A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

Tesina Finale di

Programmazione di Interfacce Grafiche e Dispositivi Mobili

Corso di Laurea in Ingegneria Informatica ed Elettronica – A.A. 2020-2021

Dipartimento di Ingegneria

docente

Prof. Luca Grilli

JMahbusa

applicazione desktop JavaFX

studenti

316683 **Giacomo** **Volini** giacomo.volini@studenti.unipg.it

321116 **Ameen** **Dabool** ameen.dabool@studenti.unipg.it

Data ultimo aggiornamento: 24 ottobre 2022

0. Indice

1. Descrizione del Problema

L'obiettivo del progetto è lo sviluppo di un'applicazione desktop, chiamata *JMahbusa*, che implementi una versione digitale del gioco da tavolo *Mahbusa*, una variante del *Backgammon* giocata nel Medio Oriente e in Asia Centrale [1, 2, 3]

Nel seguente paragrafo sarà data una descrizione del regolamento del gioco del Mahbusa.

1.1 Regolamento del gioco del Mahbusa

Ogni giocatore ha quindici pedine, che partono dal posto 1 e 24 rispettivamente. I due giocatori muovono le pedine in direzione opposta l'uno all'altro lungo il tabellone. I triangoli nel tabellone sono gli spazi che le pedine occupano e attraversano, e si chiamano **punte**.

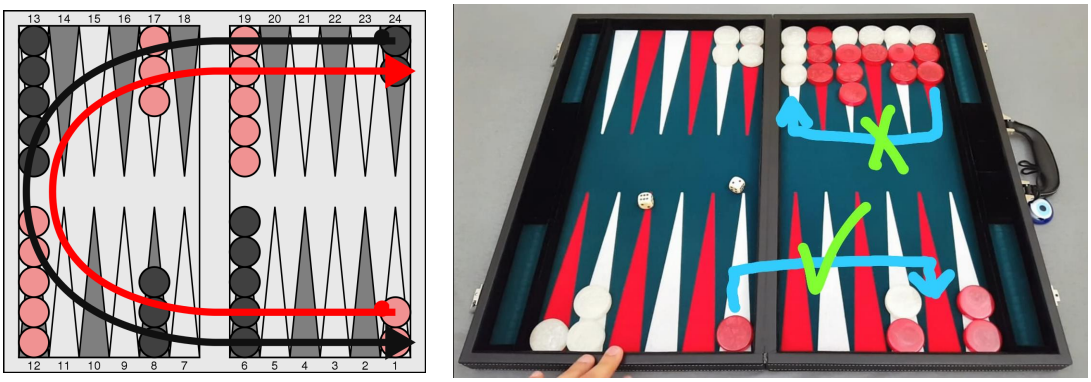
Il numero di punte che le pedine possono attraversare è deciso dal tiro di due dadi. Ogni dado corrisponde ad una mossa. Nel caso di un risultato doppio i dadi contano due volte. Ad esempio, un tiro di 3-3 conta come quattro dadi da 3. Il giocatore può anche decidere di muovere una singola pedina con più di un dado, contando però separatamente le mosse.

Il giocatore può muovere le pedine in punte libere, occupate dalle proprie pedine o al più da solo una pedina avversaria. In questo ultimo caso la pedina avversaria diventa **bloccata** e non può più essere mossa finché la punta non viene liberata. Nel caso di punte occupate da due o più pedine dell'avversario o in cui l'avversario ha bloccato una pedina, il giocatore non può piazzarvi le proprie pedine, ma è libero di "scavalcarle" nel caso il tiro dei dadi glie lo consenta.

Lo scopo del gioco è rimuovere le proprie quindici pedine dal tabellone impedendo all'avversario di rimuovere le sue.

Rimuovere le pedine

Una volta che un giocatore ha portato tutte le sue 15 pedine sulla propria **casa** (le ultime sei punte del percorso) , può iniziare a portarle fuori dalla tavola. Un giocatore porta fuori una pedina tirando un numero corrispondente alla punta in cui si trova la pedina e rimuovendo tale pedina dalla tavola. Quindi, tirando un 6, il giocatore può portare fuori una pedina che si trova sulla punta sei. Se non ci sono pedine su una delle punte indicate dai dadi, il giocatore deve muovere in maniera legale una pedina che si trova su una punta corrispondente ad un numero più alto. Se non ci sono pedine neppure su una punta più alta, allora il giocatore



(a) (b)

Figura 1.1:
(a)Il tabellone del Mahbusa. Con le due frecce sono indicati i percorsi che i due giocatori devono seguire con le proprie pedine.
(b) Un esempio di mosse possibili e non: la mossa contrassegnata con la X non è possibile perchè la pedina andrebbe ad occupare una punta già bloccata dalle tre pedine bianche. La mossa contrassegnata con la spunta è invece possibile perchè pur scavalcando una punta bloccata dal bianco andrebbe ad occupare una punta libera.

deve rimuovere una pedina dalla punta più alta che è ancora occupata da una qualche sua rimanente pedina. Il giocatore non è però obbligato a portare fuori una pedina nel caso ci siano altre mosse consentite.

