Relazione “Squares”

Karl Darrajati
Licia Valentini
Giacomo Venturini

12 Maggio 2015

Capitolo 1

Analisi

1.1 Requisiti

Il software mira a riprodurre il famoso gioco da tavolo per due giocatori “Dots and Boxes”. Inoltre il software offre la possibilità di scegliere un’ulteriore modalità di gioco che aggiunge modifiche alle regole “standard” ottenendo così un nuovo modo di giocare.

Requisiti concordati:

- Creazione di un campo da gioco in grado di gestire l’alternarsi dei turni.
- Scelta della dimensione del tabellone di gioco, per partite più o meno lunghe.
- Scelta tra due diverse modalità di gioco.
- Visualizzazione della durata di una partita una volta terminata.
- Gestione di una classifica con i risultati di tutte le partite, ordinabile secondo vari campi
- Visualizzazione delle regole di gioco per i nuovi utenti
- Implementazione dell’audio di gioco
- La possibilità di personalizzare i colori dell’applicazione

1.2 Analisi e modello del dominio

Il software dovrà essere in grado di fornire agli utenti la possibilità di personalizzare l’interfaccia pre-partita, dando la possibilità di inserire un nickname univoco, di cambiare le dimensioni del campo da gioco e di scegliere tra due diverse modalità. Le modalità dovranno essere il più possibile riutilizzabili in modo che l’inserimento di una nuova modalità in futuro non comporti la totale riscrittura dei metodi offerti per l’inserimento di una mossa e il calcolo dei punteggi già correttamente funzionanti. Inoltre, a fine partita, i risultati saranno salvati in una classifica, che può essere riordinata in quattro modi differenti, in modo da vedere le performance dei giocatori sotto campi diversi.

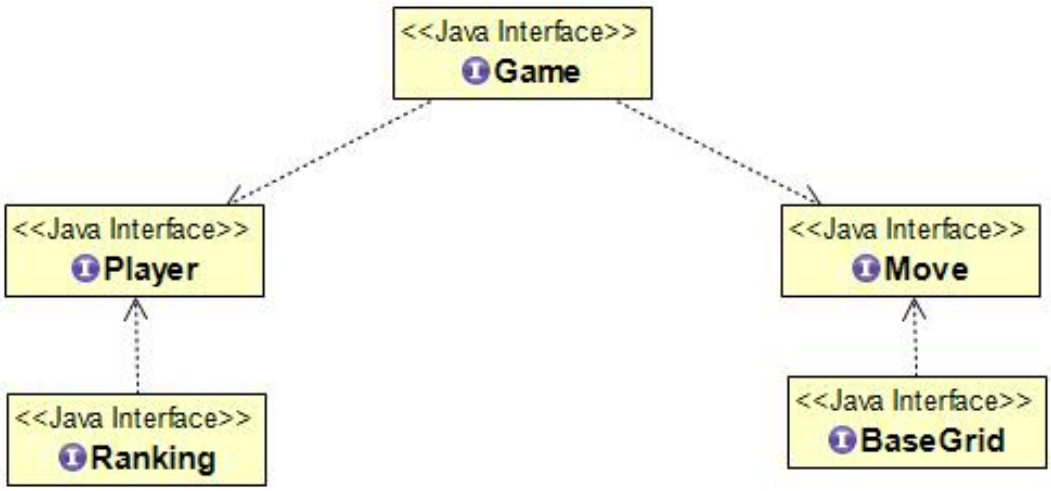


Figura 1.1: Schema UML che rappresenta le entità individuate per la risoluzione del problema, che costituiscono lo scheletro del model.

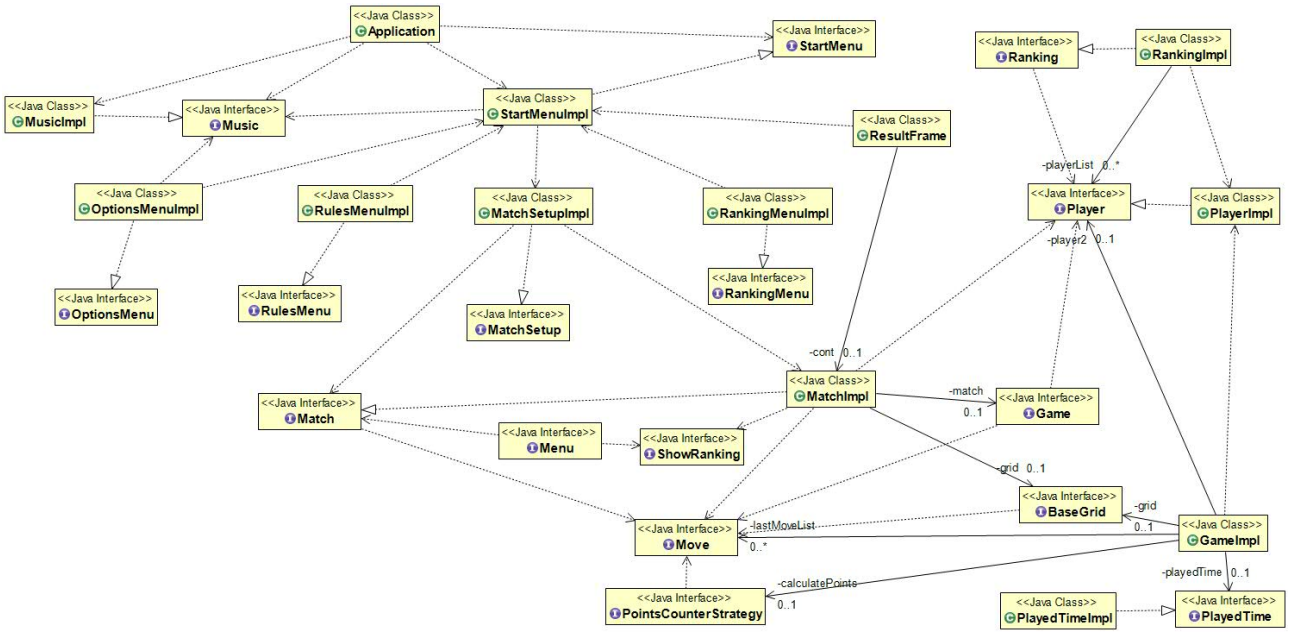
Capitolo 2

Design

2.1 Architettura

Per il progetto è stato utilizzato il pattern architetturale Model-View-Controller:

