Relazione per Progetto "Java-Chess"

Marco Drudi 0000838384
Riccardo Foschi 0000843975
Matteo Scala 0000833015
Angela Cortecchia 838352
Marco Amadei 0000830817

17 ottobre 2019

Indice

Capitolo 1

Analisi

Il gruppo si pone l'obiettivo di realizzare una versione digitale del gioco degli scacchi, rielaborato in modo da sfruttare le conoscenze apprese durante il corso e dare la possibilità all'utente di apportare modifiche di vario tipo.

Nel gioco degli scacchi ogni ad ogni giocatore appartengono pedine o pezzi di un colore che sono disposte in una scacchiera 8x8, ogni pedina può essere mangiata da una pedina dell'avversario, ogni pezzo si muove in un certo modo e questo dipende dal tipo di pezzo. L'obbiettivo del gioco è mandare in scacco matto il re avversario, ovvero l'avversario non ha più mosse disponibili per non fare mangiare il re. Per maggiori informazioni si consulti: `https://it.wikipedia.org/wiki/Scacchi`

1.1 Requisiti

Requisiti funzionali

- All'avvio sarà presente un menù di gioco in cui scegliere la modalità di gioco, che può essere 1 Vs 1 oppure 1 Vs AI, inoltre sarà possibile scegliere una lingua fra quelle che verranno proposte.
- Il programma deve riconoscere automaticamente il vincitore, ci sarà anche la possibilità di dare la resa.
- Il programma deve tenere il punteggio della partita (con contatore delle pedine mangiate) e un punteggio generale nel caso voglia fare più partite.
- Nel programma deve essere presente un cronometro per tenere il tempo di gioco dall'inizio della mossa del giocatore.

- Sarà possibile personalizzare le pedine, la scacchiera ed il nome dei giocatori.
- Sarà presente un menù "aiuto" con le regole del gioco e una modalità per visualizzare dove la pedina può essere posizionata.

Requisiti funzionali opzionali

- Una modalità di gioco in cui ci sono 4 giocatori diversi, uno contro l'altro, con una scacchiera più grande ma con lo stesso numero di pedine.
- La presenza di uno "storico partite", con la quantità di partite vinte/perse, contro che avversario, la data e l'orario ed uno storico delle mosse.
- Una modalità di gioco computer Vs computer.
- L'aggiunta di effetti sonori.

1.2 Analisi e modello del dominio

Entità di base individuate nel dominio applicativo

- I pezzi che sono di diversi tipi e hanno un colore.
- La scacchiere che è formata da celle.
- Le celle sopra le quali possono essere posizionati pezzi.
- Le mosse che consistono nello spostamento di un pezzo da una cella all'altra.
- I giocatori.

La partita deve modificare le precedenti entità al susseguirsi dei turni.

Ad ogni giocatore può essere associata un'intelligenza artificiale, che produrrà i movimenti necessari al proseguimento della partita, o un utente umano.

L'applicazione avrà un menù che gestisca le impostazioni e l'avvio di nuove partite.

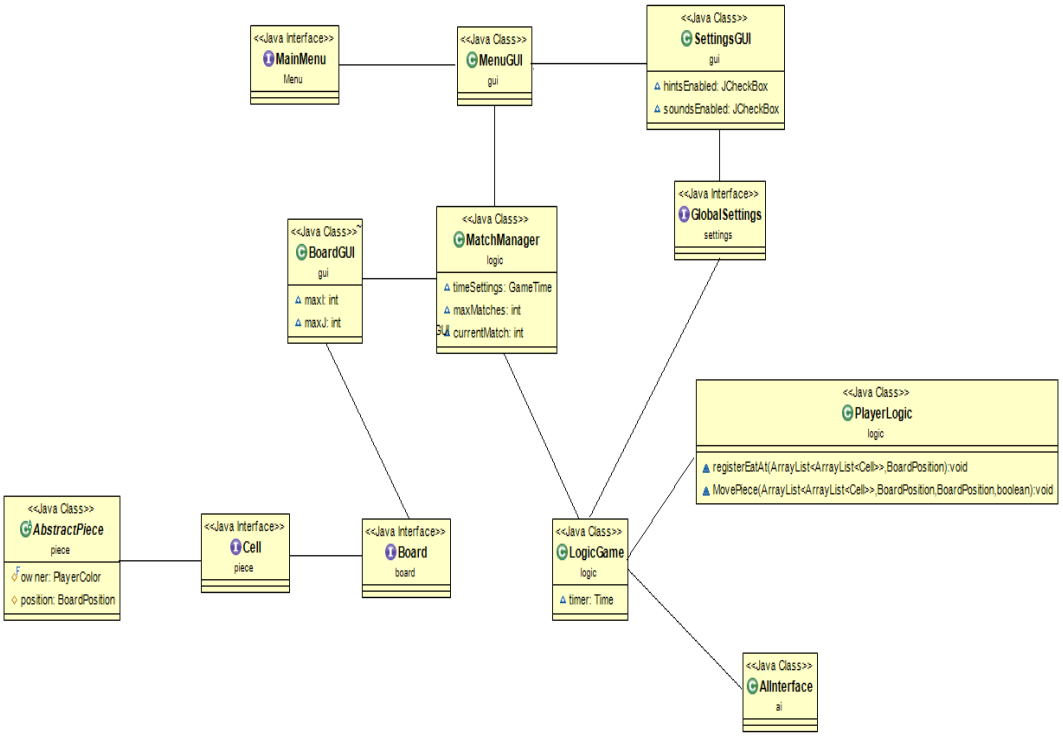
Capitolo 2

Design

2.1 Architettura

Per realizzare Scacchi abbiamo utilizzato il pattern MVC, e quindi dividere la logica dell'applicativo in Model, View, Controller.

