StoneBreaker

Object Oriented Programming project.

Preparato da: Agatensi Luca, Falco Matteo, Pucci Federico.

Professore: Viroli Mirko.

Assistenti: Casadei Matteo, Croatti Angelo, Pianini Danilo.

CHAPTER 1

ANALISI

1.1 Requisiti

L’applicazione software che ci siamo posti di creare è un gioco di tipo rompicapo, ispirato a Bejeweled, il quale si presenta con un campo di gioco quadrato in cui sono disposte, su righe e colonne, diversi tipi di pietre le quali dovranno essere allineate e quindi eliminate per ottenere un punteggio: scopo del gioco è quindi selezionare una pietra per spostarla con una adiacente in modo tale da allinearne almeno tre dello stesso tipo; almeno tre pietre allineate dello stesso tipo comporta la loro eliminazione dal campo di gioco e quindi l’assegnazione del punteggio relativo. Il giocatore avrà per cui, come obiettivo, quello di raggiungere il maggior punteggio possibile.

Requisiti concordati

- Obbiettivo base dell’applicazione è quello di creare un puzzle game in cui l’utente sia in grado di cimentarsi in un livello classico a tempo, con 4 tipi di pietre e con la possibilità di vedere salvato il proprio punteggio.

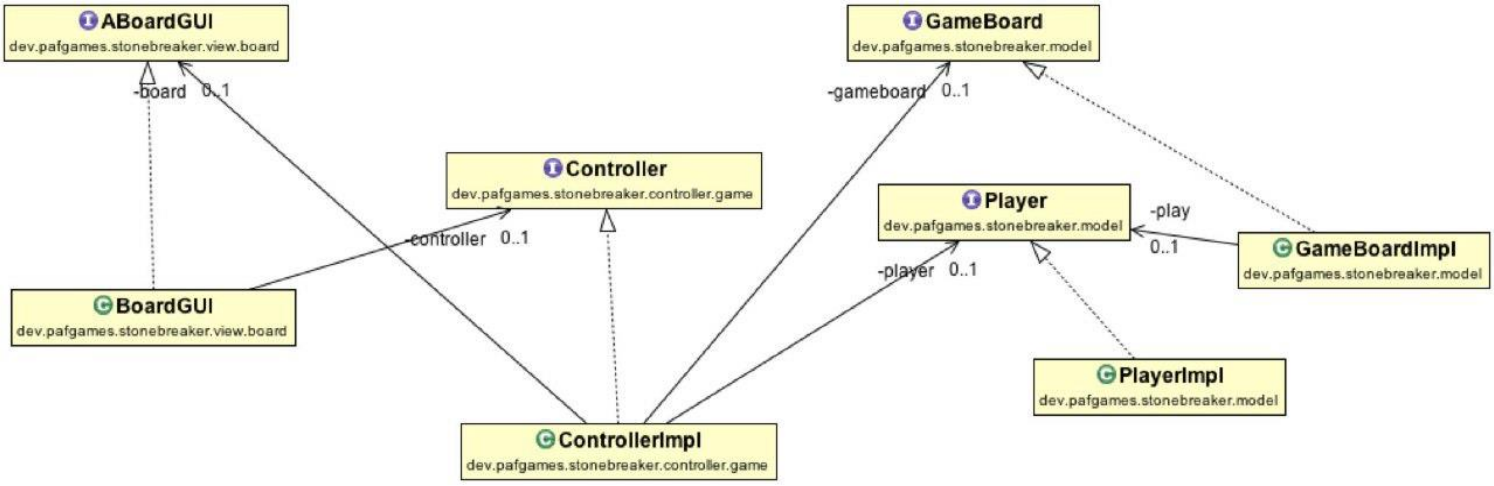
- Le features opzionali dell'applicazione consistono nel fornire all'utente la possibilità di variare tra 3 diversi livelli di difficoltà e tra 2 diverse modalità di gioco e infine nell’utilizzo di un superpotere i grado di fornire un punteggio aggiuntivo.