- **Model:** attua il dominio del modello studiato per risolvere il problema. Si occupa di implementare le entità precedentemente individuate, cercando di generalizzare l'algebra che sta alla base dell'inserimento e del calcolo dei punti in modo tale che in futuro si possano aggiungere nuove modalità riutilizzando le parti già correttamente funzionanti senza dover riscrivere da capo il codice.
- **View:** si occupa di mostrare graficamente all'utente le diverse operazioni offerte dall'applicazione, come la creazione di una nuova partita e tutte le azioni che permettono agli utenti di giocare, la visualizzazione delle regole di gioco e la visualizzazione della classifica.
- **Controller:** si occupa di gestire e stabilire la corretta connessione tra model e view. Inoltre si occupa del salvataggio su file delle statistiche di ogni giocatore alla fine di ogni partita e delle regole di gioco, che all'occorrenza possono essere caricate e mostrate all'utente.



2.2 Design dettagliato

L’applicazione è stata suddivisa in quattro sottoparti logiche, Model, Controller e View, proprio come descritto nell’architettura, a cui si affianca il package contenete il main dell’applicazione. Maggiori dettagli sulle scelte fatte:

Radice: la parte della radice in comune è **it.unibo.squaresgameteam.squares** che poi si suddivide nelle quattro sottoparti sopracitate, **.model .controller .view** e **.application**. Il sottopackage **.model** si suddivide a sua volta poi in **.interfaces** e **.classes** che contengono le interfacce e le classi del model, **.enumerations**, che contiene le sue enumerazioni, **.exceptions**, che contiene le eccezioni personalizzate lanciate in caso di errore e **.tests**, che contiene le classi che effettuano i test delle classi del model.

Il sottopackage **.controller** contiene i packages .**classes**, contenente le classi del controller, **.interfaces**, contenente le interfacce, **.enumerations**, contenente l'enumerazione che permette di scegliere la modalità di gioco.

Il sottopackage **.view** contiene i packages .**classes**, contenente le classi della view, e **.interfaces**, contenente le interfacce.

2.2.1 Model

In questa parte di progetto sono implementate le interfacce sopracitate, facendo particolare attenzione alla possibilità in futuro di poter aggiungere nuove modalità di gioco, senza comportare una totale riscrittura dei metodi già correttamente implementati. Ciò è permesso grazie all'uso del pattern strategy, che permette alla classe principale di delegare le funzioni specifiche della modalità di gioco scelta, come l'inserimento di una mossa e il relativo calcolo dei punti, a classi specializzate.

Inoltre, per la gestione delle informazioni relative ad ogni giocatore, si è utilizzato il pattern builder per evitare conflitti riguardanti l'assegnamento del numero di partite vinte prima di quello relativo alle partite totali, che ovviamente non devono mai essere superate dalle prime, dato che logicamente non avrebbe senso.

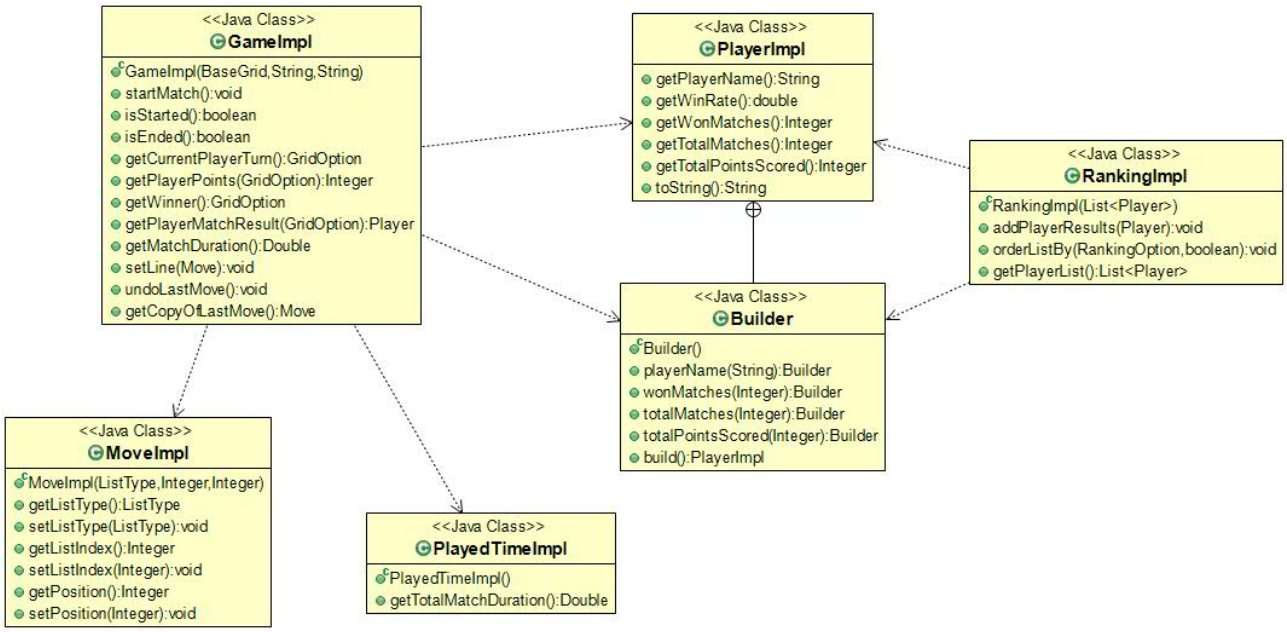


Figura 2.2.1.1 schema UML relativo alle classi del dominio, che mostra come sono implementati i metodi delle interfacce individuate in fase di analisi. Dallo schema emerge l'uso del pattern Builder per la creazione di un oggetto di tipo player che come spiegato precedentemente evita i conflitti che si sarebbero generati dall'inserimento tramite setter dei campi richiesti.