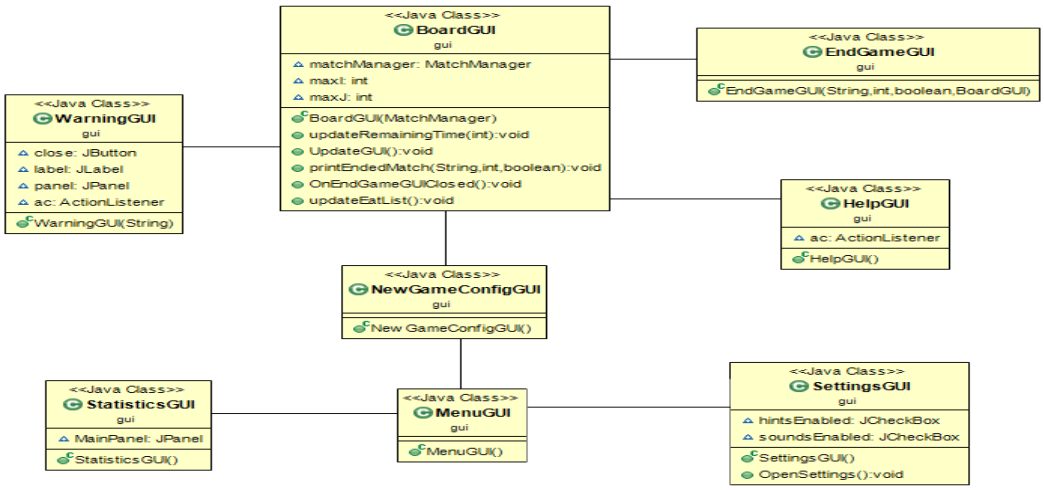
View gestisce interazione con utente e relativa interfaccia grafica. Control da all' utente la possibilità di modificare il Model. Model gestisce lo stato della scacchiera e dei pezzi.



La nostra applicazione è impostata utilizzando il pattern MVC in quanto, per eseguire una partita, l'utente è obbligato ad interfacciarsi ad un View, rendendo così indipendenti fra loro le classi generate in Model ed in Control. Tramite View abbiamo rappresentato graficamente tutti gli elementi per rendere verosimile la nostra applicazione. Abbiamo utilizzato le classi scritte in Control per gestire i turni di gioco e quindi aspettare dalla view un input dell'utente (diversamente nel caso si giocasse con AI). Dopo aver ricevuto una mossa in input il Control sviluppa il turno di gioco facendo gli opportuni controlli sulla correttezza di esso; nel caso non ci siano problemi , esso modifica il Model aggiornandolo.Infine aggiorna la view mostrando le modifiche fatte all'utente. Il Model è stato realizzato utilizzando varie interfacce con lo scopo di ottimizzare al meglio il codice e massimizzare il riuso. Viene,inoltre, data la possibilità di personalizzare una partita, e di customizzare pezzi e scacchiera. Modificando le impostazioni standard di una partita si va a modificare il Control mentre se si applicano modifiche stilistiche a pezzi e/o scacchiera, viene modificata la View. Il menù si occupa del cambiamento delle impostazioni e del corretto inizio di una partita.

2.2 Design dettaglio

Marco Amadei



Il mio ruolo nel progetto è stato parecchio flessibile. La necessità di riunire algoritmi e classi progettate da membri diversi all'interno di un unico software ha richiesto gran parte dei miei sforzi nel lavoro. Tuttavia ho curato anche

una serie di componenti e di mansioni specifiche. In particolare i miei ruoli hanno riguardato:

- Progettazione della GUI e l'implementazione della maggior parte delle grafiche e delle interfacce utente di gioco.
- Supervisione e direzione sull'integrazione di scacchiera, pezzi e logica di gioco insieme.
- Costruzione di una serie di classi di supporto allo sviluppo.
- Gestione e inclusione delle risorse garantendone la funzionalità su Windows e Linux anche in formato JAR.
- Gestione dell'istanziazione delle partite basate su set di regole ed impostazioni selezionate tramite GUI.

La primissima fase di sviluppo si è concentrata nella suddivisione del lavoro. La sincronizzazione delle mansioni è stata di molto semplificata grazie al software Git.

Una volta ottenute le prime versioni embrionali delle strutture minime del gioco, è stato possibile costruire i primi menu grafici e le prime interfacce grafiche per la scacchiera.

La costruzione dei menu è stata possibile grazie alla libreria grafica Swing, che ho scoperto essere particolarmente comoda specialmente quando le grafiche dovevano rispettare posizioni ben precise o addirittura tabellari. Ogni menu, non solo quelli di mia progettazione, sono stati costruiti tramite l'estensione di oggetti JFrame. Inutile elencare tutte le classi relative alla grafica da me realizzate, si possono trovare facilmente all'interno del progetto, nel package gui. In supporto ad oggetti con elementi testuali offerti da Swing, ho creato all'interno del package gui anche un tipo di classe TextToTranslation con le relative estensioni agli oggetti JLabel, JButton e JCheckbox.

La funzione di queste classi è di tenere aggiornato l'oggetto di riferimento passato nel costruttore con la traduzione corretta. Ciò serve a poter aggiornare immediatamente tutti i testi a schermo con la traduzione corretta in caso di cambio di lingua. A parte gli aspetti tecnici di queste classi, il loro funzionamento si riduce all'utilizzo della classe di Angela 'Translations' la quale, dato un id specifico di una parola o frase in formato stringa, ritorna la sua traduzione nella lingua corrente. In pratica, ogni qual volta che dal menu impostazioni la lingua viene cambiata, tutti i testi a schermo vengono modificati dal TextToTranslation a loro associati con la traduzione corretta.