1.2 Analisi e modello del dominio

All’avvio dell’applicazione l’utente dovrà essere in grado di scegliere tra due diverse modalità di gioco: Classica, che prevede lo scadere della partita al termine del tempo di gioco o Arcade dove il termine è fissato da un certo numero di mosse. Dopodiché l’utente dovrà scegliere tra tre diversi livelli di difficoltà che permettono di regolare il numero di pietre in gioco. Il giocatore dovrà per cui selezionare una pietra e scambiarla con una adiacente (ma non in diagonale) in quanto almeno tre pietre uguali vicine comportano la loro eliminazione e l’assegnazione del punteggio relativo: maggiore è il numero di pietre vicine maggiore è il punteggio relativo. Durante il gioco, l’eliminazione di 5 pietre uguali comporta l’attivazione di un superpotere che permette di eliminare un quadrato di dimensione 3x3 intorno alla pietra selezionata successivamente e conseguire il relativo punteggio. L’utente sarà in grado di fermare il gioco e farlo ripartire da dove lo aveva lasciato oppure scegliere di tornare indietro per cambiare modalità di gioco. Infine i risultati delle varie modalità saranno salvati e mostrati al termine di ogni partita.

Tra le difficoltà principali ci saranno quelle di avere la possibilità di fermare il gioco e di farlo ripartire, gestire le varie combinazioni sul campo di gioco a seguito di una mossa e la possibilità di estendere il gioco per eventuali aggiunte di modalità di gioco in futuro.

La figura qui di seguito serve come ulteriore spiegazione al modello del dominio e le interazioni tra le entità principali.



Schema UML relativo all’applicazione nella sua suddivisione MVC

CHAPTER 2

DESIGN

2.1 Architettura

Il gruppo, al fine di implementare le suddette specifiche, ha adottato l'architettura MVC.

- Model: ha il compito di modellare i dati e la logica del applicazione, ovvero la matrice di gioco, le gemme, i blocchi e tutte le possibili modifiche della disposizione di tali elementi sulla matrice, il livello di difficoltà del gioco, il punteggio effettuato dal giocatore e i contatori di secondi/movimenti rimasti al giocatore.

- View: ha il compito di gestire la visualizzazione grafica dell'applicazione. In particolare gestirà la visualizzazione di gemme, blocchi, powerup, indicatori di gioco e menù interattivi. Inoltre gestirà la disposizione delle proprie entità in base agli aggiornamenti chiamati dal Controller.
  Ci si prefigge il compito particolare di rendere piacevole e interattiva la comunicazione tra utente e applicazione.

- Controller: ha il compito di osservare notifiche provenienti dalla View e di conseguenza aggiornare il Model per ottenere le informazioni necessarie alla View stessa, inoltre si occupa di controllare se nel campo di gioco, in seguito di una mossa, sono presenti combinazioni che possono portare all'eliminazione delle pietre e alle relative assegnazioni di punteggio; infine si occupa di creare classifiche dove sono mantenuti i punteggi.

2.2.1 Model

Il modello consiste principalmente di una board contenente Optional<**Stone**>, ovvero le varie caselle della matrice di gioco in un determinato momento possono anche essere vuote. La classe contenente la matrice (**GameBoardImpl**) fornisce anche una serie di metodi per la modifica delle Stone e per le interazioni tra una Stone e l'altra.

