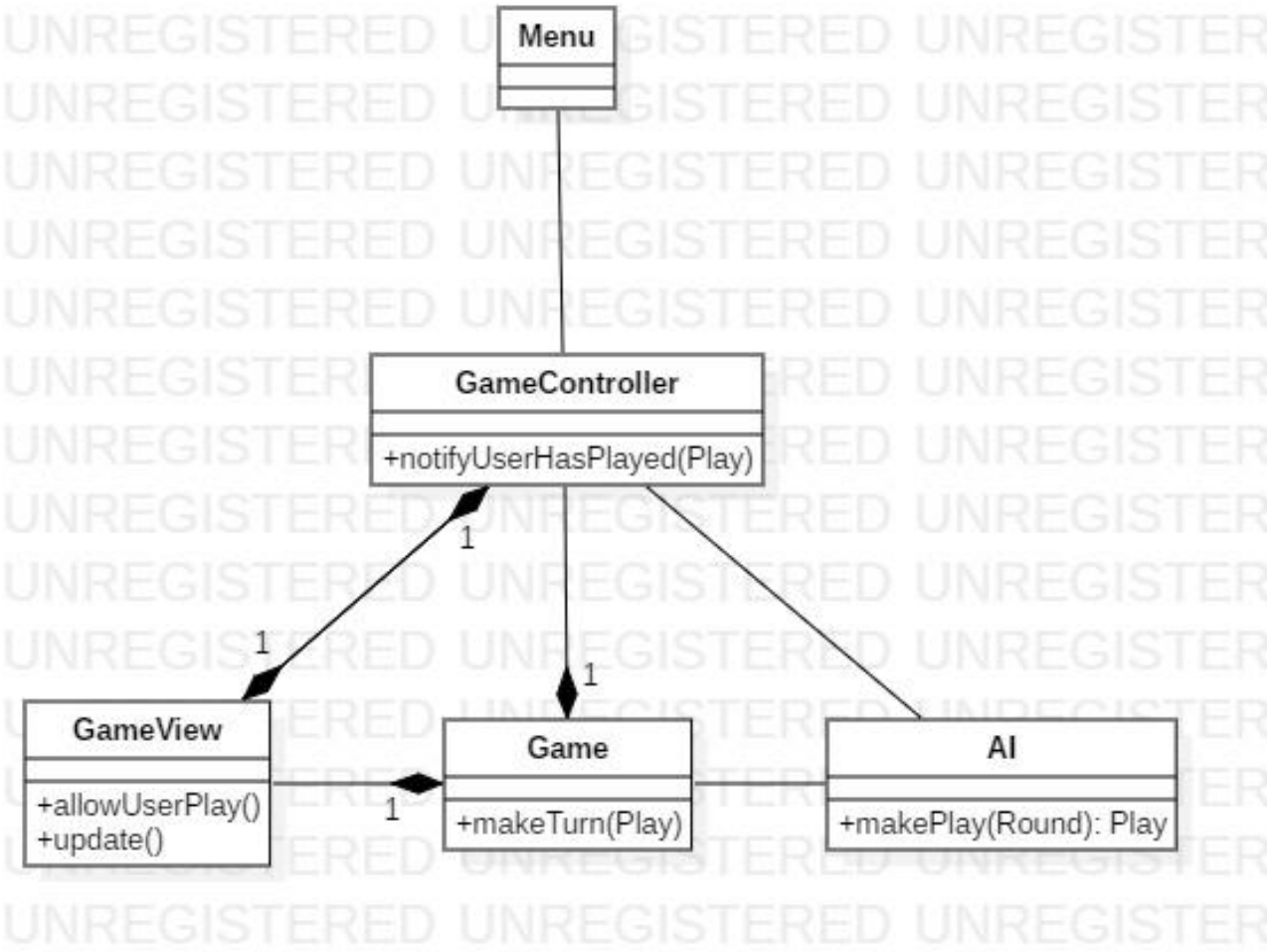

Figura 2.1: diagramma UML dell'architettura MVC. L'interfaccia GameController è il controller del sistema, mentre GameView si occupa delle interfacce che mappano la view. L'interfaccia AI invece rappresenta l'intelligenza artificiale, che quindi comunica con il controller come la view senza avere però interfacce grafiche dirette.

In particolare, come si può vedere dal diagramma, il pattern MVC è stato applicato al concetto di partita. Questa scelta rende possibile usare come view qualsiasi classe che implementi l'interfaccia GameView senza dover apportare alcuna modifica alla parte di Model o di Controller.
Per ogni turno il GameController chiederà all'entità associata al giocatore corrente (AI o view se è il turno dell'user) la giocata da effettuare. Tale giocata verrà poi passata al Game che si occuperà di processarla correttamente. Il procedimento si ripete finché il Game non è terminato.

Al termine di ogni turno il GameController comunica alla GameView di aggiornarsi.
Viceversa, quando è il turno dell'user, è la GameView a notificare il GameController quando la giocata è pronta.
Il Model è stato costruito generalizzando il più possibile, tramite interfacce, certi concetti per favorire il riuso e l'estendibilità del codice.
L'istanziazione delle classi è stata affidata ad una classe Ruleset (AbstractFactory), che, in base alle impostazioni selezionate dall'utente, crea e restituisce le giuste classi concrete.
Questa fa in modo che l'applicazione risulti molto aperta dal punto di vista di piccole modifiche al regolamento.
Il menù si occupa del cambiamento delle impostazioni e del corretto lancio di una partita (GameController).

2.2 DESIGN DETTAGLIATO

Davide Alpi
Mi sono occupato principalmente della realizzazione della classe Game.