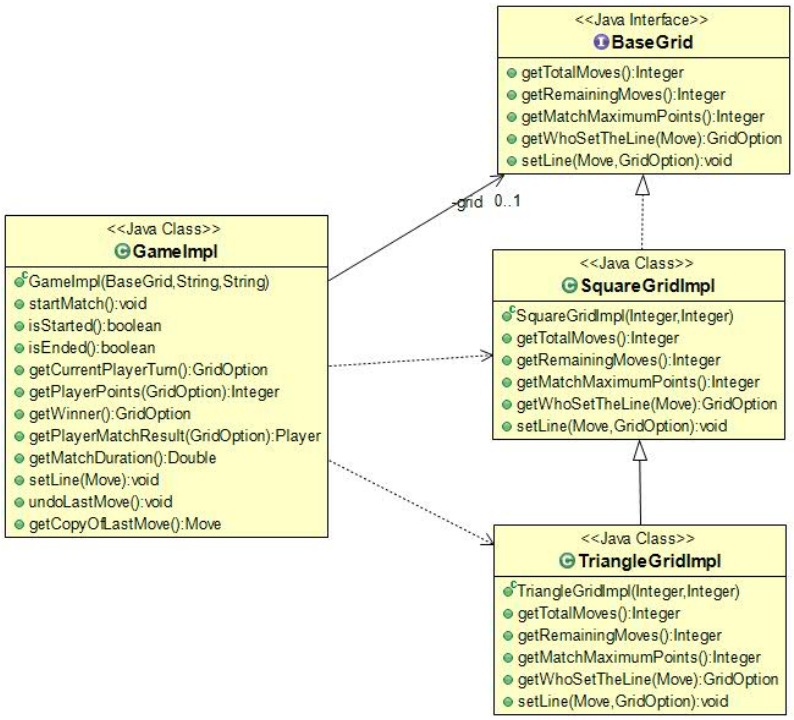


Figura 2.2.1.2 schema UML con in dettaglio le classi che compongono il campo di gioco della classe GameImpl. Viene in particolare messo in evidenza come il pattern Strategy utilizzato permetta l'implementazione di nuove modalità di gioco senza la modifica dei metodo di insermento o di annullamento di una mossa.