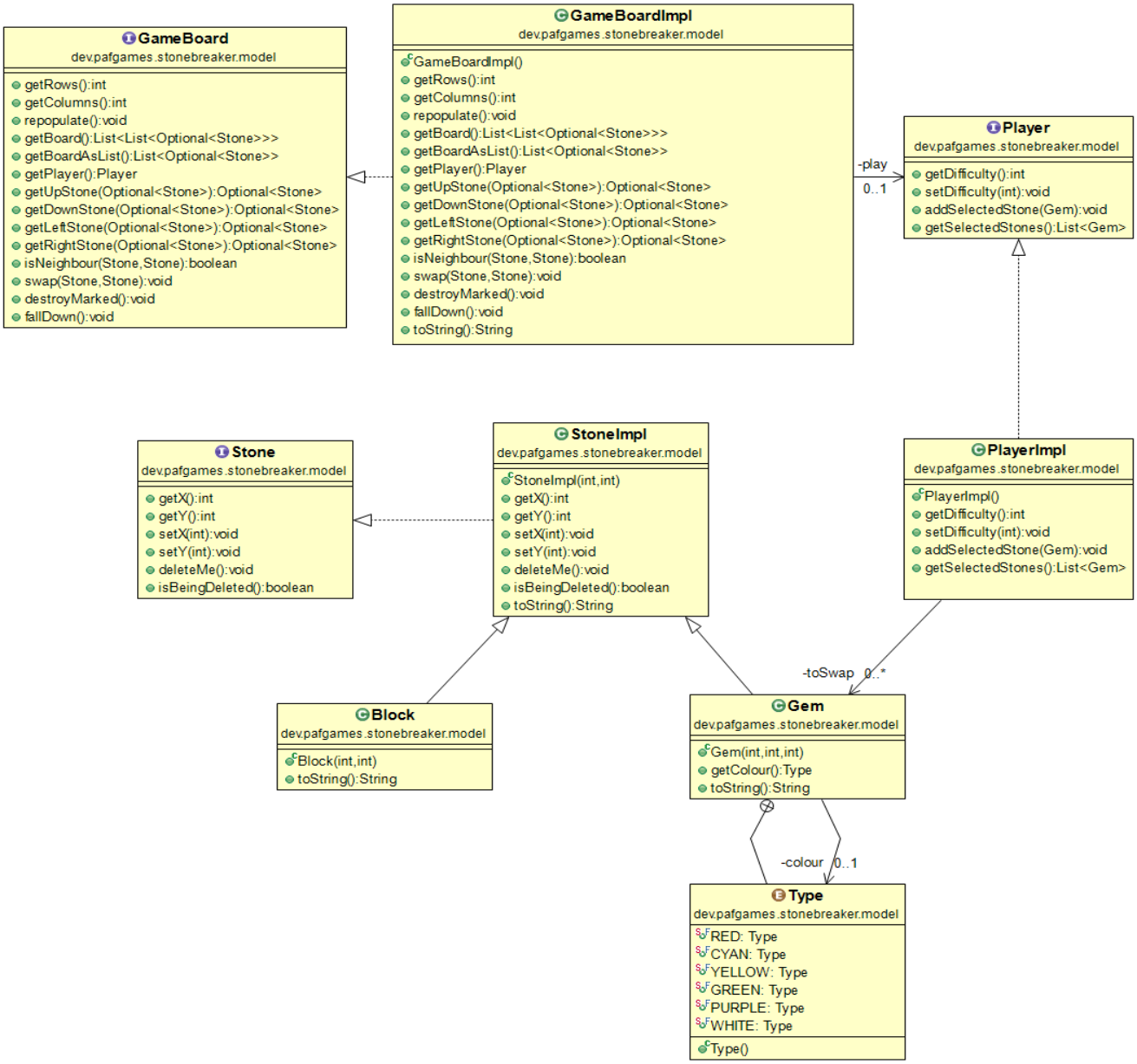
Le Stone si concretizzano poi in **Gem**, ovvero le gemme che il giocatore può muovere sulla griglia e con cui può fare punti qualora tre o più gemme siano allineate, e in **Block**, blocchi il cui unico movimento è legato alla scomparsa di altre Stone sottostanti e la cui eliminazione avviene solo se si vengono a trovare nell'ultima riga della board.

Avendo usato l'oggetto Stone come elemento della matrice questo permetterà, qualora si volesse creare una nuova tipologia di oggetti stanti nella board, di farlo liberamente assicurandosi che questi estendano dalla classe Stone, la quale si limita a fornire le coordinate di un elemento nella griglia di gioco e se tale elemento dovrà essere eliminato perché sono state soddisfatte determinate condizioni.

Gli oggetti della classe Gem possono essere di colori differenti, definiti in una enumerazione posta nella classe stessa. Grazie all'uso dell'enumerazione, è possibile aggiungere nuovi colori in futuro, lasciando inalterato la maggior parte del codice, ma necessitando di una piccola modifica all'algoritmo di scelta randomica del colore, che ora dovrà tenere conto della nuova aggiunta. Tale scelta randomica è anche relazionata alla difficoltà impostata dall'utente nelle fasi preliminari al gioco, che porta alla comparsa di un numero maggiore di gemme differenti tanto più alta è la difficoltà.

Infine la classe contenente la matrice di gioco si compone di un oggetto della classe **Player**, che permette di settare e di restituire la difficoltà corrente e contiene le informazioni sulle gemme che il giocatore sceglie e che intende cambiare di posizione.

Di seguito lo schema UML del Model.



Schema UML relativo al Model

2.2.2 View

L'intera creazione dell'interfaccia grafica è basata sui componenti messi a disposizione dalla libreria Swing.