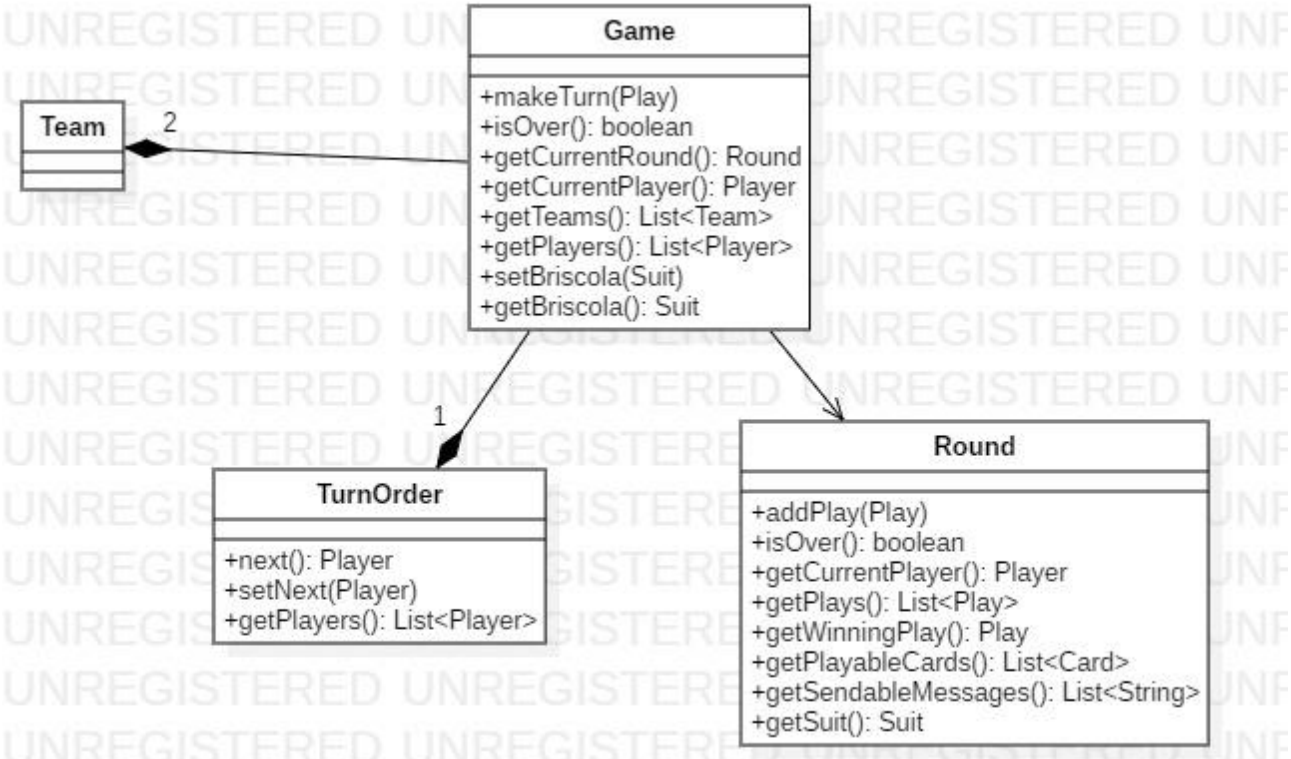

Figura 2.2: diagramma UML del design del Game tramite interfacce.

Il Game si compone di un TurnOrder, che gestisce in che modo i giocatori debbano susseguirsi nell'effettuare le proprie giocate.
Il game necessita alla creazione di 2 Team.
I Round vengono creati dal Game a mano a mano che la partita prosegue e incapsulano la logica che determina quale tra le giocate inserite in quel Round dev'essere dichiarata vincitrice.
Sia per l'implementazione di Game che di Round è stato fatto largo uso del pattern TemplateMethod. La logica di una partita e di un Round è stata spezzettata in molteplici metodi relativi a specifiche parti. Le classi astratte GameTemplate e RoundTemplate sono abbastanza generali da poter essere

ereditate da classi che potrebbero implementare altri giochi di carte a turni come Briscola, Tressette, Scopa, ecc.
Di seguito il diagramma UML di quanto appena spiegato.

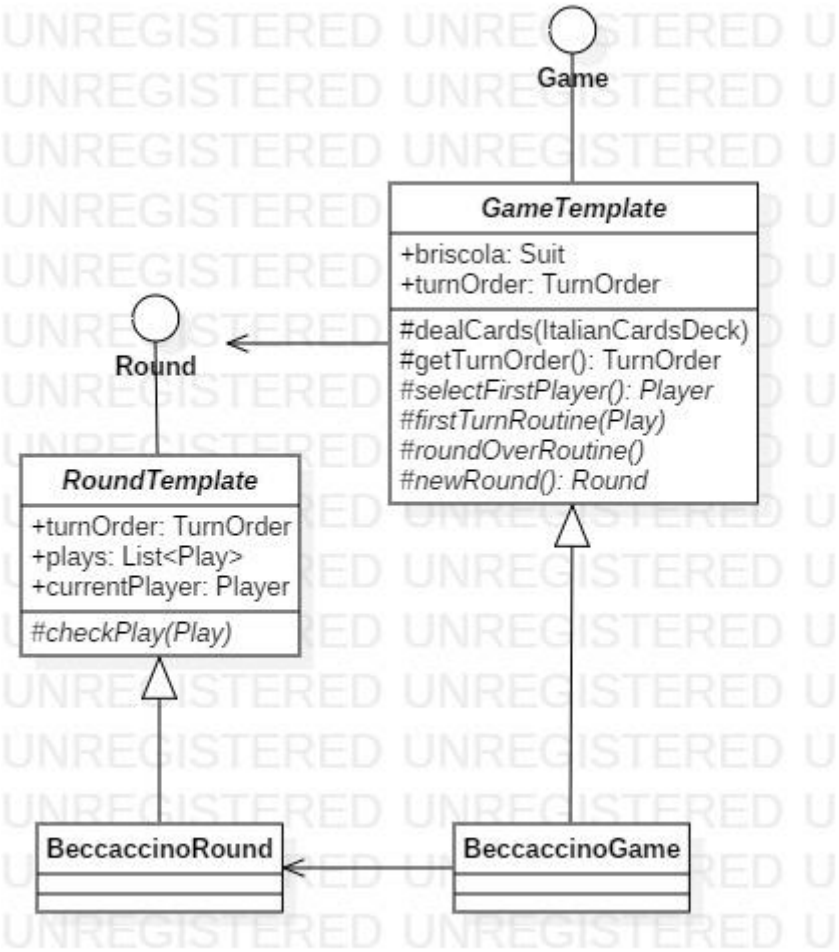

Figura 2.3: diagramma UML della realizzazione delle interfacce Game e Round.

Si noti l'uso del FactoryMethod newRound() che consente alle classi che estendono GameTemplate di produrre giuste versioni di Round (ad esempio, BeccaccinoGame implementa il metodo newRound() facendogli creare istanze di BeccaccinoRound).