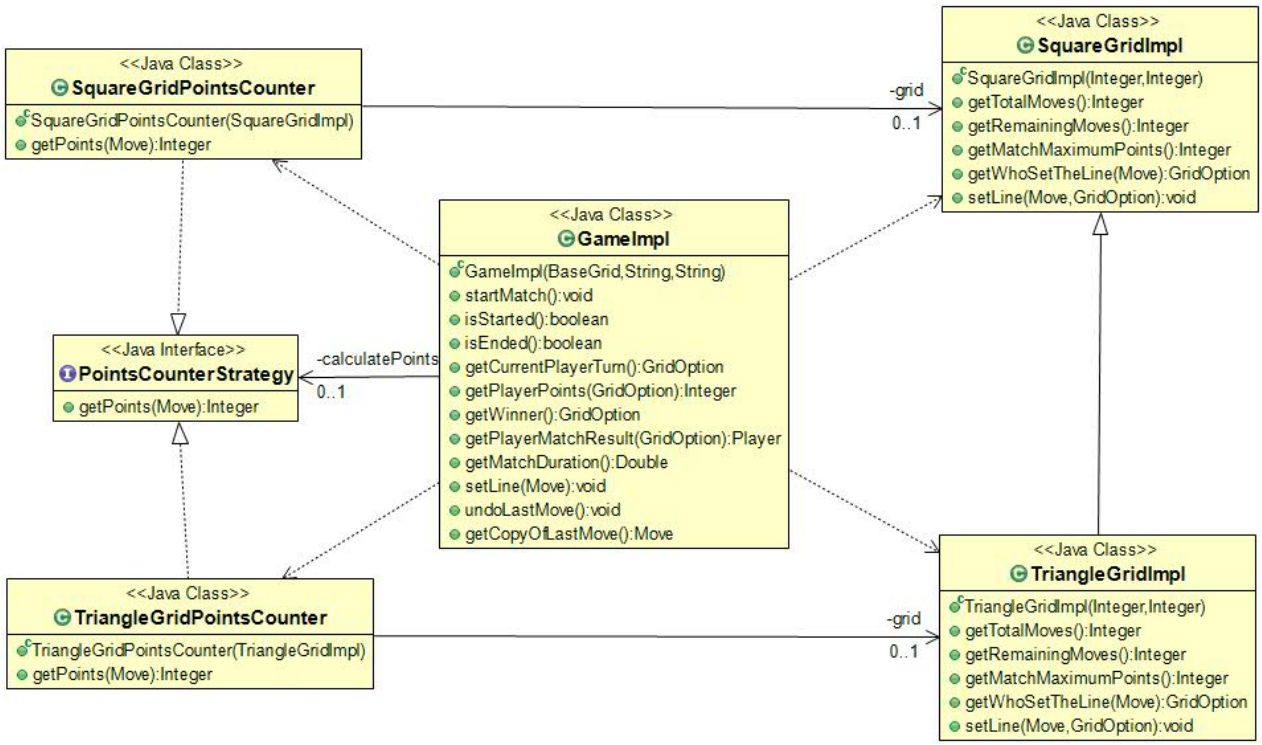
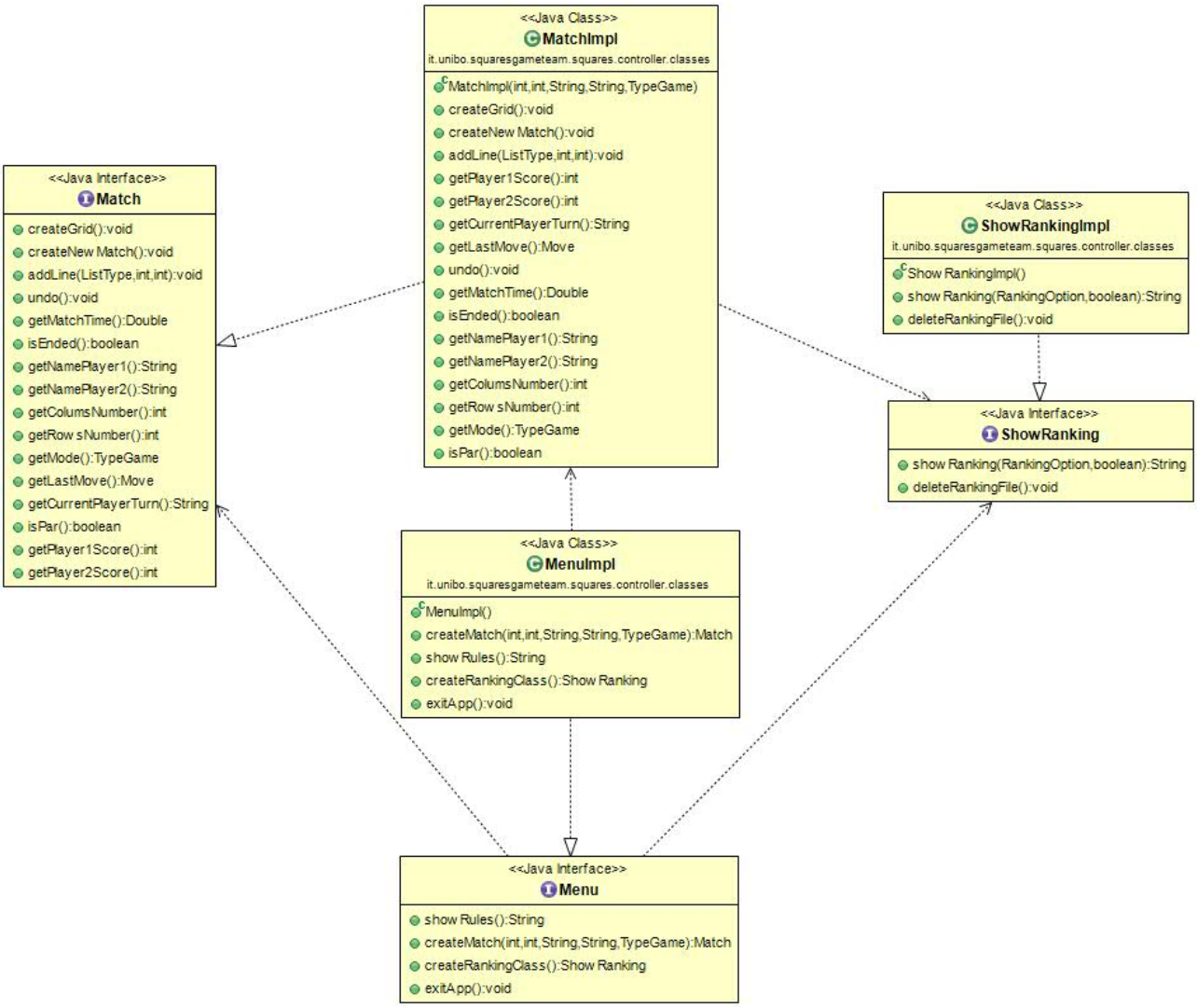


Figura 2.2.1.3 schema UML che mostra come anche il calcolo del punteggio, che avviene ogni volta che viene inserita una mossa, sia stato implementato attraverso il pattern Strategy, così che la classe GameImpl non debba occuparsene direttamente, evitando la modifica dei suoi metodi alla creazione di nuove modalità di gioco.

2.2.2 Controller

La parte del controller è dedicata alla gestione del passaggio di input da parte della view al model e a quello di output dal model alla view. Oltre questo è stata implementata la gestione della classifica, il suo salvataggio su file, il passaggio dei risultati a fine partita e la conversione in stringa, con tanto di formattazione, per far in modo che venga correttamente stampata a view.



La classe MatchImpl è il cuore dell'applicazione, in quanto si occupa dell'intera partita, dal passaggio dei parametri per la griglia e la modalità di gioco al model al salvataggio del risultato di entrambi i giocatori nella classifica. Questa classe fa uso dell'enumerazione TypeGame, che permette di scegliere la modalità di gioco e facilita in un futuro aggiornamento, l'inserimento di nuove modalità, senza il bisogno di modificare codice già funzionante.

Tramite il costruttore della classe si possono inserire i parametri per iniziare una

partita, poi i vari metodi si occupano di gestire input e output. In particolare abbiamo la conversione del tipo con cui sono individuati e distinti i due giocatori, da GridOption (model) a string (view). In una prima implementazione c'era un solo getter per lo score, visto che il gioco procede per turni e viene aggiornato solo il punteggio del giocatore che ha fatto una mossa, ma per facilitare il lavoro della view si è deciso di aggiornare entrambi gli score ad ogni turno, tramite due metodi getter.

Alla fine della partita, la classe passa automaticamente i risultati alla classifica.

La classe ShowRankingImpl gestisce la classifica, scrivendola e leggendola su file. Le varie funzionalità sono state divise in metodi privati per rendere il codice ordinato e più leggibile.