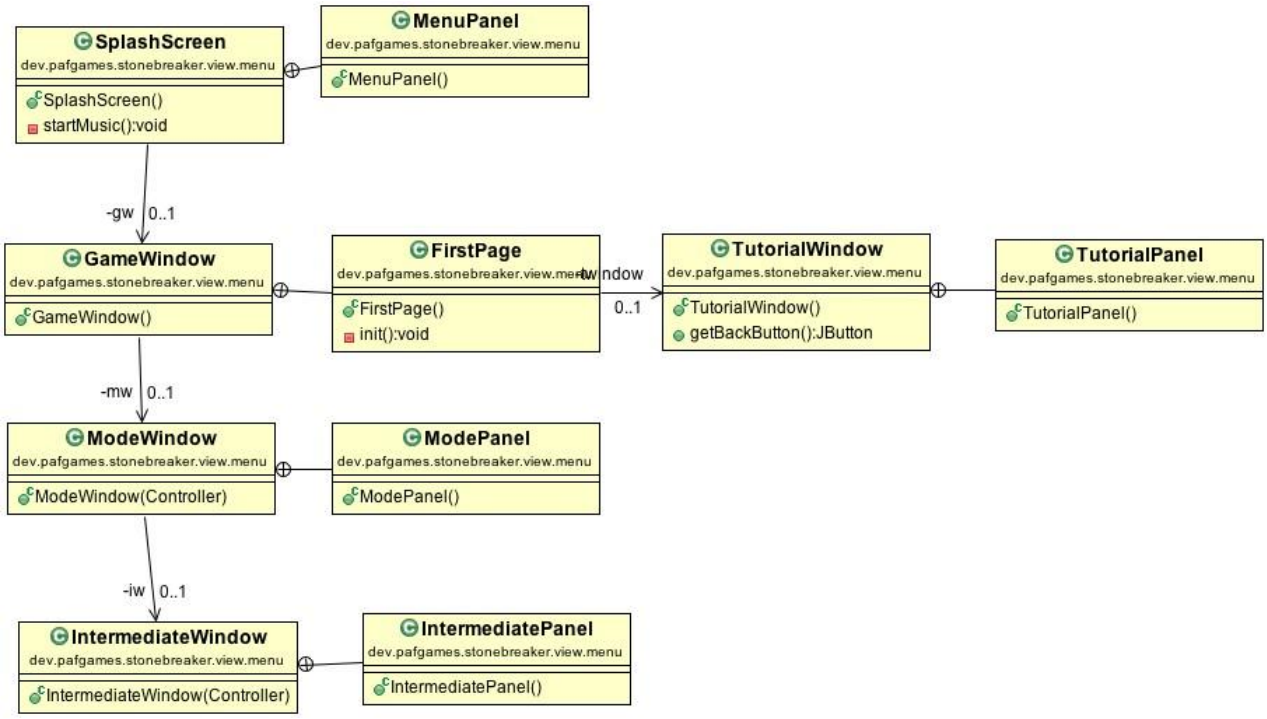
Al lancio dell'applicazione verrà visualizzata una finestra animata di caricamento (**SplashScreen**) che porterà alla schermata principale (**GameWindow**).
All'interno della schermata principale la View permette la navigazione all'interno di un menù a bottoni tra cui è possibile scegliere di iniziare il gioco, uscire dal gioco o visualizzare una finestra con un breve tutorial su come affrontare il gioco.
Quando si deciderà di iniziare il gioco una serie di finestre permetteranno all'utente di scegliere modalità (Arcade o Classic) e difficoltà di gioco (Facile - Normale - Difficile). Tali decisioni verranno effettuate premendo i bottoni opportuni che hanno il compito di notificare il controller tramite pattern **Observer**, sarà quindi compito del controller, in base alle informazioni di notifica ricevute dalla GUI di creare l'istanza della vera e propria schermata di gioco (**BoardGUI**).