Per l'implementazione della classe Hand è stato seguito lo stesso pattern del Game e del Round, producendo quindi un HandTemplate e una BeccaccinoHand, con la differenza che è stato utilizzato il template method solamente per sapere se la mano è piena (concetto che effettivamente dipende dal gioco a cui si sta giocando). Nonostante l'estrema semplicità della classe, questo design è stato comunque adottato per minimizzare la ripetizione di codice nel caso si vogliano introdurre altri comportamenti in futuro.

Riccardo Berti
La mia parte di progetto era incentrata su tutti gli aspetti inerenti ai menu, alla gestione di profili, settings e, nello specifico, all'implementazione dei concetti di carta, deck e ruleset.
Siccome i menu sono fondamentalmente statici a livello grafico, in fase di design ho preso la decisione di creare le scene necessarie con software esterni (in questo caso Scene Builder) per facilitare l'operazione, eventualmente aggiungendo modifiche quando necessario.

Per quanto riguarda l'aspetto control, ogni menu è controllato individualmente in quanto non hanno nulla in comune l'uno con l'altro.
Il concetto di profilo è stato pensato per permettere di memorizzare informazioni relative a un giocatore specifico al di fuori di singole partite e per facilitarne la distinzione nel caso il gioco venga esteso per permettere partite a più giocatori umani.
Ho deciso di impedire l'inizio di una partita fino alla creazione di almeno un profilo, per sfruttare al meglio la memorizzazione di informazioni, anche se questa non è stata implementata nella versione corrente.
Anche i settings selezionati dall'utente vengono memorizzati in modo permanente tramite file per evitare di doverli settare ad ogni aperura dell'applicazione.
I concetti di carta e deck sono stati implementati favorendo l'espandibilità e il possibile riuso in altre simili applicazioni, creando interfacce universalmente utilizzabili nel campo dei giochi di carte e implementandole specificamente per il Beccaccino.
Ruleset era stato inizialmente ideato come oggetto in grado di memorizzare opzioni e alcuni parametri utili, ma si è poi evoluto in una Factory creatrice di tutti gli oggetti necessari a una partita, facendo ciò su base delle preferenze dell'utente.
In quanto non dipendente dalle specifiche dell'applicazione, anch'esso è espandibile e riutilizzabile.

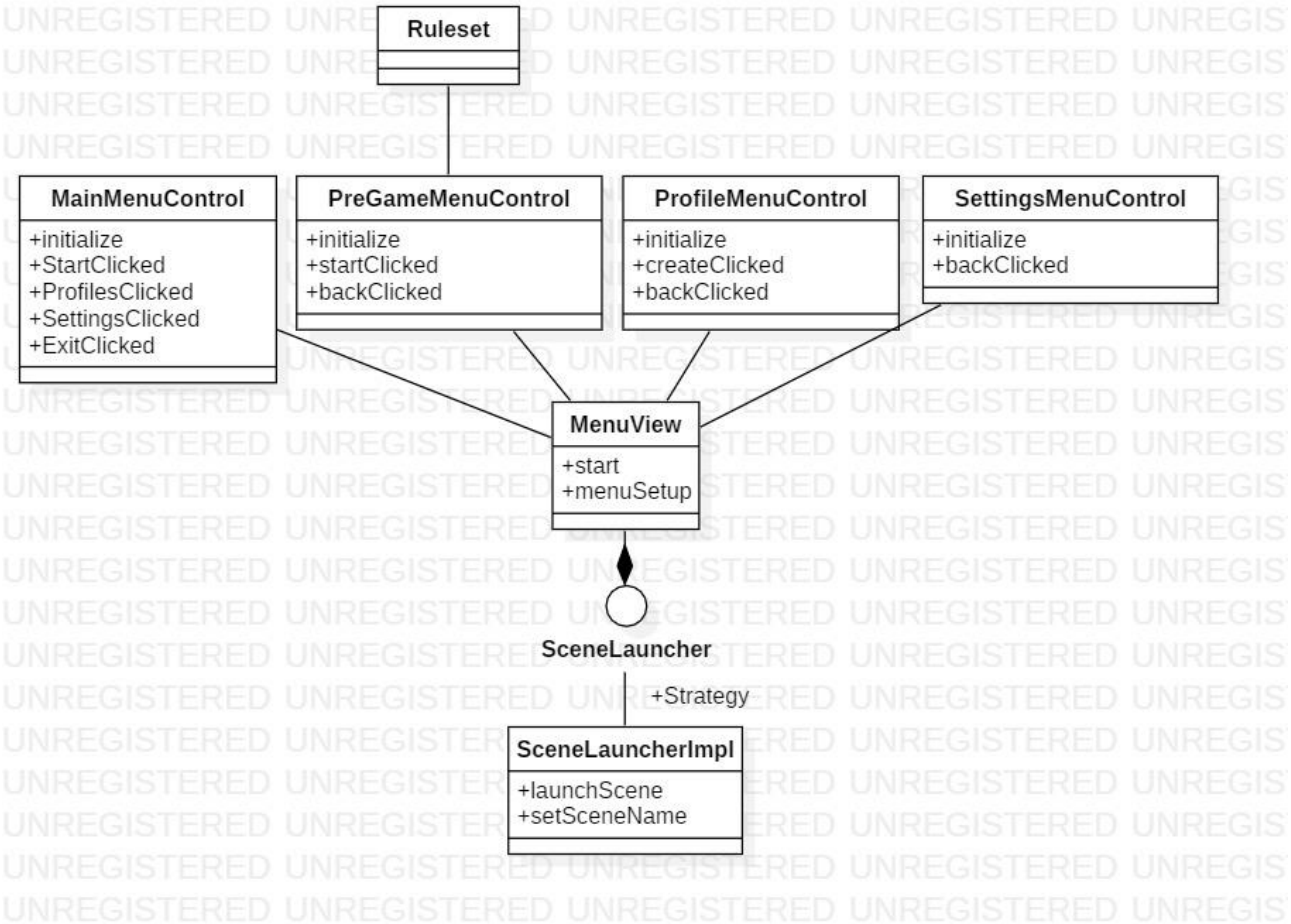

Figura 2.4: diagramma UML rappresentante l'aspetto MVC dei Menu, con collegamenti a RuleSet e SceneLauncher.