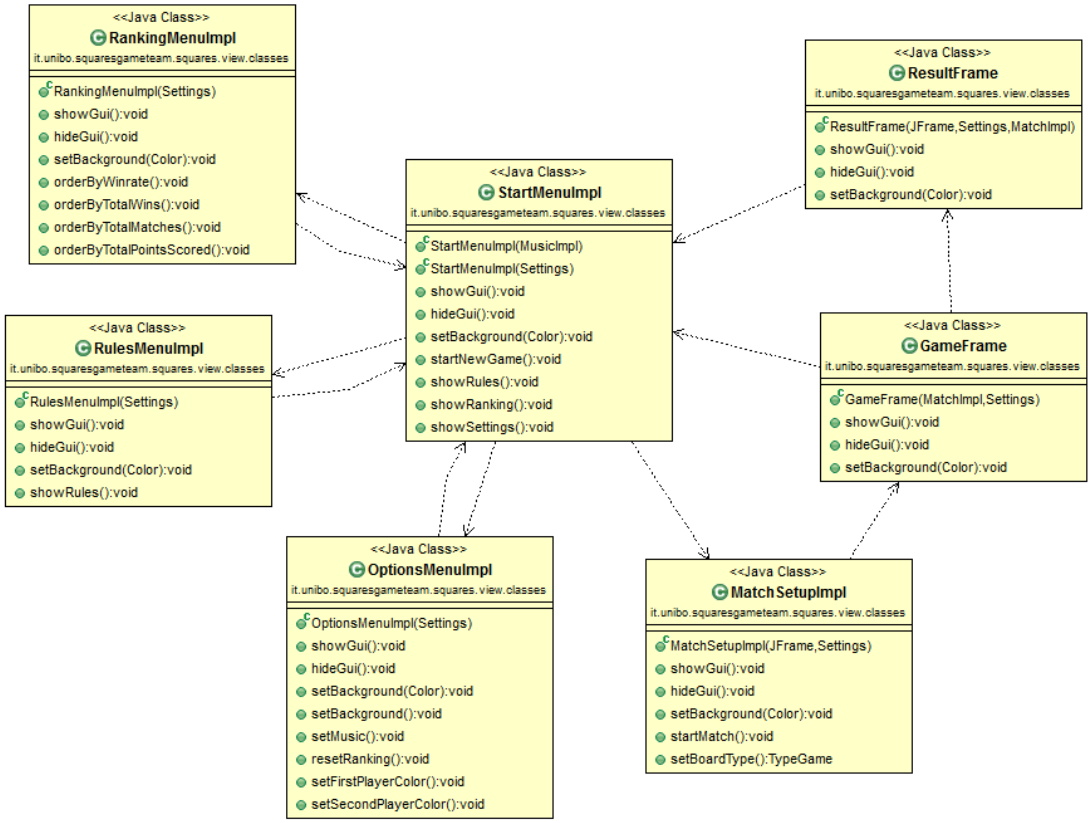
Gli unici accessi possibili alla classe sono tramite i metodi per stampare la classifica e per eliminarla, in caso l'utente voglia farlo, così che non si possa modificare se non a fine partita tramite la classe MatchImpl, grazie al metodo addPlayer che è dichiarato proprio per questo protected.

La classe MenuImpl gestisce il menù principale dell'applicazione, creando gli oggetti del controller, leggendo le regole da file o uscendo dall'applicazione, secondo le richieste dell'utente.

2.2.3 View

La View si occupa di gestire l'interazione con l'utente e di mostrare a video una rappresentazione grafica dello stato del Model durante una partita. Per realizzarla è stato scelto di utilizzare la libreria Swing al fine di poter mettere in atto tutte le conoscenze apprese durante il corso.

All'avvio dell'applicazione viene visualizzato un menù dedicato alla navigazione tra i vari frame. Al fine di rendere la gestione dei frame il più possibile flessibile ed espandibile, si è optato per reallizzare una classe per ogni frame, cosicchè sono tra loro il più indipendenti possibile.



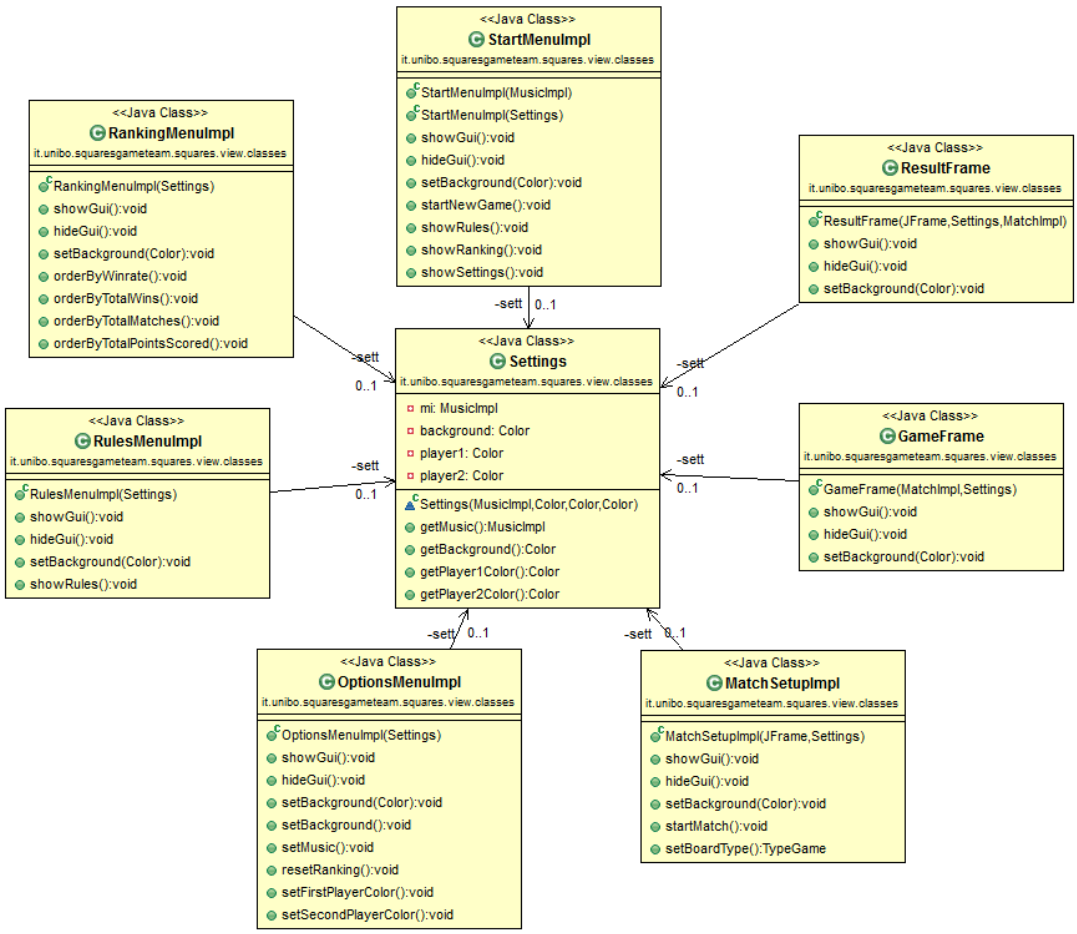
Tutti i frame hanno tra di loro degli elementi in comune.