È stato scelto di suddividere la schermata di gioco in 5 pannelli principali in base alla loro funzione:
Title Panel : Pannello nell'area superiore della finestra. È a mero e unico uso decorativo.
Score Panel : È il pannello che presenta il valore aggiornato del punteggio durante la partita. Viene anche utilizzato per accogliere il bottone relativo al PowerUp. Al momento il potenziamento è uno solo ma tale implementazione fornisce la possibilità di accogliere altri componenti per estensioni future.
Time Panel : È il pannello utilizzato per mostrare all'utente i propri indicatori di gioco: tempo o mosse rimanenti. Come Score Panel fornisce ulteriore spazio per potenziali estensioni future senza dover effettuare importanti cambiamenti.
Bottom Panel : È il pannello che accoglie il pulsante di pausa per poter fermare il gioco o, eventualmente, uscire.
BoardGrid : È il pannello dove è presente la board con gli elementi di gioco. Pietre o blocchi implementati tramite bottoni. L'interazione con essi avviene con il mouse, il quale, sempre con pattern **Observer** notifica al controller quali bottoni sono stati premuti. In particolare si è pensato di creare un'implementazione custom di JButton per ogni bottone (**StoneButton**) , in modo da poter salvare al suo interno, alla creazione, l'indice di posizionamento sulla board.

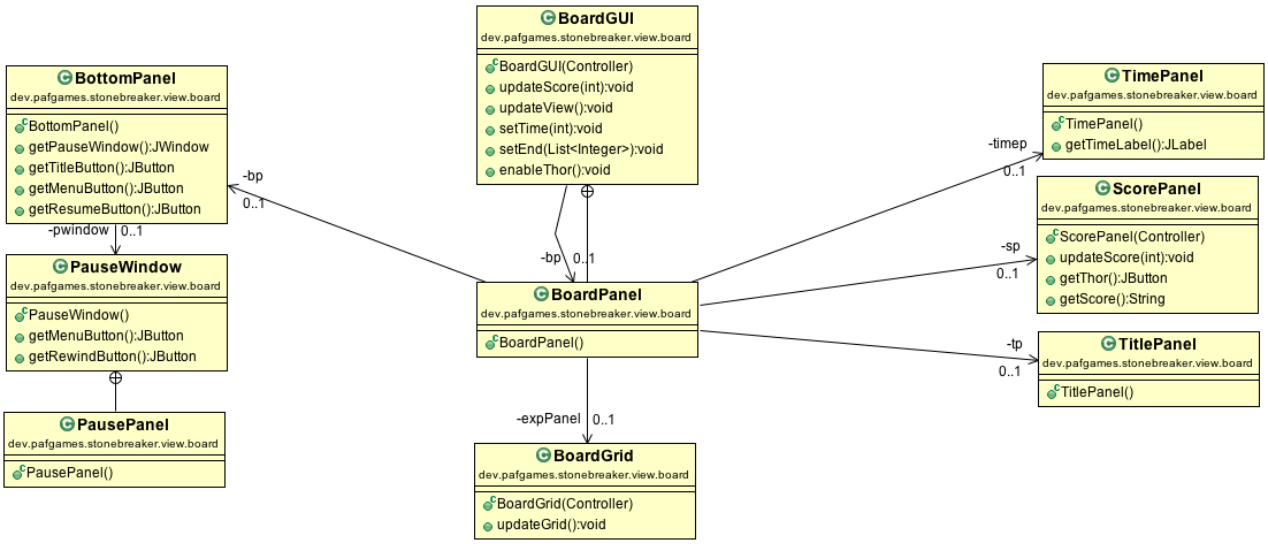
L'ulteriore controllo sul gioco passa di mano al controller che aggiornerà, modificherà, chiuderà la schermata di gioco tramite metodi e funzioni messi a disposizione in **BoardGUI**:
• updateScore : aggiorna lo score di gioco visualizzato.
• updateView : aggiorna la matrice di gioco.
• setTime : setta il tempo di gioco visualizzato.
• setEnd : dispone la chiusura del gioco e la visualizzazione della schermata di fine. (**GameOverFrame**)
• enableThor : abilita il bottone per il powerUp.

Tale implementazione della View in ottica architetturale permette semplicemente la sostituzione dell'interfaccia grafica con un'altra che presenti le sole basilari funzioni sopra citate. È eventualmente possibile sostituire anche solo parti della View (p.e. Menù) senza influenzare minimamente la schermata di gioco.

È stato scelto di dividere il progetto in tre package distinti:
dev.pafgames.stonebreaker.view.board : Package contenente la schermata di gioco e tutti i componenti utili alla schermata di gioco.
dev.pafgames.stonebreaker.view.menu : Package contenenti i vari menù di selezione e schermate di interazione.
dev.pafgames.stonebreaker.view.utilities: Package contenente classi ed estensioni di classi utili alle varie implementazioni e ad evitare possibili ripetizioni di codice.