Vincere la partita

Un giocatore vince se:

1. Rimuove tutte le sue pedine. Nel caso l'avversario non sia riuscito a rimuovere nemmeno una delle sue pedine si parla di *vittoria doppia*.

2. Riesce a bloccare una pedina nella punta di partenza dell'avversario. In questo caso il giocatore ottiene una *vittoria doppia*.

1.2 L'applicazione JMahbusa

L'applicazione JMahbusa consiste in un'implementazione digitale del gioco del Mahbusa per due giocatori, con possibilità di giocare una partita singola o una serie di partite con tracciamento del punteggio.

2. Specifica dei Requisiti

L'applicazione JMahbusa che si intende realizzare dovrà soddisfare i seguenti requisiti.

1. Riprodurre graficamente il tabellone del Mahbusa e la disposizione delle pedine tramite libreria JavaFX
2. Consentire all'utente di interagire con il tabellone con il mouse o con comandi da tastiera.
3. Evidenziare visivamente le pedine che possono essere mosse all'interno di un turno e le punte verso cui possono essere effettuate delle mosse.
4. Permettere ad un giocatore di effettuare le proprie mosse solamente nel caso rispettino il regolamento.
5. Presentare un menu principale da cui poter avviare una partita, uscire dal programma, eccetera.
6. Permettere all'utente tramite apposito menu di impostazioni o modifica di un file di configurazione di:
    - cambiare l'assegnazione dei tasti per i comandi da tastiera;
    - personalizzare i colori di tabellone, punte e pedine;
    - attivare o disattivare la modalità a schermo intero, impostare la risoluzione in finestra e scegliere se consentire o meno il ridimensionamento libero della finestra;
    - regolare separatamente il volume di musica ed effetti sonori;
7. Presentare un tutorial interattivo all'utente come scelta durante la prima partita e come opzione separata all'interno del menu principale.
8. Permettere all'utente di scegliere se giocare una singola partita o una serie di partite fino al raggiungimento di un dato punteggio, e se impostare un tempo massimo per l'esecuzione di un turno.
9. Permettere ai giocatori di inserire i propri nomi o nickname, tenere traccia delle vittorie e sconfitte e visualizzarle in una classifica raggiungibile dal menu principale.

10. Mostrare tramite animazione il tiro dei dadi, evidenziando adeguatamente l'evento di un risultato doppio e le mosse già eseguite durante il turno.

11. Permettere all'utente di annullare le mosse effettuate all'interno del suo turno tramite apposito pulsante.

12. Possibilità di salvare in qualsiasi momento la partita in corso e caricarla in un secondo momento.

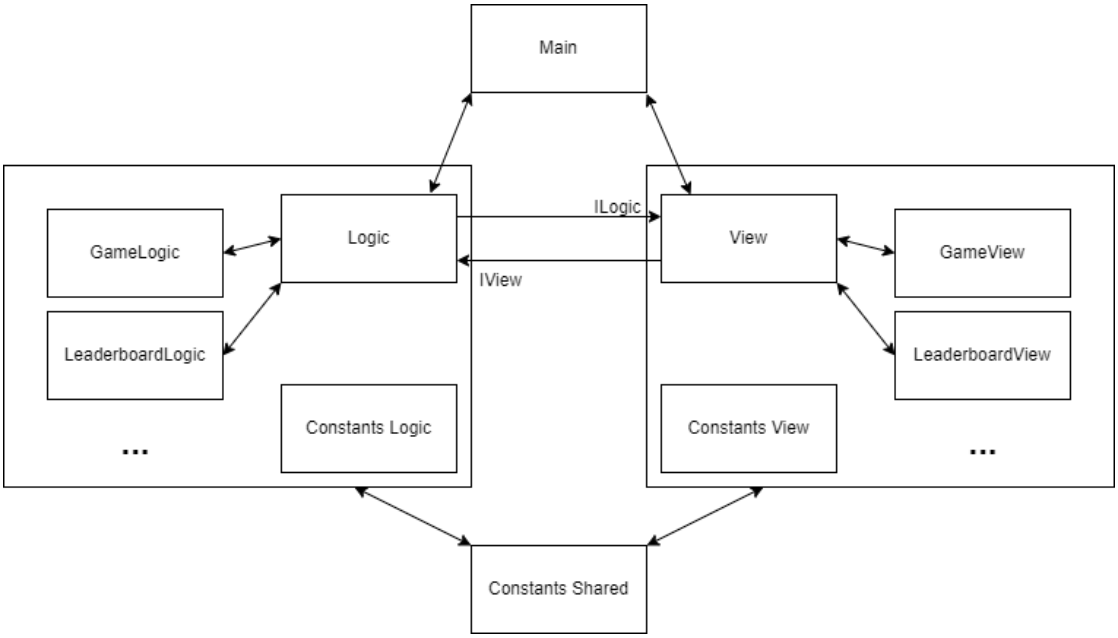
13. Presenza di musiche ed effetti sonori regolabili dalle impostazioni.