Usano tutti un AbsoluteLayout, in quanto questo layout offre le maggiori possibilità di personalizzazione.

Essi sono inoltre gestiti dall'interfaccia GuiElements, che rende disponibili tre metodi riguardanti semplicemente l'aspetto grafico che rendono visibile il frame, lo nascondono e cancellano e che applicano un colore specificato da parametro allo sfondo.

Oltre all'interfaccia GuiElements vi è una classe che gestisce le impostazioni dell'applicazione.

Per fare ciò si è optato per il passaggio di un oggetto della suddetta classe di classe in classe e da questo oggetto è possibile quindi ottenere i colori che rappresentano i giocatori, il colore dello sfondo e l'oggetto musica passato dal controller.

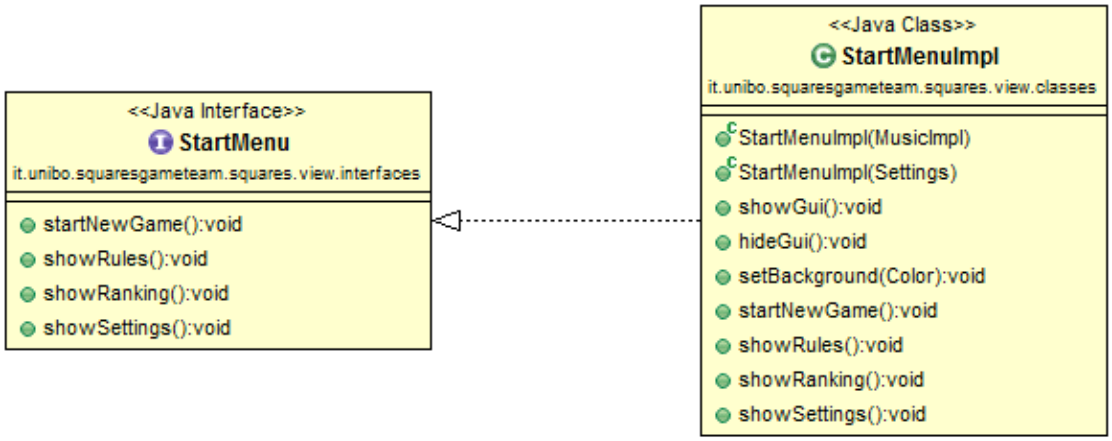


La classe StartMenuImpl contiene il menu principale dell'applicazione.

Essa implementa l'interfaccia StartMenu che contiene i metodi necessari per potersi muovere tra i vari frame.

Ciò è possibile grazie a dei JButton.

Questa classe contiene due costruttori: uno viene richiamato dal main all'avvio dall'applicazione, mentre l'altro viene richiamato ogni volta che si torna al menu principale.

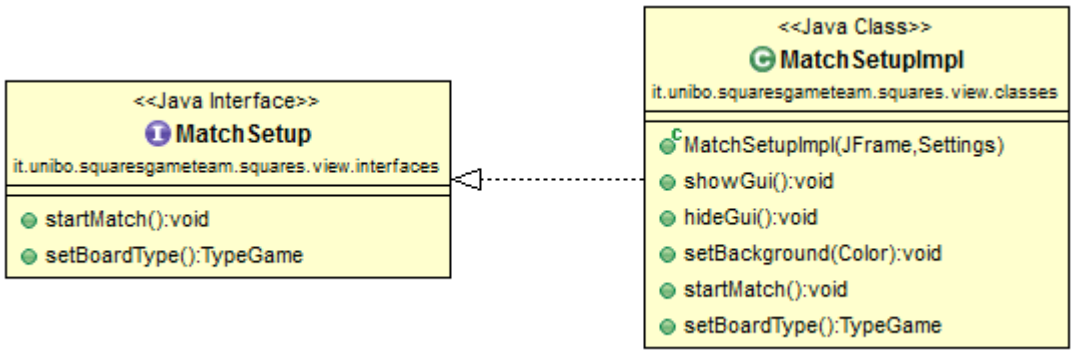


La classe MatchSetupImpl fa scegliere il nome dei giocatori, la grandezza della griglia e la modalità di gioco.

Essa implementa l'interfaccia StartMenu che contiene i metodi necessari per poter iniziare il match e passare al frame contenente il gioco.

A questa classe vengono passati, tramite costruttore, il frame del menu principale (per poterlo poi nascondere) e l'oggetto contenente le impostazioni dell'applicazione.