Altra classe molto importante contenuta nel progetto è la classe MatchManager. Durante la creazione di una partita dal menu NewGameConfigGUI, un oggetto di tipo MatchManager contenete le informazioni inserite viene creato. Questo oggetto si occupa di istanziare tutti gli elementi necessari a generare e gestire un match, come la logica e la scacchiera, e ne imposta i parametri secondo le decisioni del giocatore. MatchManager si occupa anche di fare partire il match successivo, se vengono fatte più di una partita, e di chiamare le API di Angela per poter salvare i risultati a fine partita.

Una caratteristica interessante del progetto sono gli skin pack; sarebbe infatti teoricamente possibile aggiungere in fretta pacchetti di icone per la personalizzazione della scacchiera. Per fare ciò è necessario creare una cartella ¡nomepacchetto¿ all'interno della cartella resources nel progetto. All'interno del pacchetto vanno inserite, per ogni colore, tutte le sprite delle icone, chiamandole secondo lo standard riportato di seguito:

- b_Bishop.png - icona per l'alfiere nero (w_Bishop.png per il bianco)
- b_Pawn.png - icona per il pedino nero (w_Pawn.png per il bianco)
- b_King.png - icona per il re nero (w_King.png per il bianco)
- b_Queen.png - icona per la regina nera (w_Queen.png per la bianca)
- b_Knight.png - icona per il cavallo nero (w_Knight.png per il bianco)
- b_Rook.png - icona per la torre nera (w_Rook.png per la bianca)

Se il progetto verrà eseguito poi dall'IDE, automaticamente il nuovo pacchetto skin sarà trovato e configurabile come pacchetto da usare in una partita. Se invece si intende lanciare il gioco da file JAR, sarà necessario aggiungere all'array 'knownJARSkinFolders' della classe 'SkinPackFinder' il nome della cartella contenente skin che è stata creata. In ogni caso questa procedura è consigliata solo agli utenti più esperti.

Ultimi, ma non meno importanti, alcuni elementi sono stati necessari da realizzare per poter utilizzare path di file system su diversi OS e in diverse situazioni. CrossPlatformResources ne è un esempio; Si tratta di una classe che ho scritto per tradurre path generiche e standard del progetto di eclipse di Windows, in path compatibili con Linux ed addirittura con il file JAR finale. Le risorse grafiche e sonore ed i loro percorsi infatti non sono da dare per scontati in un progetto cross platform, specialmente in questo caso dove il gioco è pensato per essere compattato in un file JAR eseguibile.

Con tutti gli elementi fino a qui descritti a mio parere di può avere una visione abbastanza chiara di come il gioco sia stato implementato.

Marco Drudi

La mia parte di progetto era incentrata sul model più precisamente sull'implementazione dei pezzi del gioco (re, regina, alfiere ecc..), le celle della scacchiera e di una piccola parte di view.
Per quanto riguara la creazione dei pezzi durante la fase di progettazione è risultato conveniente utlizzare il pattern template method in quanto l'implementazione di alcuni metodi era diversa da pezzo a pezzo mentre per altri era uguale.
Ogni pezzo ha un proprietario (bianco o nero) e una posizione nella scacchiera e tramite un metodo che prende come parametro la scacchiera è possibile trovare le possibili mosse che questo pezzo può effettuare.
L'interfaccia Piece viene implementata dalla classe AbstarctPiece e tutte le classi dei pezzi (King, Queen, Pawn, ecc..) estendono la classe AbstracPiece.

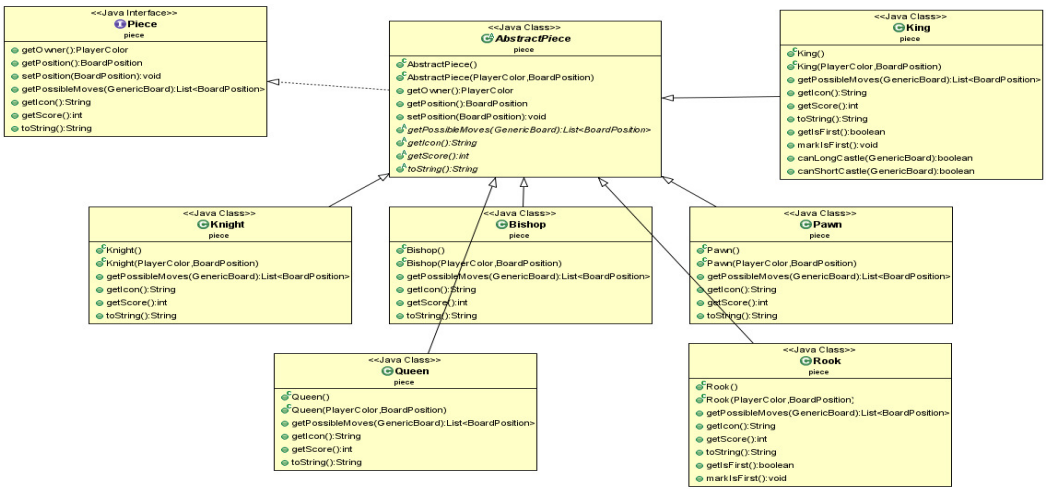


Figura 2.1: Legami tra le classi dei pezzi della scacchiera.

Per quanto riguarda la posizione di un pezzo nella scacchiera è stata creata una classe ad hoc BoardPosition.
Per quanto riguarda le celle della scacchiera è stata creata l'interfaccia Cell e la classe CellImpl che implementa i metodi dell'interfaccia Cell.
Su ogni cella può essere posizionato un pezzo e la sua posizione nella scacchiera, la scacchiera è formata da celle.

