POLITECNICO DI MILANO

DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

Corso di Laurea in Ingegneria informatica

Relazione di progetto del corso 088851 Progetto Software

.

Davide Ghisolfi
Matricola: 985669

A. A. 2023/2024

Indice

1

Introduzione

La seguente relazione rappresenta la principale documentazione riguardante il software presentato come consegna del corso *088851 Progetto Software* e fornisce sia una descrizione generale delle specifiche sia una descrizione più tecnica dell'architettura del progetto. Lo scopo del software è l'implementazione del gioco di carte *Scopone* nella variante *a dieci carte* nella sua versione a quattro giocatori e squadre da due giocatori in modo che l'utente possa giocare da solo oppure online insieme ad altri utenti. Il software è interamente scritto in linguaggio Java e utilizza la libreria JavaFX e fogli di stile CSS per l'implementazione dell'interfaccia grafica.

Acronimi e abbreviazioni durante la descrizione, per brevità i giocatori controllati dal computer sono spesso indicati con il termine **bot** o **giocatore CPU**.

1.1 Strumenti utilizzati

- Sviluppo: Java JDK 21.0.2 + Maven 3.9.5
- Unit Test: JUnit 5.10
- Diagrammi UML: Draw.io
- Sequence diagrams: Mermaid 10.9.1
- Libreria grafica: JavaFX 21

1.2 Requisiti e istruzioni per l'uso

Il lato server del software è compatibile con i sistemi operativi Windows, Linux, MacOS con processore Intel e MacOS con processore silicon e richiede l'installazione di Java JDK almeno alla versione 21. Il lato client è compatibile solamente con sistemi operativi Windows.

Esecuzione dell'eseguibile

Il software non prevede una fase di installazione. Aprendo l'eseguibile `Scopone.jar` come un normale file, si aprirà la vista GUI Client. Per avviare un'istanza del server è invece necessario aprire l'eseguibile da riga di comando e navigare nella directory dove si trova l'eseguibile e digitare il seguente comando:

```
/$ java -jar Scopone.jar server
```

In questo modo si aprirà un'istanza del server sulla porta *default* 12000.

Altre opzioni da riga di comando

- aggiungendo un numero intero tra 0 e 65535 è possibile specificare la porta su cui avviare il server:

  ```
  /$ java -jar Scopone.jar server 3000
  ```

  il comando avvia il server sulla porta 3000;

- specificando la porta e aggiungendo il parametro `d` come terzo parametro, si può attivare la modalità *debug* che permette di visualizzare informazioni aggiuntive sui giocatori CPU e gli eventi che accadono all'interno del server:

  ```
  /$ java -jar Scopone.jar server 12000 d
  ```

1.3 Specifica

Il software in questione è *distribuito* e si divide pertanto in una parte *Client* destinata all'utente e una parte *Server* che permette la connessione contemporanea di più applicativi client che si trovano nella stessa rete locale in cui si trova anche il computer che sta eseguendo il server. La parte Server non presenta nessuna interfaccia a meno delle informazioni di log passate al terminale in cui viene eseguito. La parte Client, invece presenta un'interfaccia grafica (GUI) intuitiva che permette all'utente di impostare le opzioni relative alla connessione al server (indirizzo IP e porta), selezionare il tipo di partita a cui vuole partecipare (offline/online), di visualizzare le carte che vengono giocate durante la partita, il punteggio dell'incontro e il nome dei giocatori che sta affrontando oltre che a decidere quale carta piazzare tra quelle presenti nella sua mano. Si assume che un utente che voglia connettersi ad un server ne conosca l'indirizzo IP e la porta.

1.3.1 Funzionalità

Una sessione di gioco prevede più partite: i punti conquistati dalle due squadre durante le partite precedenti sono portati nella partita successiva. Come stabilito dalle regole del gioco, se i giocatori decidono di giocare più partite consecutive, la prima squadra che raggiunge o supera *undici* punti vince l'incontro e provoca la chiusura della sessione. Nelle partite successive il primo giocatore diventa il giocatore successivo al primo giocatore della partita precedente.

Funzionalità aggiuntive modalità offline

Durante una partita offline, il giocatore può in ogni momento cambiare la difficoltà della CPU accedendo all'apposita opzione posta nel menù di gioco.