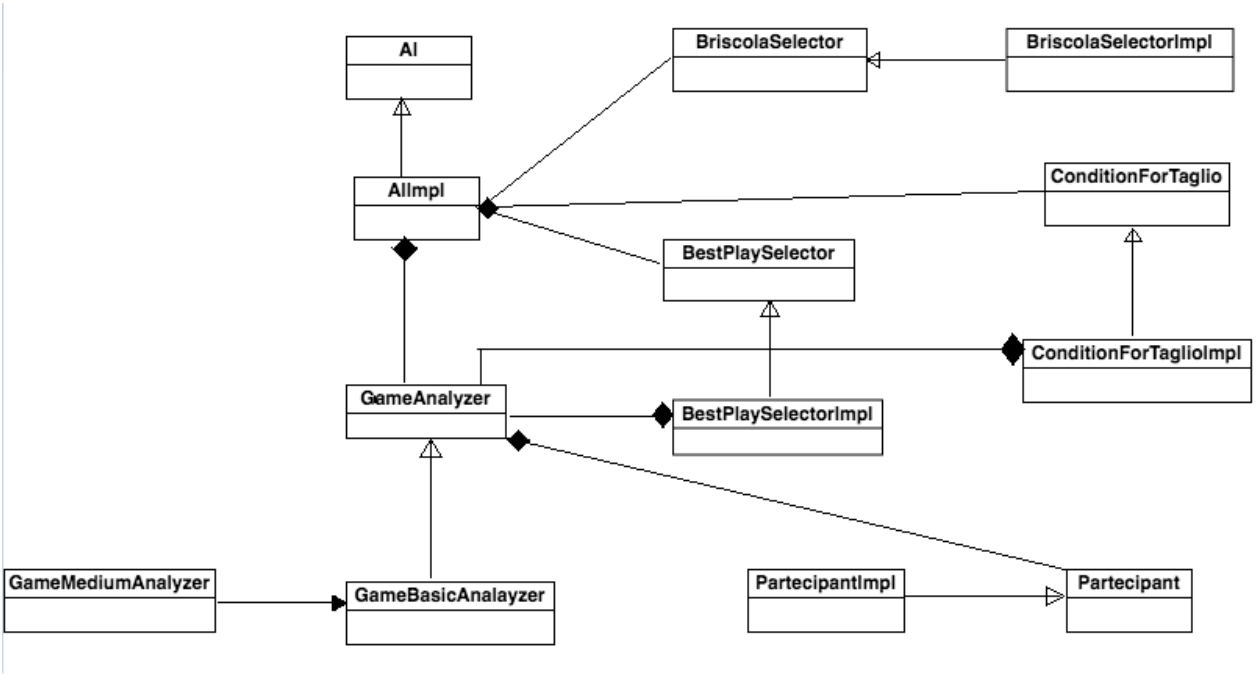

Figura 2.5: diagramma UML dell'interfaccia di AI.

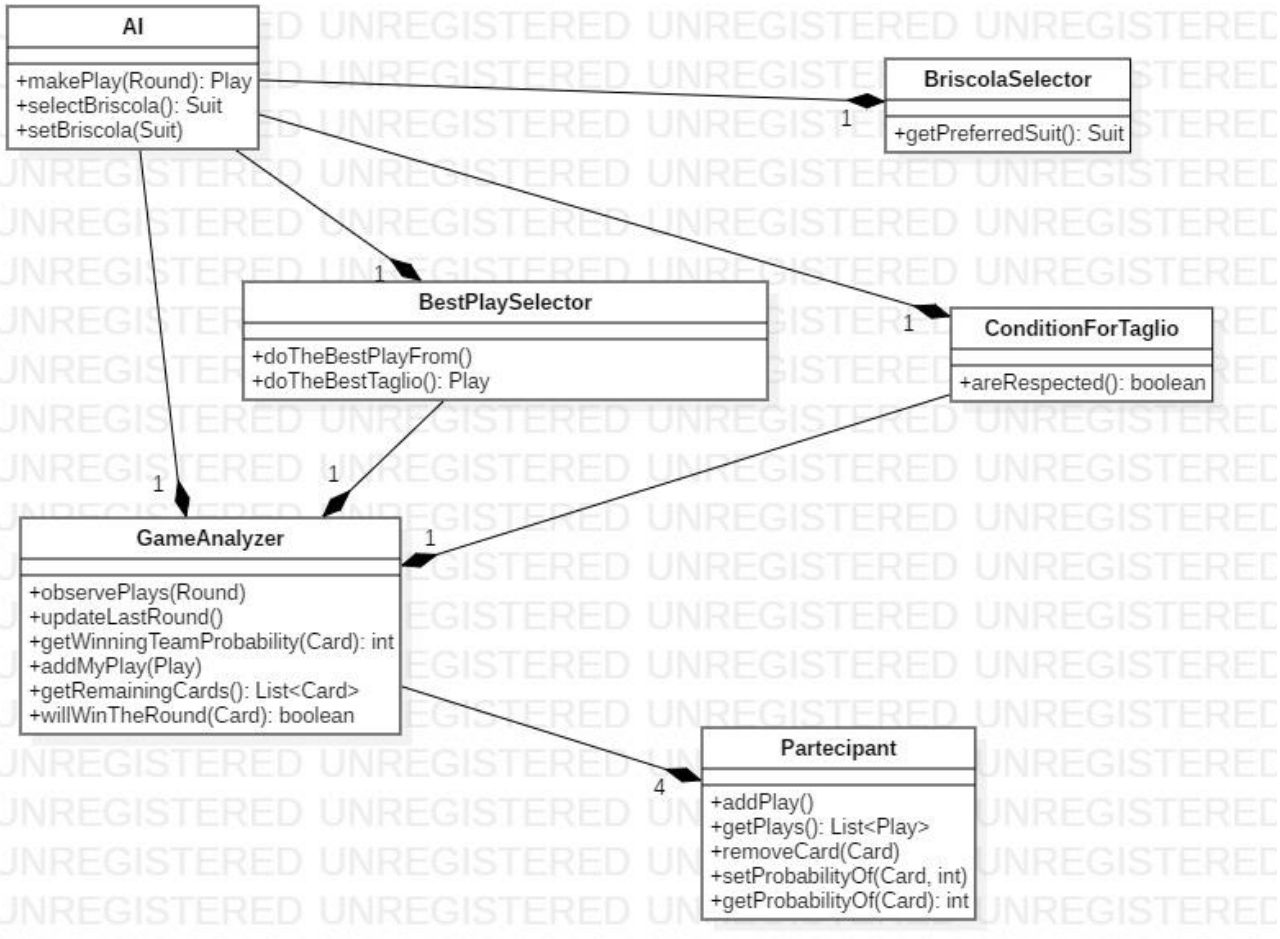

Figura 2.6: diagramma UML delle interfacce per AI.

