Relazione JBlackJack

Diana Pamfile, matricola 1943337

Canale MZ

5 febbraio 2025

Indice

1 Introduzione

Il famoso gioco "BlackJack" è un gioco di carte che ha due attori principali, il banco e il giocatore. Ogni giocatore inizia la partita con due carte in mano e può continuare a richiedere carta durante il proprio turno. L'obiettivo è quello di fare un punteggio maggiore del banco ma comunque inferiore o uguale a 21. Il banco inizialmente ha solo una carta visibile agli altri giocatori, mentre la seconda carta è girata e verrà svelata solo dopo che tutti i giocatori avranno svolto il proprio turno. In una partita di BlackJack si può fare il cosidetto "Blackjack" nel caso in cui le prime due carte che il giocatore ha in mano, formano un punteggio uguale a 21. Inoltre, nel caso in cui un giocatore fa più di 21, si dice che ha "sballato", in inglese "bust", e perde la propria mano indipendentemente dal punteggio che il banco farà.

2 Struttura del progetto

Prima di procedere con la descrizione dell'implementazione vera e propria del gioco, ritengo sia opportuno soffermarsi sulla struttura del progetto. Il progetto è suddiviso in cartelle. Le cartelle principali sono le seguenti:

- "build": questa cartella contiene tutti i file che sono stati generati durante la compilazione del progetto. All'interno di questa cartella si trovano delle sottocartelle che contengono: i file compilati generati a partire dal codice sorgente, i file generati automaticamente, la cartella principale per i file JAR (ovvero Java ARchive, uno strumento di archiviazione generale per distribuire ed eseguire programmi Java, in quanto contiene tutte le classi di un'applicazione), i file temporanei e le risorse (ad esempio le immagini).

- "gradle": la cartella contiene le risorse e i file affinchè Gradle funzioni correttamente. Si occupa inoltre anche della gestione delle dipendenze, separando il flusso del lavoro.

- "src" : è il "cuore" del progetto. All'interno di questa cartella troviamo la cartella "main" e le sue sottocartelle "data", "java" e "resources". La cartella data contiene un file "UserData.txt" che contiene le informazioni che riguardano il giocatore, quali il nome, l'avatar, il totale delle partite, e così via. La cartella "resources" contiene le immagini necessarie per la costruzione visiva del gioco e i file audio. La cartella "java" contiene 5 cartelle molto importanti, quali: "api" che semplifica la comunicazione di pacchetti di dati quando la view viene notificata di eventuali aggiornamenti tramite il meccanismo di notifica Observer-Observable, la cartella "exception" per una personalizzazione dei messaggi di eccezione con il fine di comprendere al meglio dove un problema ha origine, e le cartelle "view", "model" e "controller", che seguono proprio il pattern MVC adottato in questo progetto. In questa cartella sono contenuti anche due file, ovvero il Main e il Launcher che fanno parte della logica di avvio del gioco, in

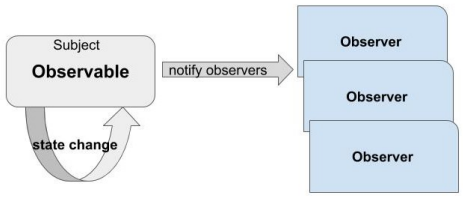
quanto Main contiene il metodo main che chiama il metodo main della classe Launcher, avviando il gioco. Mentre la classe Launcher estende "Aplication" di JavaFx e gestisce la creazione della finestra principale e imposta come prima scena quella del Menu.

Ho ritenuto che questa struttura potesse aiutare a gestire meglio le varie parti del progetto, in quanto ogni componente ha una responsabilità chiara e separata.

3 Pattern utilizzati

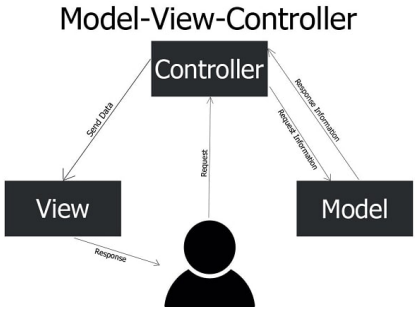
Per l'implementazione di questo progetto sono stati utilizzati i seguenti Design Pattern:

- Observer-Observable Pattern: questo pattern permette di notificare i soggetti interessati allo stato di un oggetto, ogniqualvolta avviene un cambiamento di tale stato. Il design pattern Observer-Observable definisce a tutti gli effetti una dipendenza uno-a-molti tra oggetti.
  Le componenti sono due: la prima componente è `Observer`, ovvero l'oggetto che ha interesse e aspetta di essere informato quando lo stato dell'osservabile cambia, la seconda componente è l'`Observable`, chiamato anche soggetto, possiede uno stato che nel corso del tempo, per diversi motivi, può cambiare, quando il suo stato cambia, provvede a notificare tutti i suoi osservatori.
  Riconducendoci al progetto, il soggetto osservabile è la classe "BlackJackGame", la quale si trova nel model e rappresenta una partita di BlackJack, impkementando l'interazione tra i giocatori e il banco, ma anche la gestione del mazzo. L'observer è la classe SceneManager, che si trova rispettivamente nella view e viene notificata di ogni cambiamento relativo allo stato della partita e dei giocatori.
  La classe SceneManager implementa il metodo `update` che si occupa di gestire i pacchetti ricevuti, mostrando a schermo quanto richiesto per una maggiore fluidità del gioco.



- MVC Pattern: il design pattern MVC è un pattern architetturale e viene usato principalmente nella programmazione orientata agli oggetti.
  Il principio alla base di questo pattern è quello di non avere dipendenze tra l'interfaccia utente e il modello, con lo scopo di avere un'architettura il quanto più flessibile. Possiede 3 componenti.

Il `model`, che contiene la logica del gioco ed è resposabile anche dei metodi per la gestione dei dati (da qui la scelta di avere la classe "SaveData" nel model piuttosto che nella view). Nel model del progetto sono presenti le classi che rappresentano i giocatori AI, il giocatore reale, il banco, ma anche il mazzo e la mano dei giocatori. Attraverso l'uso delle enumerazioni, sono state gestiti il seme e il valore di ogni carta, in quanto sono degli insiemi finiti e prefissati. Il model, infine, contiene la classe BlackJackGame che rappresenta una partita del gioco, con metodi specifici riguardo l'interazione dei vari oggetti presenti nel progetto, ad esempio il metodo `setUpGame` che si occupa di aggiungere il numero di giocatori desiderato alla partita e di mescolare il mazzo di carte. Il model è indipendente e non ha bisogno della view per funzionare.
La seconda componente è il `controller` che è la componente intermediaria tra view e model. Il controlle è una componente riutilizzabile soltanto a metà in quanto aggiorna la view e manipola il modello.
La terza componente è la `view`, essa rappresenta l'interfaccia del gioco. Si occupa dell'interazione con gli utenti, ad esempio quando i giocatori chiedono carta manda un input al controller che si occupa di manipolare il model. Questa componente non è riutilizzabile in quanto si occupa di mostrare i dati del modello all'utente e di inviare i vari input dell'utente al controller.



- Singleton Pattern: è un pattern che appartiene alla categoria dei design pattern creazionali. Lo scopo di questo pattern è quello di garantire che venga creata una sola istanza di una determinata classe. Il singleton pattern è stato adottato all'interno della classe SceneManager, presente nella view, per garantire che esista una sola istanza in tutto il gioco. Il pattern Singleton è stato adottato anche per la classe Deck, con lo stesso fine.

- Builder Pattern: è un pattern appartenente alla categoria dei design pattern creazionali, esso permette di separare la costruzione di un oggetto dalla sua rappresentazione. Nel progetto, il builder pattern è stato adottato per assicurare una corretta creazione del SceneManager e che abbia uno Stage valido prima di essere creato.

4 Meccaniche di gioco

4.1 Player, Computer Player e Dealer

La classe Player rappresenta un giocatore astratto, ha una serie di attributi che rappresentato la mano del giocatore, il punteggio, avatar, username, ma anche gli attributi `standing`, utilizzato per la corretta gestione dei turni dei vari giocatori in una partita, e `type` per permettere la gestione dei pacchetti di dati inviati alla view.
La classe `RealPlayer` estende la classe astratta Player. Si occupa di manipolare i dati dell'utente e quindi salvarli quando opportuno o di prelevarli nel momento necessario, ad esempio quando serve leggere lo username dell'utente per mostrarlo durante la partita. Oltre agli attributi che possiede un generico player, il real player ha anche gli attributi riguardanti le partite giocate, vinte, perse o la scommessa che ha deciso di puntare, memorizzando il totale delle fiches che il giocatore ha ancora disponibile. Questa classe ha anche un metodo che si occupa di incrementare il livello attuale del giocatore, ovvero ogni 5 partite vinte si sale di livello.
La classe `ComputerPlayer` estende anch'essa la classe astratta Player, ed effettua un Override del metodo hit, implementendo la logica secondo la quale un giocatore bot chiede carta, ovvero se ha un punteggio minore o uguale a 15, altrimenti decide di stare.
Il `Dealer` è rappresentato esattamente come un Player astratto nel gioco, ma con un nome e un avatar sempre prefissati.