Schema UML relativo al menù di selezione.



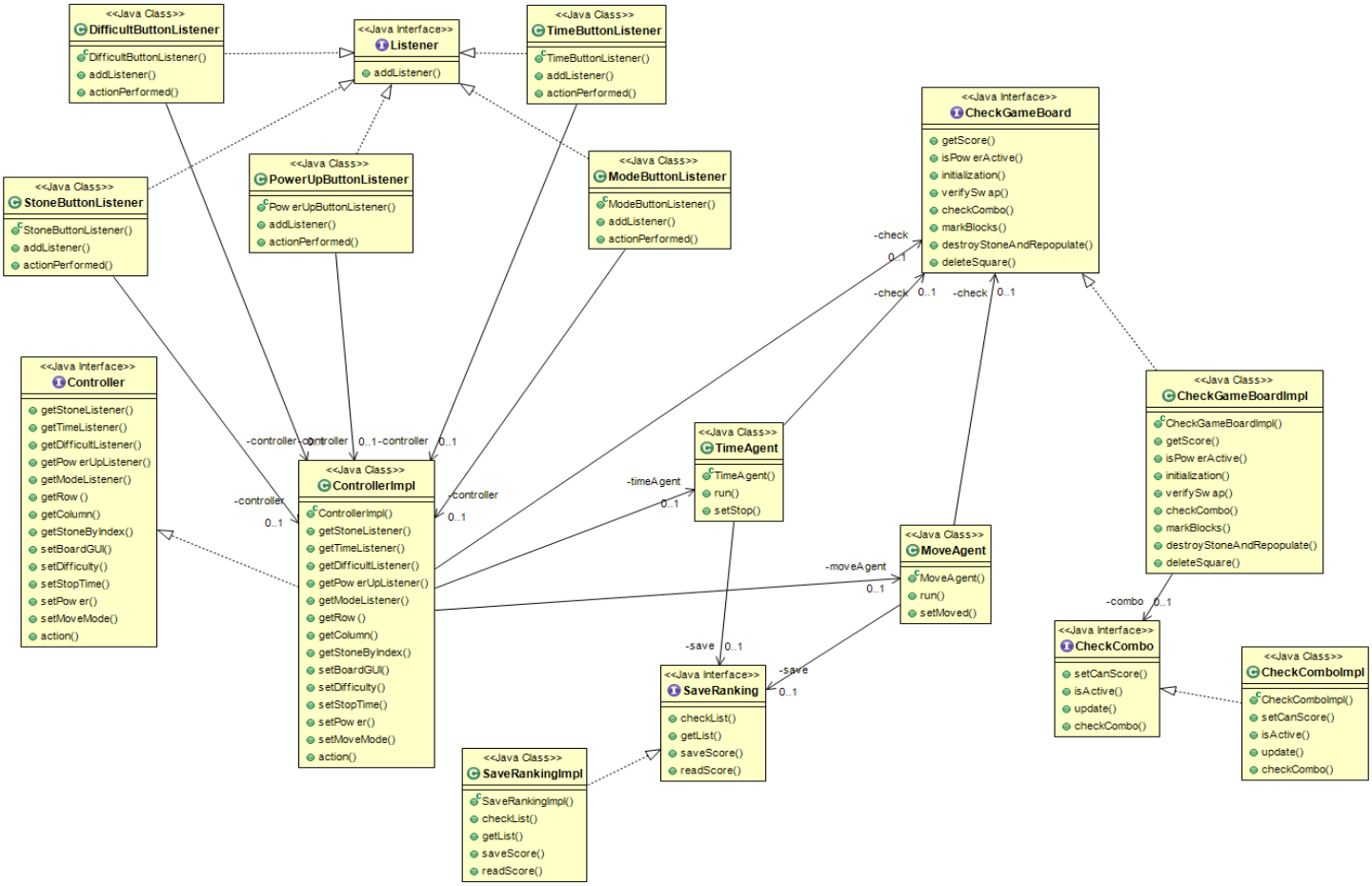
Schema UML relativo alla schermata di gioco.