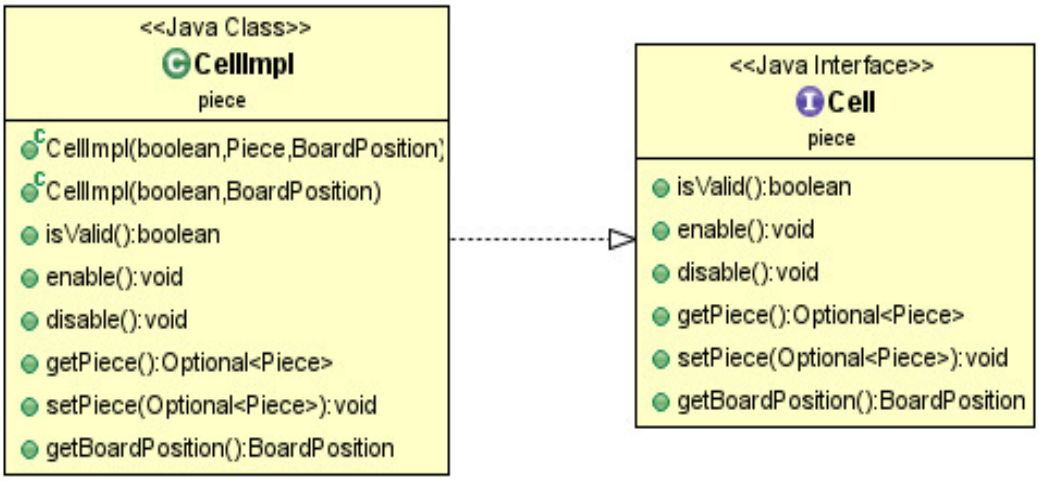
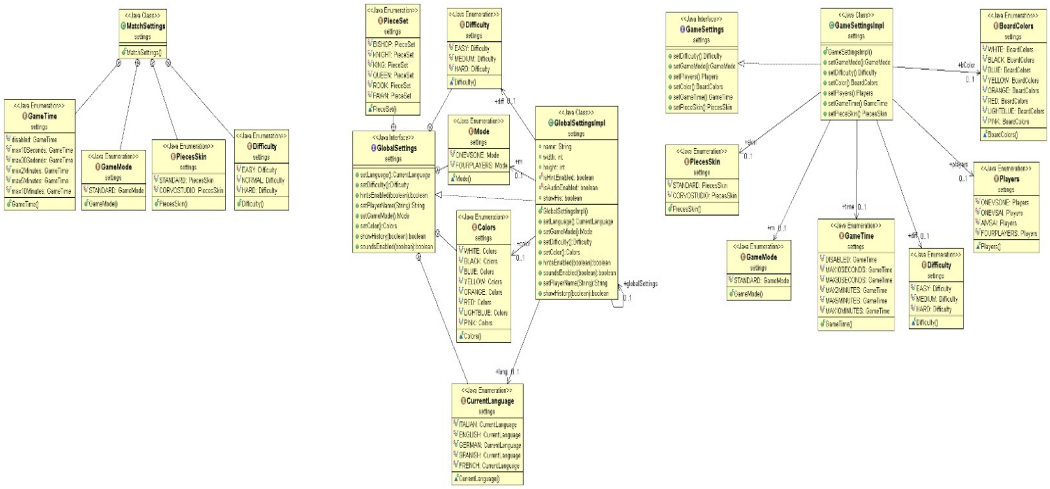
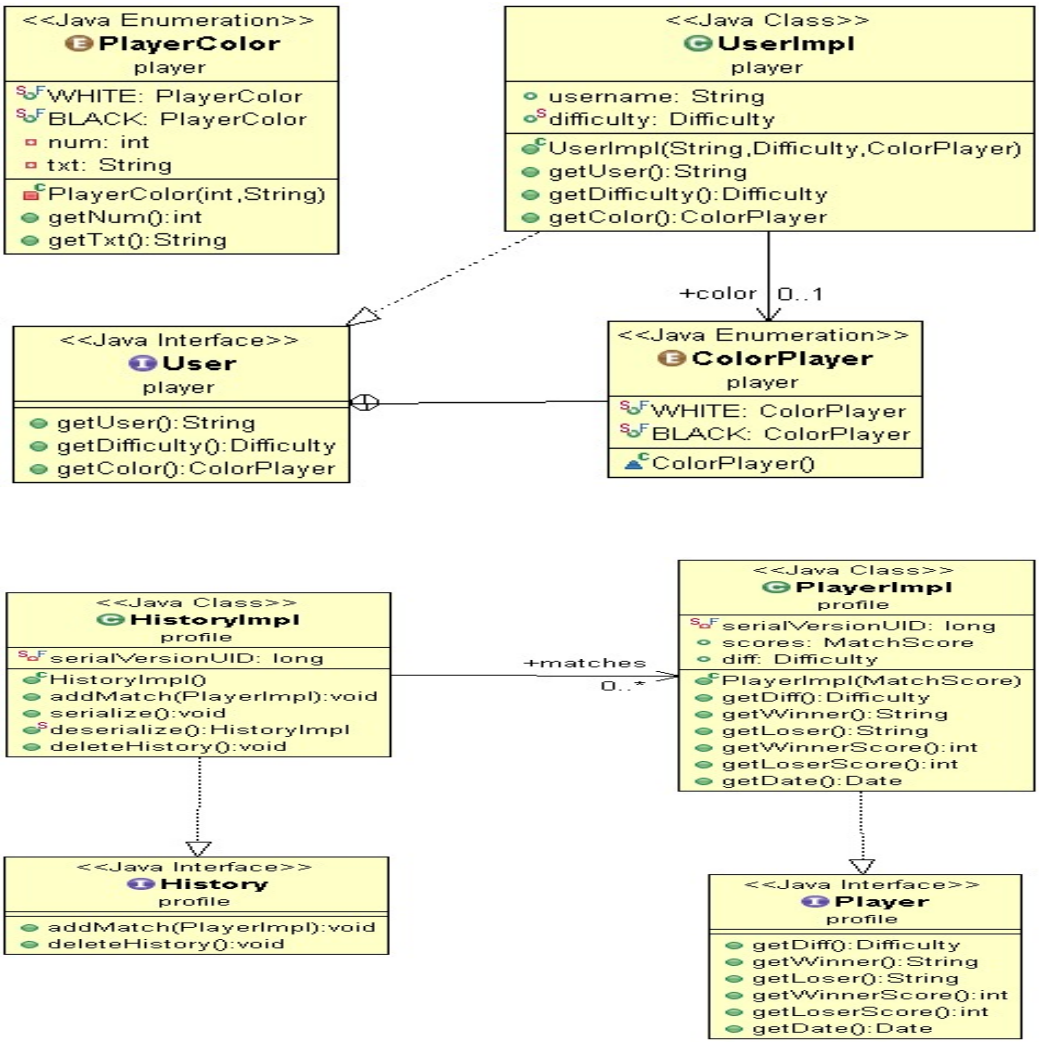


