Matricola 1885652, Canale MZ [MDP 21-22], Simone Rufo

Design pattern adottati:

Per la realizzazione di questo progetto ho usato Il design pattern MVC (Model-View-Controller), così separando in tre package principali le classi:

MODEL: nel package model ho inserito tutte le classi che definivano la base dei dati del videogioco e la modellazione di questi (solo per definire lo stato iniziale di essi), dalla definizione di una carta astratta alla concretizzazione di essa nel videogioco (le carte colore e le carte jolly), il mazzo astratto e le sue due concretizzazioni (il mazzo di pesca e il mazzo di scarto), il giocatore astratto che definisce lo stato iniziale un giocatore di Uno, e le sue concretizzazioni: Giocatore principale (che aggiunge delle informazioni aggiuntive al giocatore), e Giocatore Comp (che rappresenta lo stato iniziale del bot che gioca).

VIEW: nel package view ho inserito tutte le classi ereditate dalla libreria Java Swing, che sono servite a dare una rappresentazione grafica dello stato del gioco in un determinato momento, come l'avatar e il nome di ogni giocatore le carte nella mano del giocatore principale e dei bot, un label che rappresenta il colore della carta giocabile.

Nella view ho deciso di utilizzare molte classi diverse che separassero un componente dall'altro dove possibile per rendere più leggibile e manutenibile il codice in caso di errori (ad esempio nella creazione del Pannello dei giocatori ci sono due classe interne che separano la mano dal pannello delle informazioni).

CONTROLLER: mentre nel package Controller ho inserito le classi che rispettivamente mettono in comunicazione Model e View, senza far interagire il model esplicitamente con la view, così da tener separata la parte grafica da quella logica.

Il controller appunto riceve gli input del giocatore e aggiorna rispettivamente model e view.

Il model fa riferimento al controller quando ha bisogno di ricevere i dati necessari per instanziare il Giocatore Principale (nickname,avatar,livello,partite giocate ecc..)

invece la view fa riferimento al controller quando deve rendere visibile a schermo che il turno del giocatore è stato aggiornato.

Mentre per tutti gli altri casi è sempre il controller che gestisce ed aggiorna i dati sia del model che della view, quindi è il gestore logico delle operazioni del gioco.

(il model rimane sempre allo scuro dell'esistenza della view).

Nel package ho creato tre classi controller diverse per gestire appunto tre diversi frame in modo più semplice:

Il frame del gioco (ControllerFrame);

Il frame delle info del giocatore principale (ControllerStats);

E il frame che avvia la partita e permette di visualizzare le informazioni relative al giocatore(ControllerSchermata).

Ho utilizzato il pattern Observer Observable ,realizzando l'interfaccia Subject, ovvero la classe che registrerà gli Observers inviando a loro delle notifiche,e la classe Observer anch'essa è un'interfaccia che possiede richiede l'implementazione del metodo update.

Ho utilizzato il pattern in modo che il mazzo di pesca ed il mazzo di scarto nel model venissero osservati dalle loro corrispettive classi rappresentanti nella view (LabelMazzoScarto e BottoneMazzoPesca). Lo scopo era quello di aggiornare la grafica delle classi osservatrici ogni volta che uno dei due mazzi cambiasse stato (quando una carta viene giocata o pescata).

Ho adottato lo stesso principio per il pannello Info di ogni giocatore, implementando Subject alla classe Astratta Giocatore, mentre tutti i Pannelli info dei giocatori implementano l'interfaccia observer, registrando così il cambio di nome.

Ho poi riutilizzato il metodo update dei pannelli info per aggiornare il turno tramite il controller, ad ogni aggiornamento di turno questo viene chiamato.

Il controller invece si occupa dell'iscrizione degli obsevers al proprio subject(tramite la chiamata del metodo register dell'interfaccia), così che il model continua a non essere dipendente in alcun modo dalla view.

L'ultimo pattern che ho utilizzato è stato il Singleton, questo per permettere alle classi che usufruiscono di questo pattern di essere istanziate solamente una volta.

Le classi che fanno uso di questo pattern sono : il Giocatore Principale, poiché ce ne può essere solamente uno attivo durante il flusso del gioco.

Con lo stesso principio le tre classi controller fanno uso del pattern.

Uso degli Stream:

Ho utilizzato gli stream con parsimonia, senza "forzare" il loro uso.

Questi sono stati utilizzati maggiormente nella classe ControllerGame, nei metodi di aggiornamento del turno (nel metodo Inverti, nel metodo di inizializzazione del turno..)

Il loro uso è stato correlato all'utilizzo di Map o List, per ordinare appunto il flusso di dati o più semplicemente per collezionare questo nel campo d'istanza del turno.

Altri utilizzi frequenti sono stati quelli di filtrare le carte per ricavare quelle giocabili e filtrare i giocatori per ricavare il giocatore principale.

Altre note progettuali:

A discapito della classe consigliata per il funzionamento dell'audio, ho utilizzato una libreria alternativa (javax.sound.sampled), a causa di un errore di import che non sono riuscito a risolvere.

Ho fatto in modo che la partita di Uno si concludesse nel momento in cui uno dei giocatori non abbia più carte nella mano. Questo perché diminuisce la durata del gioco rendendo il gioco a mio avviso più entusiasmante.

I bot giocano in maniera casuale (rispettando ovviamente le regole del gioco), questo per cercare di creare sempre partite differenti, la pecca di ciò è che a volte i bot giocano in maniera autodistruttiva.

Ho creato la classe Utilities che si trova nel controller di supporto alle funzioni di lettura e scrittura dei file che vengono utilizzate nel package controller e model.

Spero che il gioco sia godibile, la realizzazione è stata molto didattica e divertente (non sempre!), Grazie dell'impegno.