La parte di progetto di mia competenza riguardava l'implementazione dell'intelligenza artificiale e di utility per una corretta gestione delle carte.
In fase di analisi, per l'intelligenza artificiale sono partito dal separare i diversi concetti e i ragionamenti che normalmente un giocatore reale effettua durante una partita di Beccaccino.
L'intelligenza artificiale ha le seguenti capacità:

- battezzare, ovvero determinare, la Briscola selezionando il seme di cui ha più carte e, in caso di stessa quantità, scegliere quella con carte di maggior valore. L'AI, quindi, si compone di un BriscolaSelector;
- osservare le giocate del round precedente dopo il suo turno e quelle del round corrente effettuate prima del suo turno e in base ad esse fare calcoli statistici determinando le possibili carte degli altri giocatori. L'AI, quindi, si compone di un GameAnalyzer che, sostanzialmente, ne definisce il livello di gioco e si occupa di ricordare le carte rimanenti, (non ancora giocate), tutte le giocate effettuate fino al proprio turno e il numero di round svolti. Per garantire un corretto funzionamento del GameAnalyzer ho ridefinito i quattro partecipanti alla partita, compreso l'AI stesso per avere un "riferimento" di ogni giocatore ed effettuare i calcoli statistici su di essi;
- giocare la migliore carta possibile al proprio turno attraverso una classe BestPlaySelector la quale sfrutta le statistiche calcolate dal GameAnalyzer;
- tagliare, ovvero giocare una Briscola quando il seme iniziale del round non è del seme di Briscola, nei momenti più opportuni della partita quando è possibile attraverso ConditionForTaglio.

Attraverso questo design, basato sul pattern "Strategy" (come ad esempio mostrato in figura 2.6), ottengo la possibilità di poter implementare in futuro AI più avanzate semplicemente creando nuove classi che estendono una GameBasicAnalyzer, che definisce il normale comportamento di un AI base, aggiornando e migliorando i suoi calcoli statistici sulle carte che potrebbero avere i partecipanti alla partita. Come quanto fatto in AIImpl la stessa situazione dell'utilizzo del pattern Strategy si ripete anche in BestPlaySelectorImpl.
L'AI di medio livello ha infatti come unica differenza rispetto a quello base di essere "gestito" da un GameMediumAnalyzer che rispetto al GameBasicAnalyzer aggiunge all'AI la capacità di ricordarsi dei semi esauriti dai giocatori ed effettuare attraverso ciò calcoli probabilistici sul possesso delle carte più avanzati.
Praticamente in tutte le classi fin qui citate mi sono avvalso delle due classi di utility create per una corretta gestione delle carte ovvero la BunchOfCards e il BeccaccinoComparator.

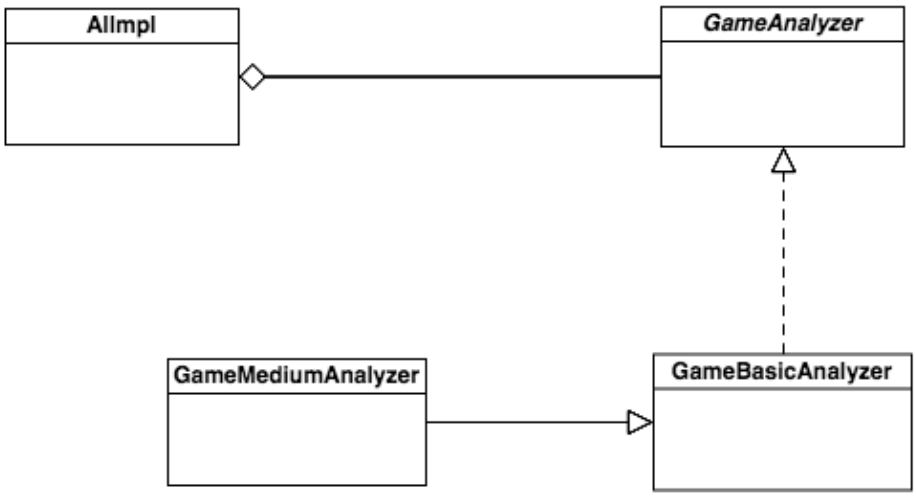

Figura 2.7: diagramma UML dell'interfaccia Strategy di GameAnalyzer.

Alessia Rocco
La mia parte di progetto riguardava la parte di view principale del gioco e l'implementazione della logica dei giocatori, delle squadre e delle singole giocate.

Durante la fase di analisi è emerso che la parte di view consisteva essenzialmente nella visualizzazione di una base, che rappresenterebbe il tavolo da gioco, in cui tutti gli elementi utili al giocatore a comprendere meglio quello che accade durante una partita trovano un posto.
Indispensabile è la presenza sul tavolo delle quattro mani di carte dei quattro giocatori. Per mantenere in un futuro la possibilità di estendere e cambiare il metodo con cui si visualizzano le singole carte dei giocatori ho deciso di creare a parte la rappresentazione delle carte attraverso il pattern creazionale Factory Method in modo che per rappresentare anche un numero più consistente di carte non si debba creare una classe a doc per ogni singola classe, inoltre questo metodo consente di poter estendere in modi diversi la loro rappresentazione non comporti cambiamenti in tutta la view. (di questo se ne occupa la classe ItalianCardViewFactory).
Ogni carta, a prescindere dalla realizzazione, ha diverse azioni che interagiscono direttamente con l'utente: selezionare una carta per giocarla e aggiungere eventualmente un messaggio. Durante l'implementazione ho notato che anche i messaggi erano un elemento che poteva essere in futuro cambiato o addirittura ignorato, per questo motivo ho risolto la loro presenza attraverso una classe separata e opzionale.
Per quanto riguarda la gestione degli eventi associati alle carte ho deciso di staccare la parte più di controllo, come il fatto di catturare quale carta fosse stata selezionata e la sua conseguente giocata, attraverso il pattern Strategy creando nella parte di controllo una classe addetta a tali operazioni (ItalianCardController).
Dato che il gioco è sequenziale e i turni di fatto si susseguono uno dopo l'altro, abbiamo deciso di non introdurre dei thread diversi che gestissero i vari turni ma semplicemente abbiamo deciso di avvalerci del thread principale come unico nella gestione della conseguenzialità delle giocate. Ho deciso per questo di avvalermi delle Dialog, che le librerie di JavaFX mettono a disposizione, per comunicare le principali informazioni del gioco con l'utente, in quanto elementi bloccanti che consentono per tanto di dare il tempo all'user di leggere e decidere quando il gioco debba continuare.
Durante la fase di design inoltre mi sono resa conto che molte informazioni come la fine del turno di ogni utente e la fine di un round potevano essere entrambe visualizzate tramite delle Alert e per tanto sarebbe stato molto ripetitivo creare ogni volta una nuova classe apposta. Così ho deciso di creare una classe Factory per le Alert in modo tale che ogni volta che si voglia comunicare con l'utente tramite questa logica basti chiamare la classe AlertInformationFactory e in poche direttive si crei immediatamente la Alert di cui si aveva bisogno.
La classe principale e tutte le classi che riguardano la view sono state realizzate attraverso la libreria JavaFX, consentendomi di separare la parte grafica della GUI della scena dalla logica che ne definisce il comportamento in modo tale da avere completa indipendenza tra l'implementazione logica delle carte, dei giocatori e delle squadre dalla grafica delle stesse.