14. Adattarsi a seconda delle dimensioni della finestra o della risoluzione dello schermo in caso di visualizzazione a schermo intero, partendo da un minimo di 640x480.

15. Consentire dal menu principale la scelta della lingua, tra italiano, inglese e arabo, con conseguente modifica all'interfaccia utente nell'ultimo caso per adattarsi alla lettura da destra verso sinistra.

3. Progetto

3.1 Architettura del Sistema Software



Per la realizzazione di JMahbusa si è deciso di utilizzare un'architettura basata sul pattern LV (Logic-View).
Il package Logic si occupa della gestione di tutti i dati gestiti dall'applicazione, sia quelli relativi alla partita in corso (come lo stato del tabellone e dei dadi) che gestione e interpretazione di salvataggi, classifiche, impostazioni e stringhe.
Il package View si occupa di mostrare tramite un'interfaccia grafica i dati del Logic e di permettere all'utente un'interazione con essi.
La comunicazione all'interno dei package e tra package è affidata a due classi, Logic e View, e a due Interface, ILogic e IView, che consentono ad un package di accedere ai metodi dell'altro senza conoscerne la loro implementazione.
Nella figura sottostante si può vedere uno schema estremamente semplificato di questa struttura. Al di fuori dei due package vi sono due classi, Main e ConstantsShared. La classe Main è la classe di avvio dell'applicazione, che contiene un metodo di avvio per JavaFX e un metodo per inizializzare le istanze di ILogic e IView per il corretto funzionamento dell'applicazione. La classe ConstantsShared contiene le costanti utili per classi di entrambi i package.

3.2 Logic

Il modulo Logic include tutte le classi inerenti la gestione dei dati e della logica del gioco.

3.2.1 Logic

Logic è la classe che permette la comunicazione tra i vari oggetti all'interno del package e, attraverso l'interfaccia *ILogic*, con il package *view*. Essa contiene due variabili di classe, una *ILogic* per le comunicazioni intrapackage, una *IView* per avere accesso ai metodi del package *view*.

Variabili d'istanza

Logic contiene come variabili d'istanza un riferimento alla cartella in cui è contenuto l'eseguibile, e agli oggetti necessari per la corretta esecuzione del programma. La variabile *DynamicBoardLogic* può contenere un oggetto di una delle due sottoclassi *TutorialLogic* o *GameLogic*, sfruttando il polimorfismo dove necessario.

Metodi d'istanza

Logic contiene una serie di metodi di inizializzazione, che creano gli oggetti delle varie classi e generano la corretta struttura di cartelle e file, e dei metodi di comunicazione che vanno a richiamare specifici metodi dei vari oggetti del package.

3.2.2 DiceLogic

DiceLogic è la classe che rappresenta i dadi in una partita di Mahbusa. Essi sono modellati in tre array di dimensione 4, uno di interi che rappresenta i valori dei dadi (*dice*), e due di booleani che rappresentano quali dadi sono già stati usati per una mossa (*used*) e quali stanno per essere utilizzati dalla mossa che si sta correntemente valutando (*toBeUsed*). E' presente inoltre un booleano (*doubleNum*) per indicare se il tiro corrente è doppio o meno e un oggetto di tipo Random (*rnd*) per avere dei tiri casuali.

Metodi

- Getter e Setter per gli attributi della classe
- *tossDice*, che effettua un tiro dei dadi e assegna di conseguenza i valori dei tre array secondo regolamento
- *doesWhiteStart*, che determina quale giocatore inizierà la partita
- *countAvailableDice*, che restituisce il numero di dadi non utilizzati

3.2.3 DynamicBoardLogic

DynamicBoardLogic è la classe astratta che modella il tabellone del Mahbusa e assicura che le mosse avvengano secondo regolamento. Il tabellone viene rappresentato con una matrice di interi 16x26, *squares*, i cui valori rappresentano se lo spazio corrispondente è vuoto o occupato da una pedina, che può essere dell'uno o dell'altro giocatore ed essere selezionata o meno per una mossa. A numeri diversi corrispondono combinazioni diverse di stati. Degli opportuni booleani rappresentano il giocatore di turno e se i due giocatori possono portare fuori le proprie pedine o meno.

Metodi principali

- *setUp*, che imposta il tabellone allo stato iniziale
- *setUpSavedBoard*, che imposta il tabellone in base ad una matrice ricevuta in ingresso, per esempio durante il caricamento di un salvataggio.
- *movePawn*, che si occupa di spostare una pedina all'interno del tabellone se il regolamento lo consente. Una versione di movePawn richiede solo due interi, rappresentanti le colonne di partenza e arrivo, l'altra richiede quattro interi rappresentanti le posizioni precise di partenza e arrivo nel tabellone.