4.2 Mazzo di carte

Il mazzo di carte, o `deck`, è implementato usando il pattern singleton per avere un solo un mazzo durante tutta la partita. Ogni deck è una lista di GameCard.
La classe `GameCard` si occupa di rappresentare una carta nel gioco, contiene gli attributi `Value` e `Suit` per la carta in sè, e l'attributo `visible` che serve per decidere se la carta viene mostrata o meno, inserito per poter gestire la carta girata del dealer fino al suo turno.
Tornando alla classe Deck, essa contiene i metodi per creare il mazzo, attraverso l'utilizzo di uno Stream, per mescolare il mazzo e quindi garantire una distribuzione casuale delle carte ad ogni giocatore, il metodo per rimuovore una carta dal mazzo ogniqualvolta venga distribuita una carta, ed infine il metodo refillDeck che serve per ricreare il mazzo nel caso in cui siano rimaste meno di 20 carte, in quanto in caso contrario un giocatore furbo potrebbe ricordarsi le carte uscite e decidere in base a quelle se chiedere o meno carta.

4.3 BlackJackGame

La classe estende `Observable` e rappresenta una partita. La classe `BlackJackGame` contiene i campi Deck, il Dealer e una lista di giocatori. I metodi di questa classe servono per creare una vera e propria partita, analizzandoli troviamo:

- setUpGame: Il metodo che fa il setup del gioco. Viene aggiunto il giocatore reale come primo giocatore nella lista per una più semplice manipolazione della lista dei giocatori.
  I giocatori bot sono scelti in maniera casuale. Viene creata una lista `availablePlayers` per contenere i giocatori disponibili, si usa uno Stream per creare un numero di bot (numero che viene scelto dall'utente a inizio partita), ogni bot viene inizializzato con un username e un avatar scelto da una lista predefinita di username e avatar, successivamente la lista availablePlayers viene mescolata per garantire una scelta casuale dei bot.
  Viene mescolato il mazzo e viene notificata la view dei cambiamenti avvenuti, mandando la lista dei username, degli avatar e il numero di giocatori in partita.

- drawInitialCard: questo metodo serve per distribuire le prime due carte che ogni giocatore ha in mano ad inizio partita. La view viene notificata delle carte e del punteggio che ogni giocatore ha in mano. Le carte che vengono distribuite al dealer sono una con visible settato a true e una settato a false, che permette alla view di poter "nascondere" la carta e disegnarne il retro.

- giveCard: è un metodo che da la carta al dealer.

- hit: viene chiamato quando un giocatore chiede carta dal mazzo.

- dealerPlay: gestisce la logica secondo la quale il dealer chiede carta, ovvero quando ha un punteggio inferiore a 17.

- checkWin: questo metodo prende la lista dei giocatori e il dealer, confrontando il punteggio di ogni giocatore con quello del dealer viene determinato il vincitore, seguendo le regole di BlackJack. Per il giocatore utente, viene fatto un controllo a parte per aggiornare anche il totale delle partite, quelle vinte, perse e controllare se va incrementato il livello.

- resetGame: permette di poter rigiocare quando l'utente vuole effettuare una nuova partita, "pulisce" la mano dei vari giocatori e del dealer, e inizializza un nuovo mazzo.

Esistono poi i metodi per controllare se un giocatore ha "sballato" quindi ha fatto "bust" oppure se ha fatto "blackjack", per mostrarne il messaggio all'utente dalla view. Infine i metodi riguardanti la manipolazione dei dati del giocatore reale.

4.4 Turno del giocatore

Il turno del giocatore viene gestito dal controller. L'attibruto standing viene messo a false e il turno può iniziare. C'è una divisione nella gestione del turno tra quella dell'utente giocatore e quella dei bot.

- Turno RealPlayer: si controlla come prima cosa se il giocatore ha fatto blackjack, in caso affermativo ne vengono incrementate le vittorie e il standing viene rimesso a true, perché il giocatore ha vinto. In caso negativo, viene usata una variabile per memorizzare la scelta del giocatore, ovvero se decide di chiedere carta, quindi se ha premuto il pulsante "Hit" oppure no, quindi ha premuto il pulsante "Stand". Dopo ogni carta che il giocatore chiede, si controlla se ha fatto bust, in caso affermativo, il giocatore ha automaticamente perso. Altrimenti si aspetta la fine del turno del dealer per decretare la vincita o la perdita del giocatore.
- Turno ComputerPlayer: la logica è simile a quella descritta per un giocatore reale, ma non si aspetta una scelta del giocatore, il bot giocherà seguendo la regola "chiedi carta se il punteggio è inferiore a 16".