Per quanto riguarda l'implementazione logica delle tre entità principali del gioco ho deciso di svilupparle tutte quante attraverso l'implementazione di un'interfaccia che generalizzasse i comportamenti principali e generali in tutti i possibili giochi di carte ed eventuali estensioni future. Per esempio, ci vorrà sempre per un giocatore un nome e la necessità di saperlo, per le giocate invece si dovrà sempre tener conto delle carte giocate e da chi giocate e infine, per quanto riguarda le squadre, se presenti, tener conto di quali giocatori ne fanno parte.

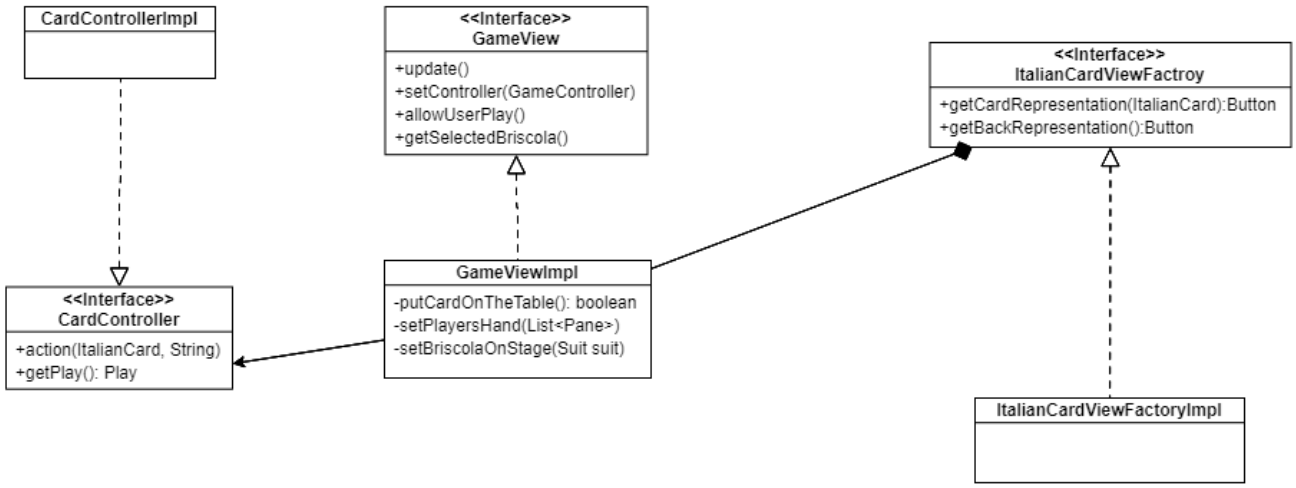

Figura 2.8: diagramma UML dell'interfaccia GameView e CardController e le loro relazioni.

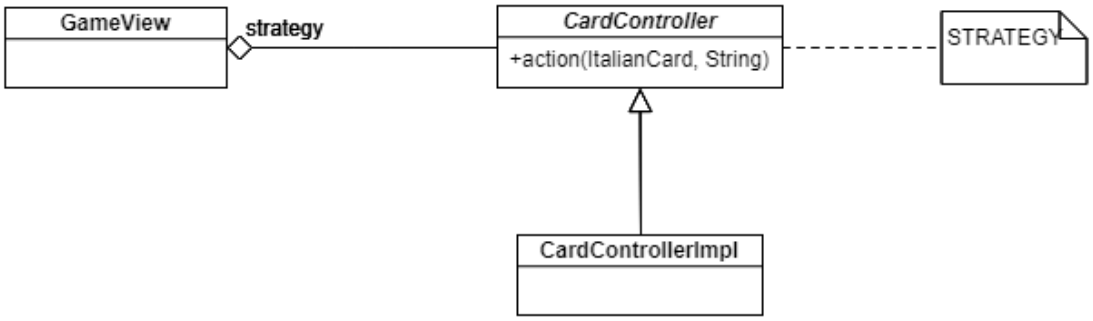

Figura 2.9: diagramma UML del pattern Strategy per il CardController.

CAPITOLO 3: SVILUPPO

3.1 TESTING AUTOMATIZZATO

Durante la fase di sviluppo ci si è avvalsi della libreria JUnit e di un nostro test sequenziale automatizzato per testare alcune funzionalità base. Esse sono:

- *AI:* è stata verificata la correttezza e la coerenza del funzionamento delle giocate effettuate dalle AI. La classe MatchTest è stata creata per simulare una partita completa con 4 intelligenze artificiali. Tale classe è stata creata senza l'ausilio di JUnit ma sfruttando opportune stampe in quanto più efficaci per capire il corretto funzionamento o meno del gioco.
- *Entità*: è stato verificata la correttezza del funzionamento delle entità che costituiscono il gioco quali Player, ItalianCard e Team. La classe TeamTest simula la creazione di alcuni giocatori, di alcune classi e di una squadra e ne testa il corretto funzionamento.

Altri aspetti invece sono stati verificati manualmente:

- *Finestra principale di gioco resizable*: controllo se è possibile ridimensionare e ingrandire la finestra di gioco principale in modo che si adatti automaticamente.
- *Input*: controllo che l'interfaccia grafica risponda in maniera corretta agli eventuali input utente.
- *Portabilità*: controllo che il gioco esegua correttamente su diversi sistemi operativi Windows, MacOs e Linux.