- *possibleMove*, che riceve le informazioni relative ad una mossa e determina se essa è effettuabile o meno secondo il regolamento
- *checkWhiteExit* e *checkBlackExit*, che controllano il tabellone e determinano se i giocatori possono portare fuori le proprie pedine o meno

3.2.4 GameLogic

GameLogic estende *DynamicBoardLogic*, e contiene le informazioni necessarie per una partita di JMahbusa, come i nomi dei giocatori, se la partita giocata è parte di un torneo o no, quanti punti deve ottenere un giocatore per vincere il torneo, il limite di tempo massimo per un turno e la lista delle mosse effettuate in un singolo turno (ogni mossa è rappresentata da un oggetto *moveRecord*.

Metodi principali

- *changeTurn*, per passare al turno successivo
- *completeMoves*, che effettua arbitrariamente le mosse per il giocatore nel caso il suo turno finisca e vi siano ancora dei dadi non utilizzati
- *victoryCheck*, che controlla se si stanno verificando le condizioni per la fine della partita, ed in caso affermativo, chiama il metodo di vittoria con i giusti attributi in ingresso
- *gameWon*, che riceve informazioni sul vincitore e sul tipo di vittoria (singola o doppia), assegna i punti al vincitore e chiama il metodo corrispondente del view
- *revertMove*, che se possibile annulla l'ultima mossa effettuata
- *setUpSavedGame*, che riceve i dati di un salvataggio e li utilizza per ripristinare quella partita.

MoveRecord

MoveRecord è una classe che incapsula i dati relativi ad una mossa del Mahbusa, con attributi che descrivono le punte di partenza e arrivo della mossa, quali dadi sono stati utilizzati per essa, e se la mossa ha permesso l'apertura della zona di uscita di uno dei giocatori.

3.2.5 TutorialLogic

TutorialLogic estende *DynamicBoardLogic*, e contiene una lista di fasi del tutorial e un riferimento alla fase corrente.

Subpackage *tutorial* e metodo *stageArrayListConstructor*

Il tutorial di JMahbusa è costituito da diverse fasi. Per consentire un livello di flessibilità aggiuntiva nella creazione del tutorial ogni fase è un'estensione diversa di una classe astratta, *ComparableTutorialStage*, che a sua volta è un'implementazione di due Interface: *TutorialStage* e *Comparable*. Interface, classe astratta e implementazioni sono contenute nel subpackage *tutorial*, interno a *logic* Ogni fase del tutorial deve contenere due metodi:

- *start*, che contiene le istruzioni da eseguire appena entrati in quella fase
- *action*, che contiene i vari controlli e messaggi aggiuntivi per la prosecuzione del tutorial

Ogni fase contiene inoltre un attributo chiamato *stageIndex*, utilizzato per confrontare le diverse fasi ed ordinarle. Il metodo *stageArrayListConstrusctor* viene richiamato all'interno del costruttore di *TutorialLogic*. Tramite reflections sul package *tutorial*, il metodo raccoglie tutte le estensioni di *ComparableTutorialStage* in un Set, le istanzia all'interno di un ArrayList e ordina la lista.

3.2.6 LeaderboardLogic

LeaderBoardLogic gestisce i dati dei giocatori che hanno completato almeno una partita, tramite un'*ArrayList* di *Player*, un oggetto *Path* contenente il riferimento alla cartella che contiene il file della classifica, ed un oggetto *BufferedReader* utile alla lettura di quest'ultimo.

Metodi principali

- *populateList*, che legge il file della classifica e popola l'ArrayList di conseguenza
- *addNewPlayer*, che aggiunge un nuovo giocatore alla lista
- *compareNameLists*, che controlla se esistono già dei giocatori con i due nomi inseriti
- *ldbWriter*, che prende i dati dell'ArrayList e li salva in un file apposito

Player

Player è la classe che incapsula i dati relativi ad un giocatore, tramite oggetti del tipo *"SimpleStringProperty"*, *"SimpleIntegerProperty"* e *"SimpleDoubleProperty"*. Nello specifico, di un giocatore viene salvato il nome, il numero di vittorie e sconfitte ed il rapporto tra vittorie e sconfitte.

3.2.7 SaveGameWriter

SaveGameWriter si occupa della scrittura dei file di salvataggio, andando a tradurre le varie informazioni di gioco in un formato testuale, e della loro cancellazione

Metodi statici

- *deleteSaveFile*, che cancella un file dal nome corrispondente alla stringa fornita in ingresso
- *generateSquareMatrixString*, che prende in ingresso un oggetto *GameLogic*, interpreta lo stato del tabellone e lo traduce in una stringa singola, fatta di numeri e caratteri "$\backslash n$"
- *writeSaveFile*, che ricevendo in ingresso un oggetto *GameLogic* ed una stringa crea un oggetto JSONObject, lo popola con le informazioni della partita da salvare, se necessario crea la cartella dei salvataggi, crea un file .json dal nome specificato e lo popola con i dati del JSONObject

Contenuto di un file di salvataggio

- Stato del tabellone
- Giocatore di turno
- Nomi dei giocatori
- Punti necessari alla vittoria (e quindi eventuale modalità torneo)
- Punti di ciascun giocatore
- Durata di un turno in secondi
- Possibilità o meno di annullare le mosse
- Apertura o chiusura delle zone di uscita

3.2.8 SaveGameReader

SaveGameReader si occupa della lettura dei file di salvataggio. Ogni oggetto *SaveGameReader* incapsula i dati di un singolo salvataggio.