Funzionalità aggiuntive modalità online

Il server dà la possibilità di cominciare la partita appena due utenti si connettono riempiendo i posti rimanenti con giocatori CPU in modo da permettere comunque di giocare anche nel caso in cui non fossero presenti quattro utenti: nel caso in cui la partita inizi con solo due utenti, le squadre vengono formate in modo tale da porre i due utenti in squadre diverse. In caso di disconnessione, il server rimpiazza il giocatore disconnesso con un giocatore controllato dal software. Nel caso inverso in cui invece il software stia controllando un giocatore, il server permette ad un eventuale nuovo utente connesso di entrare in partita rimpiazzando il bot. Al termine di una partita, un qualsiasi giocatore può decidere di continuare il gioco facendo cominciare una nuova partita. Durante un qualsiasi momento della partita, qualsiasi utente connesso può cambiare la difficoltà della CPU accedendo all'apposita opzione posta nel menù di gioco: nel momento in cui avviene la selezione, gli altri giocatori vengono notificati del cambiamento.

1.4 Regole implementate

Di seguito viene riportata una breve spiegazione delle regole del gioco implementate nel software al fine di evitare ambiguità legate alla presenza di numerose varianti regionali. I giocatori giocano a coppie di due e a posizioni opposte. All'inizio del gioco ad ogni giocatore vengono distribuite 10 carte. Procedendo in senso antiorario ogni giocatore deve piazzare una carta sul tavolo: se la carta giocata è di valore uguale a quello di una carta o alla somma dei valori di più carte che si trovano sul tavolo, il giocatore prende la carta da lui giocata e quella o quelle di valore corrispondente, altrimenti lascia la carta sul tavolo. Nel caso in cui esistano più combinazioni di presa, il giocatore è obbligato a prendere la combinazione con il numero minore di carte. Il gioco prosegue fino all'esaurimento delle carte: le carte che rimangono sul tavolo al termine della partita sono prese dalla squadra che ha preso per ultima. Nel momento in cui un giocatore riesce a prendere tutte le carte presenti sul tavolo realizza una **scopa**. Al termine della partita si calcola il punteggio e si può decidere di proseguire con un'altra manche. Nel momento in cui una squadra raggiunge o supera *undici* punti, vince la partita. Nelle

manche successive il primo giocatore diventa il giocatore successivo al primo giocatore della partita precedente.

Punteggio esistono cinque tipi di punti che una squadra può fare:

- *Scopa*: ogni scopa fatta vale un punto;
- *Denari*: la squadra che ha preso più carte del seme di denari guadagna un punto. In caso di parità nessun punto viene assegnato;
- *Carte*: la squadra che ha preso in assoluto più carte guadagna un punto. In caso di parità nessun punto viene assegnato;
- *Primiera*: la primiera è il maggior punteggio che una squadra può costruire con quattro carte di seme diverso con la seguente gerarchia: 7, 6, 1, 5, 4, 3, 2, figure. Una primiera è invalida se una squadra non ha almeno una carta per seme. La squadra con la primiera più alta guadagna un punto. Se le squadre hanno la stessa primiera nessun punto viene assegnato;
- *Settebello*: la squadra che prende il *settebello*, ovvero il sette di denari guadagna un punto.

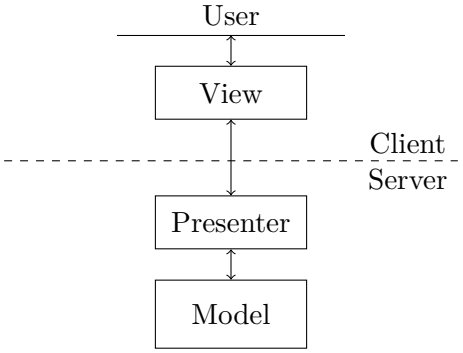
2

Descrizione dell'architettura

In questa sezione verranno approfondite l'architettura del software e le principali decisioni adottate durante lo sviluppo al fine di offrirne una visione top down più tecnica.