2.2.3 Controller

La parte di Controller si occupa principalmente di mettere insieme gli elementi della View insieme con gli elementi del Model, in particolare si occupa di osservare la View e tramite gli eventi che si scatenano poi indicare al Model le azione da eseguire. Una delle principali difficoltà che devono essere affrontate nella parte di Controller è riconoscere, a seguito dello spostamento nel campo di gioco di due pietre, le varie combo venutesi a formare: a tale riguardo la classe CheckComboImpl si occupa proprio di affrontare questa problematica partendo dalla pietra selezionata e controllando in modo ricorsivo se sono presenti combo su tutto il campo da gioco. Parte fondamentale è inoltre l'osservazione della View per individuare lo scatenarsi di determinati eventi, in particolare della pressione di bottoni: a tal proposito è stata creata una interfaccia Listener che estende ActionListener in modo tale che per ogni ActionListener che si desidera creare è sufficiente aderire al contratto di quell'interfaccia, in particolare qui il codice è facilmente estendibile in quanto se per caso saranno inseriti nuovi pulsanti nella View, la creazione di ActionListener sarà semplice e immediato. La parte di Controller poi si occupa di dare l'avvio alle varie finestre del gioco e di prelevare dal Model sotto forma di lista ogni elemento della matrice di gioco che deve essere mostrato. Le due modalità di gioco (Classico a tempo o Arcade a mosse), sono state create a partire da classi che utilizzano Thread in modo tale da mostrare a video, simultaneamente con le altre operazioni, cosa manca per terminare la partita, anche questo punto si presta facilmente all'aggiunta di nuove modalità di gioco. Infine la parte di audio è gestita attraverso un Thread che preleva dalle risorse il file audio da riprodurre e attraverso la libreria JavaZoom lo manda in esecuzione: attraverso un parametro booleano è possibile decidere se mandare in loop o meno l'audio che si desidera creare.

Un pattern utilizzato è l'Observer in cui il Controller osserva la View attraverso i Listener e di seguito ordina ciò che deve essere eseguito dal Model per poi mostrarlo di nuovo all'utente.

La figura di seguito mostra la struttura del Controller.



Schema UML relativo al Controller

CHAPTER 3

SVILUPPO

3.1 Testing automatizzato

E' stata creata una classe di testing usando il framework JUnit per verificare la correttezza dei metodi delle classi del Model. In particolare i test si focalizzano sui metodi della classe GameBoardImpl, che ha il compito di modellare la matrice di gioco, e verificano che le interazioni di questa con le altre classi siano corrette.

Per una migliore leggibiltà del test stesso, è stato creato un metodo printBoard per la stampa della matrice di gioco come matrice di lettere (nella classe compare una legenda per l'interpretazione di tali lettere), e a seguito di ogni test la matrice viene stampata a video per fornire all'utente una rappresentazione facilmente comprensibile della situazione della matrice di gioco nel momento successivo alle modifiche.

La classe è composta da due test separati:

- largeScaleTest che verifica i cambiamenti di molteplici elementi della matrice di gioco, e che fa uso del metodo printBoard per fornire la rappresentazione visiva di tali cambiamenti;
- smallScaleTest che verifica le interazioni tra singoli elementi della matrice.

3.2 Metodologia di lavoro

Seguendo la progettazione architetturale MVC, il progetto è stato diviso tra i membri del gruppo assegnando a ciascuno una parte, come dichiarato nella proposta del progetto. Le tre parti sono state scritte e modificate da chi vi era responsabile, ma per garantire un risultato ottimale e concorde al volere del gruppo ognuno era libero di proporre cambiamenti e possibili migliorie nel codice degli altri, e di chiedere un parere su parti di codice logicamente o sintatticamente complesse.

Come punto di partenza sono state create delle interfacce comuni che hanno permesso a ciascuno di lavorare autonomamente e di usare i metodi delle classi degli altri membri. Ciò ha generato anche delle problematiche nei casi in cui bisognava modificare o aggiungere codice, che tuttavia è stato quasi sempre facilmente risolvibile in quanto tale fase di creazione delle interfacce è stata lunga e attenta.

Per garantire una realistica divisione dei compiti è stata fatta molta attenzione alla suddivisione MVC, assicurandoci che tutte le interazioni tra la parte di model e la parte di view passassero per il controller.

L'attenta suddivisione del progetto è stata di grande aiuto nella fase di debugging, che ha permesso una rapida individuazione delle problematiche, che sorgevano con l'aggiunta di nuove funzionalità, nel punto preciso che scatenava l'errore.