Figura 2.2: Legami tra l'interfaccia della cella della scacchiera e la classe che la implementa.

Ho anche realizzato una piccola parte di GUI che spiega all'utente come utilizzare la scacchiera durante una partita (HelpGUI).

Angela Cortecchia





La mia parte di progetto riguardava l'implementazione delle impostazioni, del menù e del profilo dell'utente, con tutte le relative traduzioni.
Il profilo dell'utente è stato creato per permettere di salvare informazioni relative all'utente giocatore, in modo tale da salvare lo storico delle varie partite effettuate.

La parte di grafica che ho implementato interamente è "StatisticsGUI", classe che mostra lo storico delle partite effettuate, partendo da un piccolo metodo in "MenuGUI" viene aperta la classe, che a sua volta avrà modo di scegliere di mostrare lo storico ed eventualmente di eliminarlo.
Lo storico viene salvato su un file.
La classe che ho creato per gestire le traduzioni è "Translations", composta da un enum che definisce e salva la lingua scelta nelle settings e da un doppio switch che prende in input "textID" per riconoscere il testo da tradurre,

all'interno del case con tale textID è presente un altro switch che, in base a case con la lingua salvata, dice cosa scrivere al posto di textID.

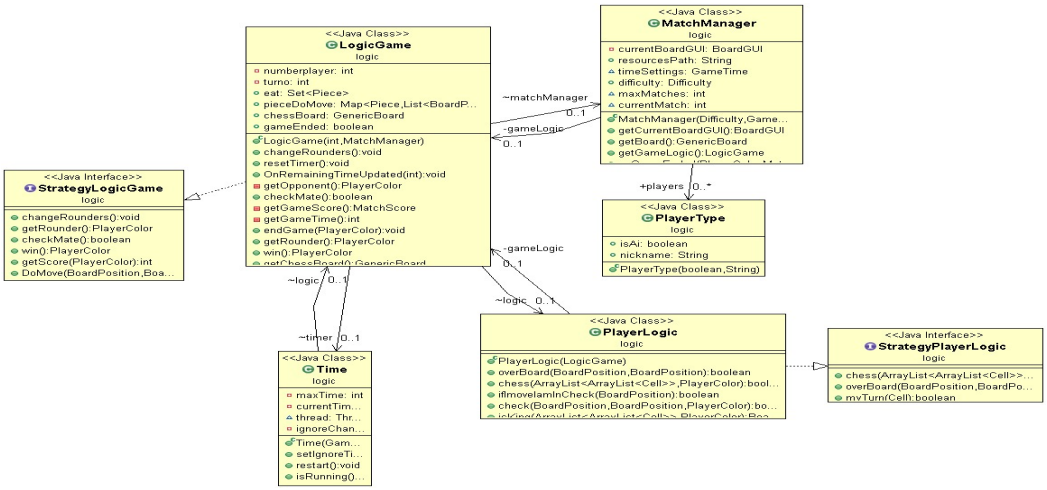
"PlayerImpl" è una classe serializzabile adibita ad ottenere nome del vincitore e del perdente, con annessi i loro punteggi.

Inoltre restituisce la data di quando è stata svolta la partita e la difficoltà scelta per l'AI.

Le informazioni come nome e punteggio le prende dalla classe "MatchScore", anch'essa serializzabile.

Ho creato anche la classe serializzabile "HistoryImpl", con annessa interfaccia e test JUnit, questa classe si occupa di aggiungere match allo storico, serializzare e deserializzare le informazioni del giocatore attraverso FileInput/OutputStream e ObjectInput/OutputStream.
Inoltre è presente un metodo per cancellare completamente lo storico in maniera permanente, per poi andare a sovrascrivere future partite.

Riccardo Foschi



La mia parte di progetto era, principalmente, la realizzazione delle classi GameLogic , e PlayerLogic. Esse hanno il compito di gestire le regole di gioco e di compiere lo spostamento effettivo (o non permettere di effettuarlo nel caso esse non siano rispettate) dei pezzi. In oltre di mia competenza erano anche le classi Audio,Time,WarningGUI e MatchScore.

GameLogic si occupa di gestire le regole di una partita effettuando controlli sullo stato di essa.Questa classe fa parte del "Control" utilizzato nel pattern MVC poichè essa si interfaccia direttamente con la "View"(scacchiera a video) e con il "Model"(i pezzi). Specificatamente si occupa di:

- Inizio/Fine Partita
- Cambio turno dei giocatori
- Scacco Matto
- Punteggio finale ottenuti dai giocatori
- Creazione di un PlayerLogic
- Comunicare ad PlayerLogic le modifiche inviatogli dalla GUI (spostamenti)
- Interfacciarsi con AI
- Creazione del thread Time.