2.1 Model-View-Controller

L'architettura di tutto il software è interamente basata sul pattern architetturale *Model-View-Controller* nella sua variante **Model-View-Presenter** che divide nettamente il modello e la vista mediante il presenter che, svolgendo la funzione di controllore, riceve gli eventi generati dalla vista e modifica il modello di conseguenza per poi ritornare all'utente lo stato aggiornato. Questo modello, rispetto ad altre varianti, permette infatti di rimuovere qualsiasi tipo di logica dal client lasciando ad esso solamente il compito di comunicare all'utente le informazioni ricevute e riducendo di conseguenza la probabilità di errori logici.



Lo schema mostra la suddivisione dell'architettura tra Client e Server durante una partita online

2.2 Design pattern utilizzati

Di seguito sono riportati i design pattern utilizzati durante lo sviluppo e il motivo per cui sono stati adottati.

Pattern Strategy la selezione della difficoltà del bot si serve del pattern strategy al fine di facilitarne la variazione a runtime: oltre a ciò, nel rispetto del principio SOLID Open/Closed, l'adozione del pattern facilita l'eventuale implementazione di ulteriori livelli di difficoltà.

Pattern Singleton le classi che gestiscono la connessione lato client e lato server e la classe che effettua modifiche sul modello utilizzano il pattern singleton al fine di evitare la creazione di più di un'istanza e fornire un accesso globale alle funzionalità che contengono.

Pattern Decorator la comunicazione client/server avviene mediante classi serializzate chiamate *eventi*. Come spiegato in seguito, queste classi contengono una serie di informazioni che vengono interpretate mediante *gestori* (Event handlers) dedicati. Alcuni di questi gestori devono conoscere l'interfaccia di comunicazione da cui l'evento proviene: per questo motivo, gli eventi per cui è necessario conoscere l'interfaccia mittente vengono *decorati* in fase di ricezione con tale interfaccia in modo da renderla disponibile all'handler. L'adozione di questo pattern permette di assegnare a runtime nuove funzionalità agli eventi solamente nei casi in cui è necessario.

Pattern Observer durante il gioco offline, al fine di mantenere distinti il controller (presenter) e la vista, la classe che controlla la vista durante la partita si pone come osservatore *virtuale* del modello mediato dal controller e reagisce di conseguenza ai suoi cambiamenti.

2.3 Comunicazione Client-Server

La comunicazione tra client e server utilizza il protocollo di rete *socket TCP/IP* e avviene mediante lo scambio di classi serializzate chiamate *eventi*: ogni evento contiene informazioni differenti e causa un cambiamento di stato nell'host che lo riceve. Questo cambiamento viene effettuato da *gestori di evento* o **Event handlers** invocati dal ricevente che in base alle informazioni contenute all'interno dell'evento, causano un cambiamento del modello se l'evento è rivolto al server oppure della view se l'evento è rivolto al client. Come accennato nella sezione dedicata ai pattern, un evento può essere decorato aggiungendo alle informazioni inviate dal mittente l'interfaccia di rete del mittente stesso: quest'ultima informazione è contenuta nella classe involucro *Connection* la quale fornisce un'astrazione del socket gestendo in modo sicuro la ricezione e l'invio delle classi serializzate. I due host gestiscono la ricezione degli eventi mediante un *Listener di eventi* (Event Listener) a cui è dedicato un thread e un *Ricevitore di eventi* (Event Receiver): quando un evento viene ricevuto da un'interfaccia di rete, l'Event Listener lo aggiunge in una coda e notifica l'Event Receiver che, interpretando l'evento, lo indirizza verso il gestore corretto. Siccome la gestione dell'evento potrebbe richiedere un tempo di

elaborazione non banale, la coda dell'Event Listener assicura che durante l'elaborazione di un evento, un altro evento in arrivo dalla rete non vada perso.

2.3.1 Fase di connessione del client

Quando un client richiede la connessione al server, il welcome socket del server compie le seguenti azioni:

1. crea una nuova istanza della classe *Connection*
2. aggiunge la connessione alla lista dei Client connessi
3. avvia un *Event Listener* su un thread dedicato
4. avvia una nuova *Ping-pong routine* al fine di assicurare la raggiungibilità del Client

Il client procede poi richiedendo al server di partecipare o cominciare una partita fornendo lo username con cui intende partecipare: il server valuta la richiesta e decide se accettare il giocatore ammettendolo alla partita oppure rifiutare la richiesta e disconnettere l'utente dalla partita.