Metodi statici

- *getSaveList*, che restituisce una *List* di stringhe contenente i nomi di tutti i file di salvataggio presenti nell'apposita cartella
- *isSaveNamePresent*, che controlla se il nome di un salvataggio fornito in ingresso è già presente nella cartella
- *readSaveGame*, che crea e restituisce un oggetto *saveGameReader* invocandone il costruttore privato

Metodi d'istanza

- *parseSquareMatrixString*, che ritraduce la stringa squareMatrix in una matrice di interi
- *getSquareMatrix*, che restituisce la matrice di interi ottenuta col metodo precedente
- *getLoadViewData*, che restituisce un array di stringhe contenenti le informazioni necessarie per la schermata di caricamento

3.2.9 SettingsLogic

SettingsLogic gestisce tutte le impostazioni relative a JMahbusa, creando, leggendo e modificando dove necessario un apposito file *current.ini* per il loro salvataggio, grazie alla libreria Ini4J, ed in particolare ad un oggetto di tipo *Ini*.

Costanti

La classe *SettingsLogic* contiene quattro costanti, due interi rappresentanti la risoluzione minima consentita in finestra e due array di Stringhe rappresentanti i colori dei due preset disponibili per tabellone e pedine

Metodi statici

- *flagTutorialPlayed*, che crea un file dummy chiamato *played.tut* per indicare che l'utente ha già completato il tutorial

- *shouldPlayTutorial*, che restituisce un booleano a seconda della presenza o meno del file *played.tut*

Metodi d'istanza

- Il costruttore, che se necessario crea la cartella *settings* ed il file *current.ini*, leggendo le impostazioni dal file *defaults.ini* presente all'interno del Jar; in caso contrario legge semplicemente i dati dal file *current.ini* precedentemente creato
- *getDefaultSettings*, che legge le impostazioni dal file *defaults.ini* e le salva nel file *current.ini*
- *restoreCurrent*, che rilegge il file *current.ini* sovrascrivendo eventuali cambiamenti non salvati
- *setSetting*, che inserisce un dato valore nell'oggetto *Ini* a seconda di sezione e chiave forniti in ingresso.
- *getSetting*, presente in due versioni:
  - una che restituisce il valore di un'impostazione ricevendo sezione, chiave e tipo di dato da ricevere
  - una che restituisce uno dei valori di preset

3.2.10 StringsReader

StringsReader si occupa di fornire al package *view* tutte le stringhe della lingua selezionata dall'utente. Ogni lingua viene gestita tramite un apposito file .ini. La classe gestisce inoltre il file *supportedLanguages.ini*, che contiene una lista delle lingue supportate dall'applicazione e una sottolista di quelle che si leggono da destra a sinistra.

Struttura dei file di lingua

I file di lingua seguono la convenzione "STRINGS_XX.ini", dove al posto di XX si trova una sigla rappresentante la lingua (ad esempio "IT" o "AR"). Ogni file contiene una singola sezione ed una serie di associazioni chiave/valore.

Metodi statici

- *getSupportedLanguages*, che legge il file *supportedLanguages.ini* e restituisce l'array dei valori presenti con la chiave supportedLanguage

- *isLanguageRightToLeft*, che controlla se una certa lingua è presente nella sottolista dei linguaggi letti da destra a sinistra

Metodi d'istanza

- Il costruttore, che costruisce il nome del file di lingua da leggere e crea un oggetto Ini che lo gestisca
- *get*, che data una chiave restituisce la stringa corrispondente

3.3 View

Il modulo View include tutte le classi inerenti la gestione dell'interfaccia grafica e di musica ed effetti sonori.

3.3.1 View

View è la classe che permette la comunicazione tra i vari oggetti all'interno del package e , attraverso l'interfaccia *IView*, con il package *logic*. Essa contiene due variabili di classe, una *IView* per le comunicazioni intrapackage, una *ILogic* per avere accesso ai metodi del package *logic*. La classe ha inoltre due variabili d'istanza: un oggetto che implementi l'interfaccia *GenericGUI*, rappresentante la schermata visibile in un dato momento all'utente, ed un oggetto *MusicPlayer* per la gestione di musica ed effetti sonori. I metodi della classe view richiamano metodi di altre classi del package, per garantire il disaccoppiamento tra *logic* e *view*.