PlayerLogic è una classe nata con lo scopo di eseguire controlli specifici per ogni tentativo di mossa da parte dell'utente. Anch'essa fa parte di "Control" in quanto i suoi compiti specifici sono quelli di:

- Controllare se i valori ricevuti siano corretti
- Controlli opportuni riguardanti allo stato della scacchiera
- Spostamento delle pedine e quindi modifica del "Model"

Audio è una classe utilizzata per la riproduzione di suoni implementata utilizzando le librerie di JavaSwing. In particolare utilizzata nella "View" per accompagnare musicalmente l'utente quando intera con il gioco. I suoi metodi hanno il "goal" di :

- Avviare un file con estensione .wav (file audio)
- Stoppare riproduzione di file con estensione .wav

Time è un thread che simula un timer, quindi ad ogni secondo incrementa una variabile fino ad arrivare ad un tempo massimo. Viene visualizzato nella "View" al contrario (il timer visualizzato si decrementa). Il suo scopo è quello di poter dare un tempo limite pre-Impostabile ad ogni mossa, scaduto quello si cambia il turno.Questa thread viene messo in esecuzione esclusivamente da LogicGame.

La classe WarningGUI è una parte di "view" utilizzata per dar messaggi in output all'utente, ad esempio la non possibilità di spostare un pezzo.Anch'essa implementata avvalendosi delle librerie di JavaSwing.

MatchScore è una classe creata appositamente per salvare i risultati a fine partita del risultato ottenuto dai due giocatori che successivamente andranno a formare una classifica.Essa fa parte del "Model", e viene istanziata da LogicGame.

Matteo Scala

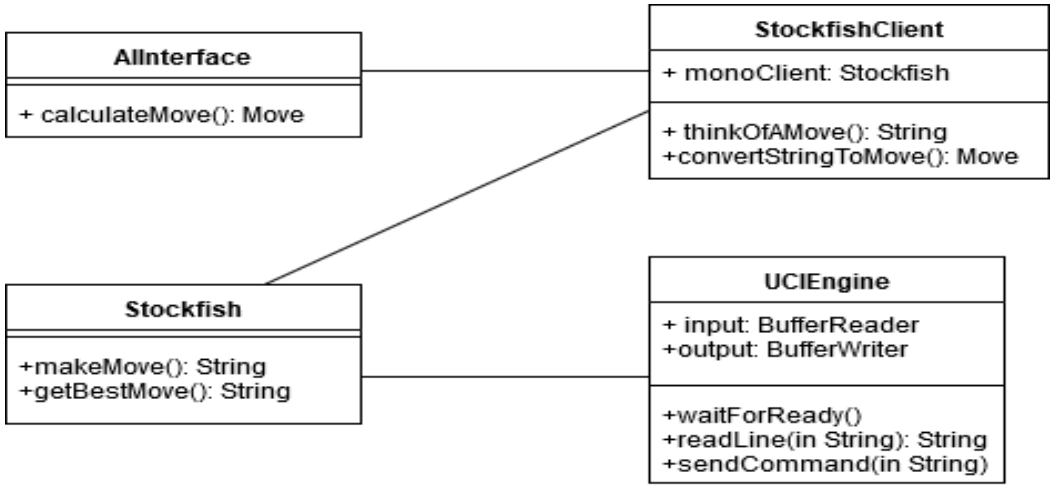


Figura 2.3: Legami tra le classi che gestiscono l'intelligenza artificiale.

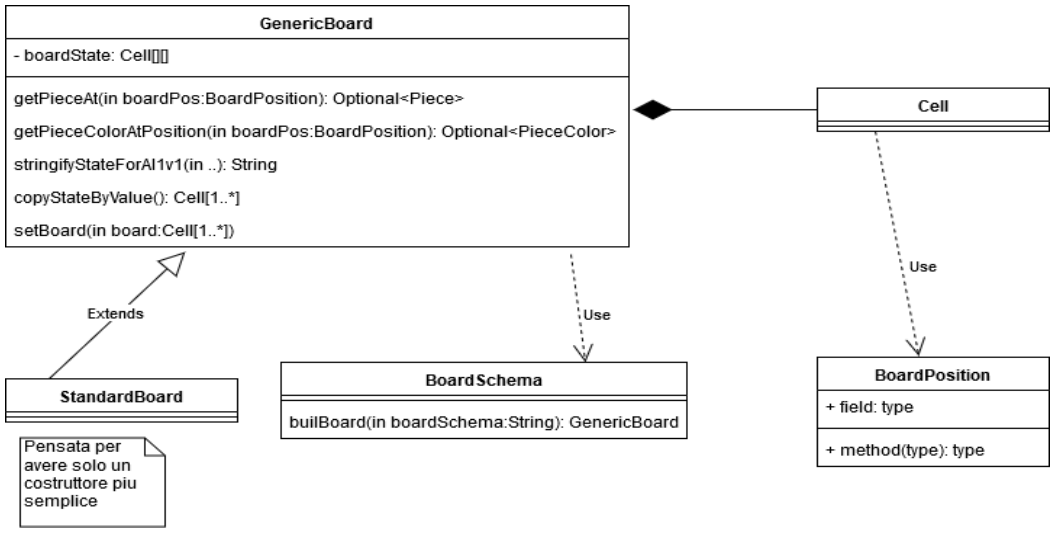


Figura 2.4: Legami tra le classi che gestiscono la scacchiera.

Il mio compito si divideva principalmente in due parti. La prima consisteva nello sviluppare una classe che potesse gestire la scacchiera e fare da ponte tra i pezzi di gioco e la logica. La seconda parte, doveva gestire l'intelligenza artificiale.