4.5 Loop di gioco

Il loop di gioco è presente nel controller e segue delle regole ben precise per la gestione di una partita.
Viene inizializzato il gioco con il metodo del model setupGame, e si aspetta che l'utente scelga la puntata. La puntata del giocatore viene settata e vengono momentaneamente sottratte le fiches dal totale, che saranno radoppiate nel caso di vincita.
Solo a questo punto vengono distribuite le prime due carte a tutti i giocatori e al dealer. Si chiama ora il metodo `handleTurn` che gestisce i turni dei giocatori in questa partita, viene fatto giocare prima il RealPlayer, poi i ComputerPlayer ed infinte il dealer, ognuno secondo la propria logica di gioco, spiegata precedentemente. Dopo che il dealer ha giocato si determina la vittoria o sconfitta di ogni giocatore.
Attraverso l'uso di una variabile, si controlla se l'utente desidera effettuare una successiva partita e in caso affermativo si ripulisce il tavolo da gioco, ovvero il game, e si ricomincia distribuendo due carte ad ogni giocatore. Il loop continua fino a quando l'utente desidera continuare a giocare.
Nel loop di gioco vengono chiamati metodi di "sleep" del Thread per permettere una fluida gestione delle scelte del giocatore e quindi per la comunicazione con la view, parte con cui l'utente interagisce.

5 Profilo utente

Nel gioco è presente anche un profilo utente, al quale si può accedere attraverso il pulsante "Profile" dal menu.
Il profilo utente contiene le informazioni riguardanti le statistiche del gioco, quali partite totali, vinte, perse e le fiches a disposizione. Le fiches hanno un valore iniziale impostato a 5.000 e vengono di volta in volta incrementate o decrementate. Nel profilo è presente anche il livello del giocatore, l'avatar e lo username. Un utente può decidere di cambiare il proprio username inserndone uno nuovo e può scegliere tra 4 avatar disponibili.

Per la gestione dei dati del profilo è stata creata la classe SaveData che si occupa di leggere i dati o di sostituirli in un file txt presente nel progetto. Ogni volta che c'è bisogno di modificare/leggere dati, sono invocati i metodi di tale classe. La view permette inoltre di vedere in tempo reale la modifica dello username e dell'avatar e di cambiare scelta in caso l'utente non sia soddisfatto.

6 Implementazione

Per il progetto scelto sono state adottate alcune scelte nell'implementazione.

6.1 Stream

L'uso degli stream si trova nella creazione dei giocatori bot e per la creazione del deck.

6.2 JavaFx

L'adozione di JavaFx come framework per la creazione dell'interfaccia grafica è motivata dalla volotà di avere una gestione fluida delle animazioni, da una possibilità di personalizzare gli elementi attraverso l'uso dei CSS, e dalla possibilità di usare classi, quali VBox, HBox, Label, ecc..., che hanno reso più semplice e leggibile la definizione dell'interfaccia.

6.3 Animazioni

All'interno del gioco le animazioni sono state utilizzate per girare la carta del dealer e per le carte che vengo distribuite ai giocatori quando un giocatore chiede carta.
Per creare le animazioni viene utilizzato un metodo presente nella view "drawAnimation" che prende il retro di una carta, la fa partire dal mazzo e a metà strada, tra mazzo e giocatore, ruota questa immagine, rivelando la carta pescata.
Lo stesso concetto è stato applicato per rivelare la carta del dealer. Ogni carta è disposta sopra le altre con un overlay che consente di mostrare tutte le carte che un giocatore ha in mano ma senza riempire l'intera view di carte disegnate.
Il codice presenta alcuni delay per rendere più fluida la partita.

6.4 Audio

Per la riproduzione dell'audio ho utilizzato la classe "MediaPlayer" fornita da JavaFx.
Questa classe mi ha permesso l'aggiunta di un background sonoro in tutte le schermate senza interruzioni, ma anche di poter regolare il volume attraverso un utilizzo di un Slider.
C'è anche un audio che riproduce il suono di una carta pescata che viene chiamato alla fine di ogni animazione, con il fine di rende il gioco un po' più realistisco e immersivo.

7 Conclusione

Una delle principali migliorie che si possono fare all'interno di questo progetto è la gestione del profilo utente. Sarebbe stato molto più fuzionale lasciare la gestione completa di tutti gli attributi, il loro incremento/decremento, ad una sola classe.
Un altro possibile miglioramento è quello riguardante la classe Game della view, in quando sarebbe più utile avere una seconda classe per la creazione delle Box. Attraverso questa divisione è possibile avere una manutenibilità del codice.
E' possibile inoltre implementare in futuro le diverse varianti di blackjack esistenti e inserire nelle settings la possibilità di scegliere a quale giocare.

8 NOTE

Nella cartella `build`, `libs` è presente il JAR del gioco per eseguirlo da terminale nel caso vi siano problemi.
Inoltre, allego il link al mio personale github nel caso vi siano problemi nell'apertura dello zip del progetto.