3.2 METODOLOGIA DI LAVORO

La ripartizione dei compiti inizialmente attribuita è stata ampiamente rivisitata, per far fronte a problemi organizzativi. La divisione aggiornata è la seguente:

- **Davide Alpi:** gestione della partita (controller), partita e mano dei giocatori (model), schermata di fine partita.
- **Riccardo Berti:** gestione dei menù iniziali, gestione dei profili, gestione settings, ruleset e implementazione delle carte e del deck.
- **Francesco Foschini:** implementazione delle intelligenze artificiali e implementazione di classe di operazioni di utility sulle carte.
- **Alessia Rocco:** gestione della view del gioco principale, schermata di scelta Briscola e dei messaggi, creazione e del controllo delle carte e implementazione delle entità quali le giocate, i giocatori e le squadre.

Per la realizzazione del progetto ci si è avvalsi del DVCS Git in modo da potersi scambiare in modo efficiente il lavoro fatto da ogni componente e potersi aggiornare e coordinare in modo efficace e immediato. Come servizio di hosting è stato utilizzato Bitbucket mentre come IDE Eclipse.
Durante lo sviluppo si è utilizzato un branch secondario di *develop* e non appena l'applicazione ha raggiunto uno stato stabile pronto per la consegna è stato effettuato un merge sul master.

3.3 NOTE DI SVILUPPO

Alpi Davide

Le feature avanzate del linguaggio Java che ho utilizzato sono:

- **Optional**: largamente utilizzati perché trovati molto utili e di facile comprensione a qualunque sviluppatore si interfacci col mio codice.

Riccardo Berti
Le feature avanzate del linguaggio Java che ho utilizzato sono:

- **Optional**: usati nella classe PreGameMenuControlImpl.
- **JavaFX**: libreria utilizzata per la creazione delle Scene in FXML.
- **FXML**: linguaggio per creare la parte implementativa grafica dei menu, in modo da staccare parte di controllo a parti puramente grafici.
- **File input/output**: per la creazione e gestione dei profili e delle opzioni.

Per poter apprendere in maniera approfondita delle funzionalità di JavaFx, ho fatto riferimento alle slide del corso, ai progetti degli anni passati e a tutorial.

Francesco Foschini
Le feature avanzate del linguaggio Java che ho utilizzato sono:

- **Optional:** praticamente presenti in ogni classe implementate, in particolare in BunchOfCard e GameAnalizer.
- **Comparatore:** attraverso la classe BeccaccinoBunchOfCards, creata apposta per confrontare i valori delle carte.

Alessia Rocco
Le feature avanzate del linguaggio Java che ho utilizzato sono:

- **Optional**: usati in tutte le classi delle entità di model e in CardControllerImpl.
- **Lambda expression**: usate in GameViewImpl.
- **JavaFX**: libreria utilizzata per la creazione della Scene del gioco principale e in tutti gli elementi grafici del gioco.
- **CSS**: il linguaggio che ho integrato in alcune componenti di grafica per settare caratteristiche come font e colore.
- **File input**: per il caricamento delle immagini nelle carte.

Per poter apprendere in maniera approfondita delle funzionalità di JavaFx, ho fatto riferimento alle slide del corso, ai progetti degli anni passati, a tutorial e alla documentazione ufficiale di JavaFX rilasciata da Oracle.

CAPITOLO 4: COMMENTI FINALI

4.1 AUTOVALUTAZIONE E LAVORI FUTURI

Davide Alpi
Questo progetto mi è stato di grande aiuto per comprendere molti dei problemi e dei vantaggi derivati dalla programmazione in team. La principale mancanza che ho sentito è stata quella di una figura coordinatrice e organizzatrice del lavoro del team, anche se alla fine ci siamo alternati abbastanza bene nel ricoprire tale ruolo.
Dal punto di vista personale mi sono visto molto impegnato soprattutto nella fase di design iniziale per quanto riguarda le scelte progettuali. Sono abbastanza soddisfatto del risultato anche se forse avrei dovuto introdurre un livello di astrazione più alto nella realizzazione del Game che costituisse un template per un Game senza il concetto di briscola (seme dominante).
Inizialmente ero anche preoccupato per le poche classi di testing prodotte, ma si sono rivelate ampiamente sufficienti a risolvere i pochi bug incontrati in fase conclusiva.

Riccardo Berti
Per quanto a priori avessi pensato che gestire le scene con SceneBuilder aiutasse a risparmiare tempo, farlo ha sortito l'effetto opposto. I File Fxml creati da SceneBuilder hanno avuto gravi problemi tra cui i mancati riconoscimenti di controller class, injectable fields o metodi di controllo. A posteriori posso dire che se avessi fatto le classi "a mano" avrei risparmiato molto tempo e avrei potuto concentrarmi su aspetti migliorabili come lo scaling delle spaziature nelle scene, l'aspetto grafico degli elementi, misuca o miglioramento della questione profili. Mi prendo la piena responsabilità della decisione e mi rendo conto che la mia parte di programma risulta molto più scarna di quella dei miei compagni e perciò non ne sono al pieno soddisfatto. Nonostante questo, mi trovo a essere contento del lavoro fatto relativo all'I/O, in quanto parte per me ostica durante il periodo di lezione.

Francesco Foschini
Mi ritengo personalmente soddisfatto di questo progetto in quanto, nonostante tutte le difficoltà di organizzazione di gruppo incontrate, il risultato finale rispecchia molte delle caratteristiche proposte inizialmente.
Noto, tuttavia, che queste difficoltà riscontrate hanno portato alla creazione di un prodotto qualitativamente non perfetto; sarà quindi stimolante la possibilità di effettuare migliorie future.
Durante l'implementazione del codice mi sono concentrato principalmente sulla riusabilità per eventuali progetti futuri.
Sono molto soddisfatto del risultato della parte di mia competenza che era relativa principalmente all'implementazione dell'intelligenza artificiale, anche se alcune migliorie potranno essere elaborate in futuro. In particolare:

- Migliorare le condizioni per tagliare.
- Aggiungere migliori funzionalità alla GameMediumAnalyzer.
- Creare un'intelligenza artificiale avanzata attraverso la creazione GameAdvancedAnalyzer che estenda GameMediumAnalyzer e aggiunga nuove funzionalità. Ad esempio, considerare, ad ogni round, le carte giocate dai giocatori stabilendo, in maniera più efficace e precisa, quale giocatore potrebbe avere una determinata carta.