Per la prima parte ho cercato di renderla più flessibile possibile. Ho quindi deciso di creare un piccolo standard che permettesse di rappresentare

la posizione di pezzi su scacchiere di qualsiasi tipo tramite stringhe. La gestione di questo tipo di stringhe veniva affidata alla classe *BoardSchema* che aveva lo scopo di convertire queste stringhe in liste bidimensionali contenenti celle con pezzi all'interno.

Queste matrici andavano a formare lo stato, *model*, della scacchiera nella classe *GenericBoard* la quale è fornita di tutti un set di funzioni che permettono di alterarne lo stato (alcune di esse sono state poi spostate nella classe *LogicGame* per questioni di organizzazione e quindi deprecate in *GenericBoard*). Questa classe era poi servita alla parte di *controll* per tutte le elaborazioni.

Nella seconda parte del lavoro ho preferito concentrarmi sul dare semplicità all'uso della AI.

Una volta implementata ho deciso di raggruppare tutto in una classe molto semplice, *AIInterface*, che fornisce solo due metodi. Uno con il quale mostrare una GIF nei tempi morti durante i quali l' AI calcolava la mossa. Il secondo che serviva a chiedere all' AI il calcolo di una mossa.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per i test più semplici è stato utilizzato JUnit5. Alcuni esempi di test effettuati in questo modo sono stati fatti su:

- Possibili mosse eseguili con un dato pezzo.
- La condizione di scacco con una data configurazione della scacchiera.
- La corretta gestione e corretto posizionamento di pezzi alla creazione della scacchiera.

Altri test, in particolare quelli riguardanti l'organizzazione della grafica sono stati effettuati manualmente durante lo sviluppo.

3.2 Metodologia di lavoro

A Marco Amadei è stato richiesto inizialmente la generazione di alcune interfacce grafiche di prova da cui ora derivano quelle effettivamente in gioco.

A Matteo Scala è stato richiesto la scrittura di classi relative ad oggetti per creare una scacchiera virtuale con specifiche proprietà e metodi. Gli ho anche assegnato il ruolo di implementare API per richiamare una AI in modo facile e stabile.

A Marco Drudi è stato assegnato la creazione dei pezzi e delle loro proprietà, come comportamenti di movimento, e informazioni.

A Foschi Riccardo è stato richiesto la creazione utilizzando le strutture appena citate di una logica di gioco che permettesse di fare operazioni controllate sui pezzi e i loro spostamenti.

Mentre ad Angela Cortecchia è stato chiesto di occuparsi di aspetti fondamentali ma meno circoscritti all'interno di una partita, come la gestione delle impostazioni, parziale gestione del menù iniziale, delle traduzioni e dei salvataggi dei record.

Tutti i membri in più occasioni si sono esposti a mansioni che vanno oltre le richieste iniziali fatte per poter ricoprire ambiti importanti che non erano stati assegnati inizialmente. Inoltre la collaborazione è stato un elemento chiave, specialmente dove le abilità già in possesso dei membri poteva velocizzare l'aiuto e l'apprendimento degli altri.

La sincronizzazione delle mansioni è stata di molto semplificata grazie al software Git.

3.3 Note di sviluppo

Marco Amadei

Oltre allo sviluppo della mia parte di software, mi sono dedicato anche a trovare, riadattare, o nel caso di alcuni elementi quali l'icona del programma, di creare le varie risorse grafiche e sonore presenti nel gioco.

Marco Drudi

- Optional, utilizzati in molte classi.
- Stream, utilizzati nel metodo getPossibleMoves di alcuni pezzi.
- Lambda expression, utilizzati nel metodo getPossibleMoves di alcuni pezzi.

Angela Cortecchia

Le feature avanzate del linguaggio Java che ho utilizzato sono:

- Serializzazione: è stata utile nella creazione e gestione dello storico partite.
- HTML: linguaggio che ho integrato in alcune parti della grafica per impostare font e colore.
- Java Swing: libreria utilizzata per gestire le parti di grafica del gioco.
- JUnit 5: utilizzato per la creazione di un test relativo ad una classe da me implementata.

Riccardo Foschi

Optional, utilizzati in alcune classi

Matteo Scala

Lo sviluppo dell' AI non è stato fatto da me.

Il mio compito è stato solo quello di trovare una libreria utilizzabile all'interno di un programma scritto in Java. Avevo la possibilità di cercare qualche servizio online che fornisse, tramite chiamate HTTP, questa funzionalità, ma spesso questi servizi richiedono autenticazione e soprattutto una costante connessione a internet, cosa che volevo evitare. Ho quindi optato per qualcosa che potesse funzionare in locale.

Su GitHub ho trovato un progetto che permetteva di interagire con degli eseguibili che contenevano il motore di scacchi StockFish. Il progetto forniva solo una gestione asincrona, quindi mi sono attivato per poter aggiungere la possibilità di fare interazioni sincrone, più comode per le nostre necessità.

https://github.com/Niflheim/Stockfish-Java

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Marco Amadei

Lavorando ogni giorno con C#, la programmazione ad oggetti non è nuova per me; Il lato logico e algoritmico della mia parte è stato molto piacevole e senza particolari interruzioni. Lavorare su aspetti come path che possano essere cross platform è stata una sfida più dura, i cui frutti sono stati un ampliamento in generale della conoscenza di in che modo i linguaggi virtualizzati ed ad alto livello possono risolvere questo genere di problemi. Nonostante lavori quotidianamente con motori grafici 3D, l'implementazione dell'interfaccia tramite Swing è stata una novità per me. E' stato interessante ed utile scoprire un modo diverso (e come già definito, più "tabellare") della costruzione di interfacce utente. Lavorare in gruppo in oltre ha ampliato di molto le mie abilità nella sincronizzazione delle mansioni, nella quale devo dire essermi trovato molto bene, in quanto comunque in gruppo con gente molto disponibile e intraprendente.