3.3.2 GenericGUI

L'interfaccia *GenericGUI* viene implementata da tutte le classi di *view* rappresentanti una schermata del gioco. Essa contiene due metodi:

- *initialize()*, il metodo nativo di JavaFX richiamato dopo il costruttore e dopo aver caricato tutte le informazioni del file *.fxml* associate alla classe
- *changeDimensions()*, il metodo richiamato per il ridimensionamento di tutte le componenti della schermata visibile all'utente

3.3.3 AnimatedBoard

L'interfaccia *AnimatedBoard* viene utilizzata per generalizzare l'invocazione di metodi analoghi di *TutorialView* e *GameView*.

3.3.4 App

App è la classe di lancio di JavaFX. Essa contiene una variabile di classe *Stage*, la finestra all'interno del quale opera tutto il resto dell'applicazione ed una variabile di classe *Scene*, ovvero la schermata caricata nello Stage in un dato momento dell'esecuzione di JMahbusa. Il metodo d'istanza *start* viene invocato all'avvio di JavaFX e si occupa di inizializzare *Stage, Scene* con i valori adeguati (andando laddove necessario a richiedere valori a *ILogic*, e *MusicPlayer*.

Metodi di classe

- *main*, che lancia JavaFX
- *changeRoot*, che cambia la schermata visualizzata all'interno dello *Stage* modificando la radice della *Scene*
- *loadFXML*, che carica la classe associata ad un file .fxml e assegna il relativo oggetto come schermata corrente in *View*
- *setStageOptions*, che imposta la *Stage* a schermo intero o in finestra e blocca o meno la possibilità di ridimensionare la finestra.

3.3.5 ConstantsView

ConstantsView è la classe che contiene tutte le costanti utili esclusivamente all'interno del package *view*

3.3.6 DiceView

DiceView è la classe che gestisce l'aspetto dei dadi di JMahbusa. Esso contiene due array statici di *Image* per le immagini delle facce dei dadi normali e doppi e due oggetti *ColorAdjust* per distinguere tra dadi usati e non.

Metodi principali

- *rndrolls*, assegna ad ogni dado normale una faccia casuale, richiamando il metodo qui sotto. Viene utilizzato per l'animazione del tiro dei dadi.
- *rndroll*, riceve in ingresso un'*ImageView*, e gli assegna come immagine una faccia casuale del dado.
- *setDiceValues*, assegna ai dadi le facce giuste chiedendo a *logic* il risultato del tiro dei dadi.

- *setDiceContrast*, assegna a ciascun *ImageView* uno dei due *ColorAdjust* chiedendo a *logic* se il dado corrispondente è stato usato o meno.

3.3.7 GameBoard

GameBoard è una classe astratta che contiene variabili e metodi relativi alla resa visiva del tabellone del Mahbusa e delle sue pedine senza animazioni e senza dadi. Per fare ciò esso utilizza una serie di oggetti *Rectangle*, *Polygon*, e *PawnView*, una classe che estende *Circle* aggiungendogli dei metodi di utilità per determinare la posizione all'interno del proprio *Parent* del centro. Il ridimensionamento delle componenti di *GameBoard* è delegato alla classe *GameBoardRedraw*.

3.3.8 DynamicGameBoard

DynamicGameBoard è una classe astratta che estende *GameBoard* aggiungendovi i dadi e la relativa zona del tabellone, le animazioni del tabellone ed i metodi per manipolare le pedine tramite mouse o tastiera. Il ridimensionamento delle componenti di *DynamicGameBoard* è delegato alla classe *DynamicGameBoardRedraw*.

3.3.9 GameView

GameView è la classe relativa al gioco del Mahbusa vero e proprio. Essa estende *DynamicGameBoard*, a cui aggiunge i pulsanti per la gestione della partita, delle componenti per visualizzare le informazioni relative ad essa, un menu di pausa (da cui poter salvare la partita) e un pannello per congratularsi con il vincitore della partita. Il ridimensionamento delle componenti di *GameView* è delegato alla classe *GameViewRedraw*.

3.3.10 LeaderboardView

LeaderboardView è la classe che rappresenta la schermata della classifica. Essa è composta da una *TableView* e quattro *TableColumn* per rappresentare la classifica vera e propria, un *TextField* utilizzato come casella di ricerca e due *Button*, uno per tornare al menu principale ed uno per effettuare la ricerca per nome.