Per fare inserire le impostazioni del match si è scelto di utilizzare delle JTextField per i nomi dei giocatori, dei JSpinner per le dimensioni della griglia e una JComboBox per scegliere la modalità di gioco.

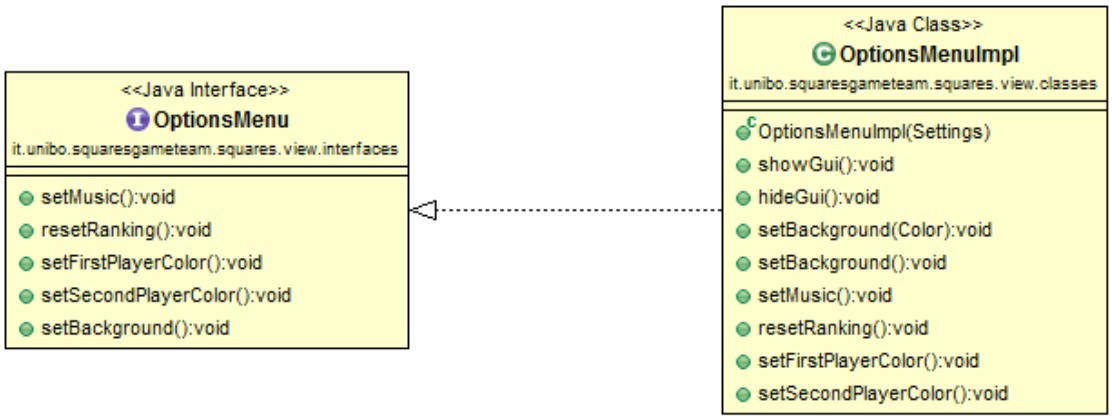


La classe OptionsMenuImpl fa scegliere il colore dei giocatori, il colore dello sfondo, se accendere o spegnere la musica e se cancellare la classifica.

Essa implementa l'interfaccia OptionsMenu che contiene i metodi necessari per poter modificare le varie impostazioni.

A questa classe viene passato, tramite costruttore, l'oggetto contenente le impostazioni dell'applicazione, nel quale verranno memorizzate.

Per cambiare le impostazioni dell'applicazione si è scelto di utilizzare dei semplici JButton.

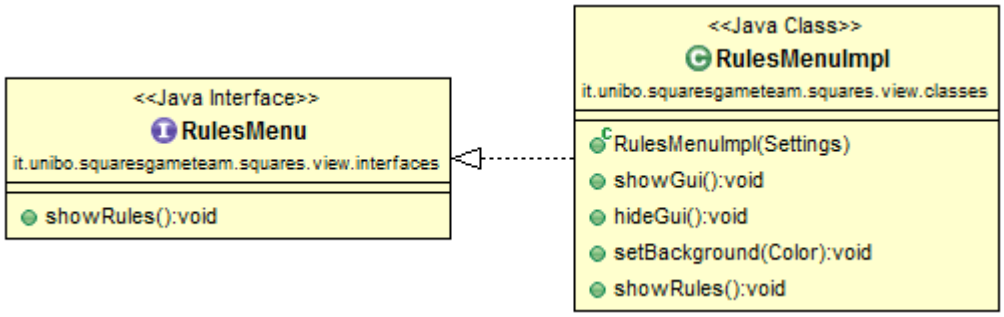


La classe RulesMenuImpl visualizza le regole del gioco.

Essa implementa l'interfaccia RulesMenu che contiene i metodi necessari per poterle visualizzare.

A questa classe viene passato, tramite costruttore, l'oggetto contenente le impostazioni dell'applicazione.

Per visualizzare le regole si è optato per una JTextArea, la quale è inserita in un JScrollPane per poter permettere l'eventuale navigazione verticale.

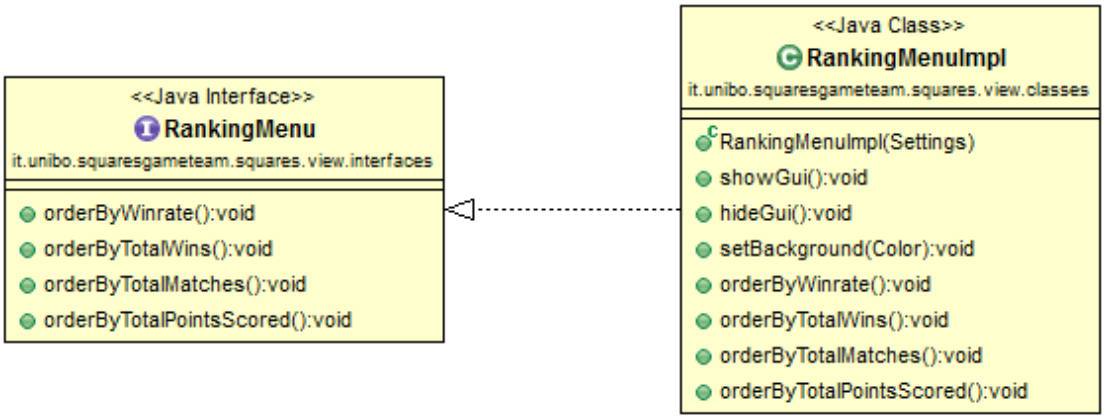


La classe RankingMenuImpl visualizza le statistiche dei giocatori.

Essa implementa l'interfaccia RankingMenu che contiene i metodi necessari per poter ordinare e visualizzare il tutto.

A questa classe viene passato, tramite costruttore, l'oggetto contenente le impostazioni dell'applicazione.

Per visualizzare il ranking si è scelto di utilizzare, come nel caso delle regole, una JTextArea dentro a un JScrollPane. Inoltre sono stati utilizzati dei JRadioButton per scegliere in base a cosa ordinare le statistiche e se dal basso verso l'alto o viceversa.