Marco Drudi

Sono parzialmente soddisfatto di come è stato svolto il progetto, forse avrei dovuto partecipare più attivamente nella parte di progettazione e mi rendo conto che certi aspetti finali non sono stati realizzati al meglio, forse proprio queste cose sarebbero da migliorare in futuro. Nonostante questo però questo progetto mi è stato particolarmente d'aiuto per notare gli aspetti sia positivi che negativi del lavoro in team.

Angela Cortecchia

Personalmente sono soddisfatta del risultato finale del progetto nonostante le varie difficoltà incontrate, non è perfetto ma penso che rispecchia ampiamente la proposta inoltrata.

La creazione del progetto mi ha aiutata a comprendere meglio molti aspetti della programmazione Java, soprattutto a livello di grafica.

La programmazione in team non mi è sembrata un problema siccome ho riscontrato tanta disponibilità ad aiutare e ad apportare eventuali modifiche da parte di tutti i membri.

Spero che in un futuro si possano apportare modifiche costruttive al progetto per renderlo ulteriormente "user friendly" su qualsiasi tipo di piattaforma.

Riccardo Foschi

Questa è stata la mia prima esperienza di progetto utilizzando un linguaggio di programmazione ad oggetti e tuttavia posso ritenermi soddisfatto del risultato ottenuto. Inizialmente ho riscontrato qualche problema perché ,avendo mansioni principalmente logistiche, non riuscivo ad eseguire test ed a capire se ciò che stessi scrivendo fosse realmente funzionante. Risolto questo problema insieme al mio team siamo riusciti a completare correttamente l'applicativo. Con questo progetto penso di aver acquisito molte capacità nel lavorare in team e di aver ampiamente migliorato le mia capacità da sviluppatore software.

Matteo Scala

Sono globalmente soddisfatto del risultato prodotto, ma credo che con una migliore organizzazione iniziale si sarebbero potuti raggiungere risultati ancora migliori.

Questo progetto mi ha aiutato a capire le difficoltà del lavoro di gruppo e l' importanza di strumenti come Git, al di fuori del semplice utilizzo come sistema di controllo di versione.

Credo di aver anche meglio compreso i punti di forza della progettazione ad oggetti e di essere ora più in grado di capire quando è meglio farne uso.

4.2 Difficoltà incontrate e commenti per i docenti

Marco Amadei

Ci sono state svariate difficoltà incontrate durante la mia parte di sviluppo. Ho inizialmente sottovalutato l'aspetto di coordinazione dei ruoli, il che ha portato talvolta alla creazione di classi sbagliate, difficili da gestire o addirittura inutili e successivamente rimosse. Con il proseguire del tempo, quando finalmente le prime build funzionanti hanno iniziato a dare risultati, una riorganizzazione ha rimesso stabilità nel lavoro di tutti. Per quanto riguarda il mio operato, ho trovato difficoltà in alcuni ambiti, ne citerò un paio:

- La gestione dei percorsi; Specialmente all'interno del file JAR, la gestione dei percorsi è diversa da quella che sarebbe nel progetto Eclipse di Windows o Linux. Per poter leggere i file, specialmente gli eseguibili della AI, ho adottato un metodo che consiste nel creare file temporanei delle risorse del progetto (Solamente però se il gioco riconosce di essere eseguito tramite un file JAR). Infatti se eseguito da JAR il progetto può sembrare richiedere un caricamento piuttosto lento (circa 2-3 secondi più del previsto). Ciò infatti dipende dalla necessità di dover prendere il controllo della creazione dei file temporanei basati su risorse interne al pacchetto. In ogni caso i problemi di prestazioni spariscono immediatamente dopo al completamento del caricamento della prima risorsa.

- La gestione dei pacchetti di skin; La ragione per cui in caso di aggiunta di nuovi pacchetti skin, prima di esportare il JAR sia necessario aggiungere il nome della cartella all'array delle cartelle conosciute, sta nel fatto che dentro al file compresso è difficile (se non a volte limitato) l'uso delle directory e dei path. La necessaria creazione di template ne è la dimostrazione.

Marco Drudi

Credo che questo sia il corso più bello affrontato finora, ma anche uno dei più impegnativi in quanto la programmazione ad oggetti e risultata per me uno scoglio. Nonostante questo ho potuto apprendere cose molto interessanti e aspetti nuovi della programmazione. Mi sarebbe piaciuto fosse stata affrontata nel corso la parte di JavaFX anche se non è stata utilizzata per il progetto.

Riccardo Foschi

Inizialmente, il lavoro in team non è stato dei migliori, causato dalla mia scarsa autonomia con GIT e dalla distanza fra i vari componenti, ma superato questo problema siamo riusciti a svolgere il progetto al meglio delle nostre possibilità. Un'altra difficoltà non trascurabile è stata la realizzazione della relazione. Riguardo al corso, penso sia uno dei più coinvolgenti, mi sarebbe piaciuto affrontare non solo JavaSwing ma anche JavaFX.

Appendice A

Semplice guida utente

All'avvio, il gioco presenta tre tasti che permettono di proseguire nell'utilizzo e uno per chiudere il programma. Dal tasto delle impostazioni si accede ad una interfaccia con alcune opzioni per settare dati globali nell'applicazioni. Con il tasto dei dati si accede ad un semplice storico delle partite.

Alla parte principale del programma si accede con il tasto 'Nuova partita'. da qui si possono impostare gli aspetti della partita (difficoltà, nome dei giocatori, tipi di giocatori, numero di partite). Con l' ultimo tasto, 'Nuova partita', si avvia definitivamente la partita.