3.3 Note di sviluppo

L'intero progetto è stato pensato, scritto e modificato dai membri del gruppo. Tuttavia si segnala per correttezza che è stato preso dal web la parte di codice relativa alla gestione dell'audio, che fa uso della libreria JLayer e dei suoi API per la gestione dei file in formato MP3, in quanto la soluzione pensata inizialmente dal gruppo dava alcuni problemi. Tuttavia il codice non è un meschino copia-incolla, ma è stato a sua volta modificato per meglio adattarsi allo stile proposto a lezione.

Federico Pucci: Fonte utile e fondamentale nello sviluppo della mia parte di progetto è il sito StackOverflow.com. Non ho effettuato nessuna sorta di copia ed incolla ma ho effettivamente sfruttato consigli utili: Caricamento icone con enum per le gemme (per possibili estensioni future). Resize automatico del background dei pannelli. Strategie per il caricamento di gif animate usate nello splashScreen.

Tra materiali e immagini utili nella view cito:

- Preloaders.net e FlamingText.com per le gif animate
- opengameart.org per le immagini relative a gemme e blocchi
- Flaticon.com per icone
- Il resto di bottoni e immagini è stato fatto da me

CHAPTER 4

SVILUPPO

4.1 Autovalutazione e lavori futuri

Il team giudica soddisfacente il lavoro effettuato.

Si è speso molto tempo nella fase preliminare per l'utilizzo degli strumenti e per accordarsi sul lavoro da svolgere, non essendo inizialmente abituati a lavorare in gruppo. Successivamente ognuno ha apportato il proprio contributo e ha collaborato alla riuscita finale del progetto con impegno e serietà.

Federico Pucci: posso dire di essere soddisfatto del mio lavoro. Ho cercato di organizzare il mio progetto in maniera da essere il più chiaro e comprensibile possibile, volto anche a possibili estensioni future. Personalmente il lavoro fatto mi è stato molto utile ad assimilare in modo concreto concetti studiati. La parte su cui ho investito più tempo è stato lo studio per l'interazione con il controller. Probabilmente avrei potuto sfruttare librerie alternative per migliorare il codice, tra queste cito JavaFx, che però ho scelto di non adottare perché temevo che il tempo a disposizione non fosse sufficiente per ottenerne la padronanza necessaria. Sono soddisfatto anche dei membri del mio gruppo che reputo degli ottimi collaboratori anche in ottica di progetti futuri. La cooperazione è stata a mio avviso molto stimolante e al tempo stesso divertente.

Matteo Falco: sono soddisfatto del progetto, dai numerosi test sono emersi 0 bug e il codice non ha warnings. La mia parte di codice potrà essere anche numericamente inferiore a quella degli altri miei colleghi, ma ha richiesto principalmente un lavoro di ragionamento più che di scrittura, e una volta concluso la mia parte ho contribuito alla ricerca di eventuali errori e alla loro risoluzione anche nel codice non mio (e come mostrato a lezione nei lucidi, ho potuto capire sulla mia pelle come la parte di debugging copra effettivamente una significativa quantità di tempo). L'intero team era affiatato e volenteroso di consegnare un progetto di qualità, farei volentieri un altro progetto con questa squadra.

Luca Agatensi: personalmente sono molto soddisfatto del nostro progetto, credo che il lavoro di gruppo si sia sviluppato nel migliore dei modi, anche se con qualche piccolo intoppo iniziale dovuto a Mercurial ed Eclipse, e che quindi siamo riusciti a coordinarci e lavorare in modo più possibile parallelo ognuno sviluppando la sua parte. Nel complesso credo sia veramente un lavoro ben riuscito e che si pone a ulteriori modifiche e aggiunte future nel caso vorremmo ancora lavorarci sopra.

APPENDICE

GUIDA UTENTE

L'applicazione contiene al suo interno una sezione "tutorial" che spiega come approcciarsi al gioco.
L'utente prima di giocare può scegliere tra due diverse modalità di gioco:

- classic, in cui deve fare il maggior numero di punti possibili nel tempo a disposizione;
- arcade, in cui ha un numero limitato di mosse per fare più punti possibili.

Una volta selezionata la modalità di gioco dovrà scegliere tra tre difficoltà:

- Easy, 4 gemme differenti;
- Medium, 5 gemme differenti;
- Hard, 6 gemme differenti