La richiesta viene rifiutata in uno dei seguenti casi:

- se esiste un utente già connesso che ha fornito uno username uguale;
- se la partita non è in corso e il server è già connesso con quattro utenti;
- se la partita è in corso e nessun giocatore è controllato dal server.

2.4 Giocatore CPU

In questa sezione viene approfondita l'implementazione dei giocatori controllati dalla CPU. All'interno del controller di gioco, sia esso appartenente ad una partita online o offline, contiene un'istanza della classe *Bot*: durante il turno di un giocatore controllato dalla CPU, il controller invoca la classe *Bot* che, conoscendo le liste delle carte sul tavolo, le carte già giocate e, necessariamente, le carte nella mano del giocatore CPU di cui deve scegliere la carta, calcola per ogni carta nella mano un *peso* basato su diversi criteri e sceglie quella con il peso più alto. I criteri di scelta sono comuni alle tre difficoltà ma cambiano i pesi e le carte considerate nel calcolo degli stessi:

- difficoltà *facile*: il giocatore CPU valuta le carte conoscendo solamente le carte nella sua mano e le carte sul tavolo. In caso di tavolo vuoto, gioca la carta che ha in numero maggiore in mano. Negli altri casi calcola tutti i pesi e aggiunge ad ognuno una componente aleatoria scegliendo poi la carta con peso massimo: in questo modo è possibile che venga scelta la seconda carta migliore simulando così la scelta di un giocatore inesperto
- difficoltà *media*: come nel caso di difficoltà facile, il giocatore CPU valuta le carte conoscendo le carte nella sua mano e le carte sul tavolo ma si *ricorda* anche alcune carte già giocate. In caso di

tavolo vuoto, gioca la carta il cui numero appare più volte nella mano e nelle carte già giocate. Negli altri casi calcola tutti i pesi e sceglie la carta con peso massimo.

- difficoltà *difficile*: il giocatore valuta le carte conoscendo le carte in mano, sul tavolo e tutte le carte che sono già state giocate. In caso di tavolo vuoto, gioca la carta il cui numero appare più volte nella mano e nelle carte già giocate. Negli altri casi calcola tutti i pesi e sceglie la carta con peso massimo.

2.4.1 Criteri di scelta

Di seguito sono elencati i criteri secondo cui vengono pesate le carte. Ogni criterio restituisce un numero decimale che viene poi moltiplicato per un ulteriore *peso* in modo da dare più importanza ad alcuni criteri rispetto ad altri: nella tabella sottostante il campo *Priorità* specifica la il campo range indica tutti i possibili valori assumibili dal criterio prima di essere moltiplicato per il peso. I criteri che contengono *risk* diminuiscono o lasciano inalterato il peso totale mentre quelli che contengono *proficiency* possono alzarlo oppure ridurlo a seconda del caso.

Nome	**Priorità**	**Range**	**Descrizione**
inHandValueProficiency	bassa	$(0; 10]$	aumenta la priorità delle carte con numero basso nei primi turni, diminuisce durante la partita con andamento esponenziale
takenCardsProficiency	medio bassa	$[0; 5]$	aumenta la priorità della carta in maniera proporzionale a quante carte prende quando piazzata
doesScopaProficiency	alta	$\{0; 1\}$	aumenta la priorità delle carte che provocano una scopa
sevenProficiency	medio alta	$\{-1; 0; 1\}$	si applica solamente ai sette, ne aumenta la priorità solo se prende almeno una carta, altrimenti la riduce
takesSevenProficiency	medio alta	$\{-1; 0; 1\}$	aumenta la priorità della carta se prende un sette

goldProficiency	media	$\{0; 1\}$	si applica solamente alle carte di denari, ne aumenta la priorità solo se prende almeno una carta, altrimenti la riduce
takesGoldProficiency	media	$[0; 4]$	aumenta la priorità della carta in modo proporzionale alle carte di denari prese
scopaRisk	alta	$[-4; 0]$	diminuisce la priorità delle carte che danno la possibilità al prossimo giocatore di fare scopa. È proporzionale al numero di carte con cui il prossimo giocatore può potenzialmente fare scopa
sevenRisk	medio alta	$[-4; 0]$	diminuisce la priorità delle carte che danno la possibilità al prossimo giocatore di prendere con un sette. È proporzionale al numero di sette rimanenti