3.3.11 LoadGameView

LoadGameView è la classe che rappresenta la schermata di caricamento. Essa è divisa in due *AnchorPane*: in quello di sinistra è presente un *ListView* contenente la lista dei nomi dei salvataggi ed un *Button* per tornare al menu principale, mentre

in quello a destra, una volta selezionato uno specifico salvataggio, si possono vedere tutte le informazioni relative a quella partita, e caricare o eliminare il salvataggio tramite appositi *Button*. Il ridimensionamento delle componenti è delegato alla classe *LoadGameRedraw*

Metodi principali

- *deleteSave*, per cancellare un salvataggio.
- *loadGame*, per caricare un salvataggio.
- *renderSelection*, che imposta le componenti dell'*AnchorPane* di destra per visualizzare i dati di un salvataggio
- *renderNoSelection*, che imposta le componenti dell'*AnchorPane* di destra nel caso non sia selezionato alcun salvataggio

3.3.12 LogIn

LogIn è la classe che rappresenta la schermata di prepartita, in cui i giocatori inseriscono i propri nomi, scelgono la difficoltà della partita e se giocare un torneo o una partita singola. All'utente sono mostrati tre *TitledPane*, uno sempre aperto, gli altri due a scomparsa. Nel primo *TitledPane* sono presenti due *ComboBox* con cui l'utente può scegliere uno dei giocatori già salvati o inserire dei nuovi nomi, due *Circle* per indicare quale giocatore controllerà quali pedine e due *Button* per proseguire alla partita o tornare al menu principale.

Metodi principali

- *savePlayer*, che nel caso l'utente abbia scelto nomi dalla lista o inserito nuovi nomi chiama *GameView* con le impostazioni scelte, e negli altri casi fa comparire a schermo un messaggio adeguato di errore insieme ad un effetto sonoro
- *titledPaneAnimation*, che gestisce l'animazione di apertura o chiusura dei *TitledPane* richiamando il metodo *lowerTournamentPanel* o *raiseTournamentPanel* a seconda della necessità.

3.3.13 MainMenu

MainMenu è la classe che rappresenta il menu principale. Al suo interno si trovano una serie di *Button* per permettere all'utente di passare alle altre schermate o di uscire dal gioco ed una *ComboBox* per cambiare lingua. La

ComboBox è composta di oggetti *ImageListCell*, che permettono di avere come opzioni delle immagini. Sullo sfondo sono presenti due *ImageView* che, cambiando immagini alternatamente e variando il proprio valore di trasparenza, permettono di avere un'animazione gradevole alla vista. E' inoltre presente un *TitledPane* che avvisa l'utente della presenza del tutorial nel caso esso abbia selezionato l'opzione "Nuova Partita" senza averlo mai effettuato prima.

3.3.14 SettingsView

SettingsView è la classe del menu delle impostazioni. E' composta da una serie di *TitledPane*, uno sempre visibile contenente i *Button* per cambiare scheda, per applicare o resettare le impostazioni e per tornare al menu, uno per avvisare l'utente di cambiamenti non salvati, ed i restanti rappresentanti le varie schede delle impostazioni. Ogni scheda contiene dei *Text* per descrivere le varie impostazioni e una serie di componenti (*CheckBox*, *Slider*, *TextField* e *ColorPicker*)per modificarle. Nella scheda di personalizzazione sono inoltre presenti degli *ImageView* per mostrare all'utente come si presenteranno tabellone e pedine con i colori scelti; gli *ImageView* dei minitabelloni hanno inoltre delle animazioni per simulare la scelta tramite tastiera delle punte. Nella sezione *Comandi*, i *TextField* sono utilizzati in maniera atipica: quando uno di essi viene selezionato con il mouse si mette in attesa della pressione di un tasto, e cambia il proprio valore in base ad esso.

3.3.15 TutorialView

TutorialView è la classe che rappresenta la schermata del tutorial di JMahbusa. Essa estende *DynamicGameBoard*, ereditandone tutto quel che riguarda il tabellone. I messaggi relativi al tutorial stesso sono mostrati utilizzando due *AnchorPane* contenenti un *Rectangle* e una *Label* ciascuno. La scelta di utilizzare due *AnchorPane* anziché uno solo è dovuta alla volontà di avere un'animazione di scomparsa dell'ultimo messaggio e di comparsa simultanea del successivo. Alla fine del tutorial inoltre viene visualizzato un *TitledPane* contenente un messaggio di congratulazioni e la scelta fra tornare al menu principale o iniziare una partita di JMahbusa. Il tutorial è interrompibile in qualsiasi momento pigiando un *Button* posizionato in basso a destra per tornare al menu principale. Il ridimensionamento delle componenti di *TutorialView* è delegato alla classe *TutorialViewRedraw*.

3.4 Librerie Utilizzate

Nello sviluppo di JMahbusa si sono utilizzate le seguenti librerie aggiuntive

- *Ini4j*, che permette di leggere e scrivere i file *.ini* [4]
- *JSONSimple*, che permette di leggere e scrivere i file *.json* [5]
- *Reflections*, che semplifica l'utilizzo delle funzionalità di reflection native di Java[6]

3.5 Problemi Riscontrati

Si riportano di seguito i principali problemi riscontrati nello svolgimento del progetto JMahbusa.