La classe GameFrame contiene il gioco vero e proprio. In essa si trovano la griglia, un pulsante per tornare indietro di una mossa e i punteggi che vengono aggiornati dinamicamente.

A questa classe viene passato, tramite costruttore, l'oggetto contenente le impostazioni dell'applicazione e l'oggetto che gestisce il match(controller).

Per visualizzare la griglia di gioco sono stati creati 3 metodi. Questi rendono possibile l'integrazione delle impostazioni passate dall'oggetto MatchImpl.

Alla fine del match viene visualizzato il frame dei risultati(ResultFrame).

Esso visualizza a video chi ha vinto, con che punteggio e il tempo di gioco.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per verificare la correttezza delle classi implementate dal model sono stati creati alcuni test JUnit. In particolare si è scelto di testare i risultati forniti dalle due modalità di gioco in due classi differenti per migliorarne la leggibilità ed evitare di "sovraccaricare" di test un'unica classe. Inoltre è stato effettuato il test per verificare che i metodi di ordinamento della classifica fossero eseguiti correttamente in base al parametro scelto, soprattutto nei casi in cui il confronto non fosse limitato a soli due campi, ma dovesse essere effettuato basandosi anche sulle altre statistiche del giocatore.

Per quanto riguarda le classi del controller, non sono stati effettuati dei test automatizzati, ma delle simulazioni di partita dal main, creando direttamente da lì gli oggetti e chiamando metodi, a causa della mancanza della view. Non è rimasta traccia di questi test nel codice finale.

Le classi della view invece sono state testate direttamente avviando l'applicazione e verificandone il funzionamento.

3.2 Metodologia di lavoro

Il progetto è stato sviluppato come dichiarato alla sua presentazione:

Model: Giacomo Venturini

Controller: Licia Valentini

View: Karl Darrajati

La parte preliminare di progettazione non è stata svolta in gruppo, dato che la fase di analisi e la sua soluzione è stata lasciata completamente al model. Il difetto di questa scelta è stata che non sono state create le interfacce prima delle classi, ma il contrario, ottenendo come risultato un perfezionamento man mano che il progetto proseguiva. L'unico fattore positivo di questa scelta è stato il fatto che controller e view sono partiti in ritardo rispetto al model, quindi non hanno subito continui cambiamenti dovuti alla modifica delle classi o dei metodi che non erano stati ben definiti.

Il model si è occupato prevalentemente dell’algebra di gioco, trovando una strategia efficace che permettesse l’attuazione di un metodo generale per l’inserimento di una mossa e il calcolo del punteggio che non dipendesse dalle dimensioni del campo da gioco. Inoltre si è occupato di gestire i confronti dei campi necessari al riordinamento della classifica e di trovare un modo di non legare la classe che si occupa dello svolgimento della partita alle classi che determinano la tipologia di gioco.

Il controller ha svolto il ruolo di legare view e model, implementando il sistema di input-output e si è occupato di salvare i risultati delle partite su file. Le varie classi scritte sono dedicate ognuna ad una funzionalità distinta: gestione della partita, gestione della classifica (salvataggio su file, aggiunta nuovi dati a fine partita, conversione in stringa per la stampa), gestione dell'audio, creazione degli oggetti a seconda di cosa si vuole fare nel menù d'avvio.

La view ha svolto il ruolo di stampare a video l'interfaccia grafica dell'applicazione, implementando i metodi e gli oggetti passati dal controller. Inoltre si è preoccupata di rendere l'interfaccia grafica il più flessibile possibile.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Giacomo Venturini: per quanto non abbia seguito la procedura consigliata definendo le interfacce del model in maniera generalizzata per poi procedere con la loro implementazione, sono piuttosto contento di come alla fine è stato ultimato il codice. Sicuramente quest'esperienza mi ha aiutato molto a capire aspetti che non avevo ben compreso a lezione, in particolare l'uso dei pattern che mi hanno aiutato molto a ridurre ed eliminare duplicazioni di codice rendendo il risultato più pulito e ordinato. Non posso tuttavia dire che sono contento di come invece si è svolto il lavoro in team, dato che non siamo riusciti a rispettare la deadline prestabilita. Ma forse la cosa che mi è dispiaciuta di più è stato il fatto che non c'è stato un confronto reciproco sulle scelte adottate, ma ci sia semplicemente adattati dato che i tempi erano stretti. Oltre al ruolo di model mi sono occupato di gestire il repository, di spiegare con più dettagli le regole del gioco ai miei colleghi che non lo conoscevano e di dare consigli riguardo ad alcuni aspetti implementativi sollevati dagli altri membri.

Licia Valentini: Il lavoro di controller, per quanto non troppo articolato, è stato da una parte ostacolato dalla mancanza della view, ma favorito dal model, che era ormai ben definito a parte qualche piccolo dettaglio. Sono abbastanza soddisfatta di essere riuscita a dividere le varie componenti dell'applicazione, soprattutto per quanto riguarda quelle parti che non erano state approfondite a lezione, come la gestione dei file e dell'audio. Sicuramente la mia implementazione non è perfetta, ma ho cercato di renderla il più efficiente possibile. Mi sono occupata anche di disegnare su carta i vari pannelli dell'interfaccia grafica, per poter avere una base su cui lavorare.

Karl Darrajati: purtroppo non avendo il tempo a causa del lavoro, ho fatto slittare la consegna del progetto. Tuttavia ho trovato il lavoro dei miei colleghi praticamente finito, quindi è stato abbastanza veloce finire il lavoro. Mi sarebbe piaciuto avere il tempo di implementare qualche pattern di progettazione ma visto il poco tempo a mia disposizione ho optato per non farli. Inoltre non sono riuscito a trovare un modo di colorare i triangoli nella modalità Triangle anche cercando su internet. Nonostante cio sono abbastanza soddisfatto di come è venuto il progetto.