In futuro vorrei poter lanciare questa applicazione sul Play Store, per far giocare diversi utenti e questo sarebbe motivo di grande soddisfazione.

Essendo un giocatore medio-esperto di Beccaccino sono soddisfatto di avere implementato le intelligenze artificiali in quanto questo mi ha permesso non solo di approfondire aspetti dell'OOP ma anche di migliorare e di apprendere nuove strategie di gioco.

Alessia Rocco
La realizzazione di questo progetto mi ha consentito di dedicarmi in maniera approfondita agli aspetti di programmazione OOP, consentendomi di farmi comprendere in maniera definitiva anche argomenti e aspetti più delicati di questo linguaggio. Nonostante il codice e la sua estendibilità possa essere migliorato in quanto mi rendo conto non sia perfetto, sono comunque globalmente soddisfatta del lavoro che ho portato avanti e che abbiamo alla fine realizzato.
Alcuni fattori che potrebbero essere migliorati sono la distinzione tra View e Controller, che riguarda l'implementazione della view generale, che mi sono resa conto non essere stata completamente rispettata.
Mi auspico vivamente che il progetto possa essere mandato avanti anche in futuro, magari per un'estensione che permetta di effettuare una partita tra più di un giocatore umano o di poterlo riutilizzare in maniera efficace per la realizzazione di altri giochi di carte.
Dal punto di vista personale mi ha aiutato a crescere sia nella gestione di situazioni organizzative, che nel mio personale rapporto con gli altri. Nonostante ci siano stati problemi organizzativi e non sia sempre stato facile accordarsi sono soddisfatta di come siamo riusciti a risolvere ogni problema nel miglior modo a noi possibile.

4.2 DIFFICOLTA' INCONTRATE E COMMENTI PER I DOCENTI

Riccardo Berti
A causa dei miei problemi con SceneBuilder consiglierei vivamente di suggerire ai prossimi studenti di non utilizzarlo. Per quanto riguarda il corso ritengo che dedicare qualche ora in più alla parte di GUI (in particolare quella relativa a JavaFX) sarebbe di grande aiuto per il progetto finale, in quanto io la ritenga parte estremamente importante.

Alessia Rocco
Visto il tempo dedicato e i problemi riscontrati sia nell'analisi nell'apprendimento di JavaFX, consiglierei di dedicare qualche ora in più all'approfondimento di JavaFX. Questo credo possa essere non solo di grande aiuto per il progetto finale ma anche molto utile per una conoscenza più dettagliata alla fine del corso.

APPENDICE A: GUIDA UTENTE

All'avvio dell'applicazione si aprirà il menu principale, nel quale si potrà scegliere tra diverse opzioni:

- **Start:** si inizia il gioco con una nuova partita. Si apre la finestra di scelta dei giocatori, eventuali giocatori creati in precedenza potranno essere selezionati.
- **Create new Profile:** permette di creare un nuovo profilo da poter utilizzare nel gioco. Si viene reindirizzati in una nuova finestra con le indicazioni di come fare.
- **Settings:** indirizza alle impostazioni del gioco che possono essere selezionate o meno. Si può infatti selezionare i "Points reward for Cricca" che è l'opzione per considerare i 3 punti nel caso un giocatore possieda la "Cricca".
- **Exit:** consente di chiudere l'applicazione.

Figura A.1: Menù iniziale del gioco

Figura A.2: Schermata Settings

Start

Appena l'utente avrà premuto start comparirà un'altra schermata che presenta la disposizione dei quattro giocatori. Tre di questi, rispettivamente quelli di destra, sopra e sinistra, apparterranno alle intelligenze artificiali. Di questi tre giocatori è possibile selezionare, tramite apposita tendina, le difficoltà degli AI che in questo gioco sono "AI basic", ovvero una AI base pari alle conoscenze di un giocatore novizio, e "AI medium", ovvero una AI media paro alle conoscenze di un giocatore più esperto. Il quarto giocatore invece è lo User che dovrà selezionare uno tra i profili creati per poter iniziare a giocare.

Una volta impostato tutti i giocatori sarà possibile premere sul pulsante "start" per poter iniziare effettivamente il gioco.

Figura A.3: Schermata Start

Create New Profile

Si ricorda inoltre che prima di iniziare un nuovo gioco l'utente dovrà creare almeno un nuovo giocatore attraverso "Create New Profile". Questo è necessario per iniziare un nuovo gioco in quanto l'applicazione necessita delle informazioni di almeno un profilo per poter salvare il giocatore corrente e le sue relative informazioni, per tanto se è la prima volta che si apre il gioco la prima cosa da fare è creare un nuovo profilo da questo menu. In caso contrario il gioco segnalerà questa necessità aiutando l'utente sul da farsi, bloccando l'avvio dell'applicazione fino a quando almeno un profilo non è stato creato e selezionato.

Figura A.4: Schermata Create New Profile.

Come si vede dall'immagine sopra, in "Create New Profile" c'è un'area di testo dedicata all'inserimento del nome del profilo che si vuole creare. Quando il nome è stato inserito e si vuole creare si può cliccare sul bottone "Create New Profile" e se il nome del profilo non esiste già e rispecchia i vincoli (senza caratteri speciali e che non sia un nome vuoto) allora verrà creato con successo.

Schermata Principale di Gioco

Figura A.5: Schermata principale di gioco.

Durante il gioco nella schermata principale saranno presenti le 40 carte distribuite tra i 4 giocatori, l'utente possiede le carte in basso e scoperte, mentre le carte degli avversari si trovano agli altri 3 lati, questa volta coperte. Al proprio turno l'utente dovrà selezionare una carta tra quelle disponibili, si riconoscono in quanto le uniche selezionabili. Con la selezione della carta si avvierà una finestra che richiede al giocatore di allegare un messaggio, nel caso lo si voglia, alla carta giocata. Se si vuole cambiare carta al momento della finestra di messaggio basterà cliccare su "annulla" se invece si vuole giocare la carta e il messaggio basterà premere "ok". Ogni qualvolta inoltre l'utente deve selezionare la briscola, uscirà una finestra di dialogo che permetterà la scelta, si ricorda che nel caso l'utente preme "annulla" verrà selezionata in automatico la briscola di bastoni dato che il gioco non può continuare senza una briscola.