3.5.1 Gestione del tabellone nella schermata di caricamento partita

Nelle prime build del progetto si era scelto di mostrare lo stato del tabellone relativo ad un salvataggio catturandone una schermata in fase di salvataggio, convertendolo in testo all'interno del file di salvataggio e riconvertendolo in immagine quando necessario. Ciò, pur funzionando, comportava i seguenti svantaggi:

- Una dimensione troppo elevata dei file di salvataggio
- L'assenza di reattività ai cambiamenti dei colori di tabellone e pedine
- La risoluzione dell'immagine era vincolata alla risoluzione della finestra di JMahbusa al momento del salvataggio

Per ovviare a questi svantaggi si è scelto di adottare la struttura di superclassi e sottoclassi *BoardView, DynamicBoardView, eccetera*, di estendere *BoardView* in *LoadGameView* e di memorizzare nel file di salvataggio la stessa matrice di interi utilizzata in *BoardLogic*.

3.5.2 Memory Leak in alcune Scene

Effettuando un profiling di JMahbusa tramite VisualVM è emerso che delle istanze di *MainMenu*, *SettingsView* e *GameView* persistevano in memoria anche dopo la loro chiusura. Ciò portava ad un graduale aumento delle dimensioni dello heap fino ad un eventuale crash dell'applicazione una volta raggiunto il limite

massimo di memoria allocabile. Una delle cause del memory leak era l'utilizzo di oggetti Timeline a cicli infiniti senza istruzioni di stop: la Timeline continuava la propria esecuzione anche una volta cambiata schermata, fermandosi solo in caso di chiusura dell'intera applicazione. La soluzione è stata aggiungere dei comandi di stop ai metodi di uscita dalle Scene. Un'altra causa del memory leak, nel caso di *MainMenu*, era l'utilizzo di *MenuBar* e *MenuItem* per la scelta della lingua. La soluzione del problema è stata utilizzare al loro posto una *ComboBox*.

3.5.3 Ridimensionamento delle Scene

In alcune Scene di JMahbusa il ridimensionamento della finestra non avveniva in maniera abbastanza reattiva: pigiando i pulsanti "massimizza" o "ripristina" alcune componenti non assumevano posizione e dimensioni corrette, comportandosi come se la dimensione della finestra fosse un'altra. Il ridimensionamento delle componenti problematiche avveniva in maniera derivata da altre componenti, in base quindi ad informazioni obsolete. Il problema è stato risolto derivando tutte le dimensioni e posizioni dalle dimensioni della finestra.

4. Estendibilità del programma

Di seguito sono riportate due possibilità di estensione di JMahbusa.

4.1 Aggiunta di ulteriori lingue

JMahbusa offre la possibilità all'utente poliglotta di aggiungere una nuova lingua al programma senza dover modificare o aggiungere del codice, eseguendo questi semplici passi:

- Nella cartella /languages/:
  - Aggiungere una nuova associazione chiave-valore al file *supportedLanguages.ini*, nella sezione *[Languages]* (esempio: "supportedLanguage = **EXP**")
  - Nel caso la lingua si legga da destra verso sinistra, aggiungere una nuova associazione chiave-valore allo stesso file nella sezione *[RightToLeft]* (esempio: "language = **EXP**")
- Nella cartella /languages/strings/:
  - Creare una copia di uno dei file Strings e rinominarlo cambiando la parte dopo l'underscore (es. STRINGS_**EXP**.ini)
  - Modificare il file STRINGS_**EXP**.ini sostituendo ai vari valori delle chiavi la nuova traduzione
- Nella cartella /languages/flags/ aggiungere un file png (si consiglia di dimensione 80x80) e rinominarlo seguendo la nomenclatura degli altri file (es. flag_**EXP**.png)

4.2 Aggiunta o modifica del tutorial

Il tutorial di JMahbusa può essere modificato, o addirittura completamente sostituito, andando ad agire opportunamente sul package *jmb.logic.tutorial*. JMahbusa infatti costruisce il proprio tutorial scandendo il package alla ricerca di classi concrete che estendano *ComparableTutorialStage* (e quindi implementino *TutorialStage*), e formando una lista di oggetti *ComparableTutorialStage* ordinati in base ai valori dell'attributo *stageIndex*.

5. Bibliografia

[1] Wikipedia. Backgammon — Wikipedia, l'enciclopedia libera, 2021. [Online https://it.wikipedia.org/wiki/Backgammon].

[2] Wikipedia. Backgammon — Wikipedia, the free encyclopedia, 2021. [Online https://en.wikipedia.org/wiki/Backgammon].

[3] Wikipedia. Tables (board game) — Wikipedia, l'enciclopedia libera, 2021. [Online https://en.wikipedia.org/wiki/Tables_(board_game)].

[4] Ivan Szkiba. Ini4j, documentation overview, 2011. [Online https://ini4j.sourceforge.net/apidocs/index.html].

[5] Google. Json, documentation google, 2008. [Online https://code.google.com/archive/p/json-simple/].

[6] Ronma. Reflections, documentation github (ronomamo). [Online https://github.com/ronmamo/reflections].