BLACKJACK

Relazione Progetto

Programmazione e Modellazione a Oggetti

Comelli Pietro

matricola: 313732

Benvenuti Daniele

matricola: 315478

Neve Luca

matricola: 315141

1 Analisi

Il gruppo si pone come obiettivo lo sviluppo di un software per simulare una sessione del gioco del Blackjack.

Utilizzando le carte francesi, il gioco consiste nell'avvicinarsi il più possibile al punteggio 21. Il giocatore vince quando il suo punteggio è maggiore di quello del banco e non superiore a 21.

Le carte francesi sono 52 in un singolo mazzo e sono composte da quattro semi:

- Cuori.
- Quadri.
- Picche.
- Fiori.

Ogni seme contiene 10 carte numeriche e 3 figure: Fante, Regina e Re. Totalizzando 13 carte per ogni seme.

Nel Blackjack viene usato un minimo di 4 mazzi mischiati insieme. Nel gioco l'asso può avere valenza di 1 o 11, le figure valgono 10 mentre le altre carte da gioco valgono il loro valore nominale. I semi non hanno alcuna influenza o valore e la somma dei punti al fine del calcolo del punteggio avviene per semplice calcolo aritmetico.

Il gioco inizia con i giocatori che piazzano una puntata e di seguito il dealer distribuisce due carte scoperte per ogni giocatore, quest'ultimo, però, avrà solamente una carta scoperta mentre l'altra rimane coperta fino al suo turno (il dealer è l'ultimo a giocare).

Il giocatore una volta ottenute le due carte potrà chiedere carta o decidere di fermarsi (stare/stay) a propria discrezione. Se un giocatore supera il 21 perde e il mazziere incasserà la puntata.
Il dealer ha delle regole precise da seguire: una volta che i giocatori hanno definito i loro punteggi il dealer deve chiedere carta finché non arriva ad un punteggio maggiore o uguale a 17. Se il dealer oltrepassa il 21 "sballa" e deve pagare tutte le puntate rimanenti sul tavolo.

Una volta definiti i punteggi il dealer confronta il proprio con quello degli altri giocatori, paga le combinazioni superiori alla sua, ritira quelli inferiori e lascia quelli in parità. Il pagamento delle puntate vincenti è alla pari.

Blackjack

Il giocatore che fa 21 con le prime due carte assegnategli dal mazziere (asso e figura/dieci) forma il cosiddetto Blackjack e ha il diritto di pagamento di 3 a 2.
Se anche il dealer realizza blackjack la mano è considerata alla pari.

DoubleDown

Il giocatore ha una particolare giocata a disposizione: può raddoppiare la puntata al momento della chiamata della carta, ma impegnandosi a chiedere una sola carta. Dopo aver ricevuto questa carta il giocatore è obbligato a fermarsi. Per effettuare questa giocata è necessario che il saldo a disposizione dopo la prima puntata sia almeno uguale alla puntata stessa, in quanto il double down consiste proprio nel raddoppiare la puntata.

Split

Se nella prima distribuzione il giocatore riceve due carte dello stesso valore può effettuare lo “Split”:

- Separare le due carte formando due mani distinte e aggiungere una uguale puntata alla seconda.
- Aggiungere una carta su ciascuna mano.
- Giocare come se si avessero due mani distinte.

1.1 Requisiti

Come requisiti principali l’applicazione dovrà soddisfare i seguenti:

- Inserire il nome con il quale si vuole iniziare a giocare.
- Scelta dell’utente di giocare in modalità single player o multiplayer (bot).
- Possibilità di scegliere se iniziare a giocare oppure terminare la partita.
- Possibilità di effettuare delle puntate per ogni giocata, prelevabili da un saldo predefinito all’ingresso al tavolo.
- Possibilità di scegliere il tipo di giocata disponibile da fare.
- Possibilità dell’utente di uscire dal tavolo alla fine di ogni mano, con l’alternativa di cambiare tavolo o uscire definitivamente dal gioco.

1.2 Modello del dominio

Il BlackJack dovrà essere in grado di gestire partite in modalità single player o multiplayer, dove la prima è una partita tra dealer e utente, la seconda oltre dealer e utente comprende anche la presenza di bot, ovvero giocatori automatici in aggiunta.

Le entità fondamentali su cui si basa il gioco sono le carte caratterizzate da figura, valore e seme, esse compongono il 'Deck' formato da 4 semi di 13 carte ognuno, il quale a sua volta andrà a comporre un'altra entità chiamata 'Shoe', ovvero l'unione di più deck (minimo 4).

All'inizio di ogni partita verranno distribuite un minimo di due carte ad ogni giocatore (dealer compreso) estratte dalla shoe, le quali caratterizzano un'ulteriore entità chiamata 'Hand', in sostanza la mano dei giocatori.

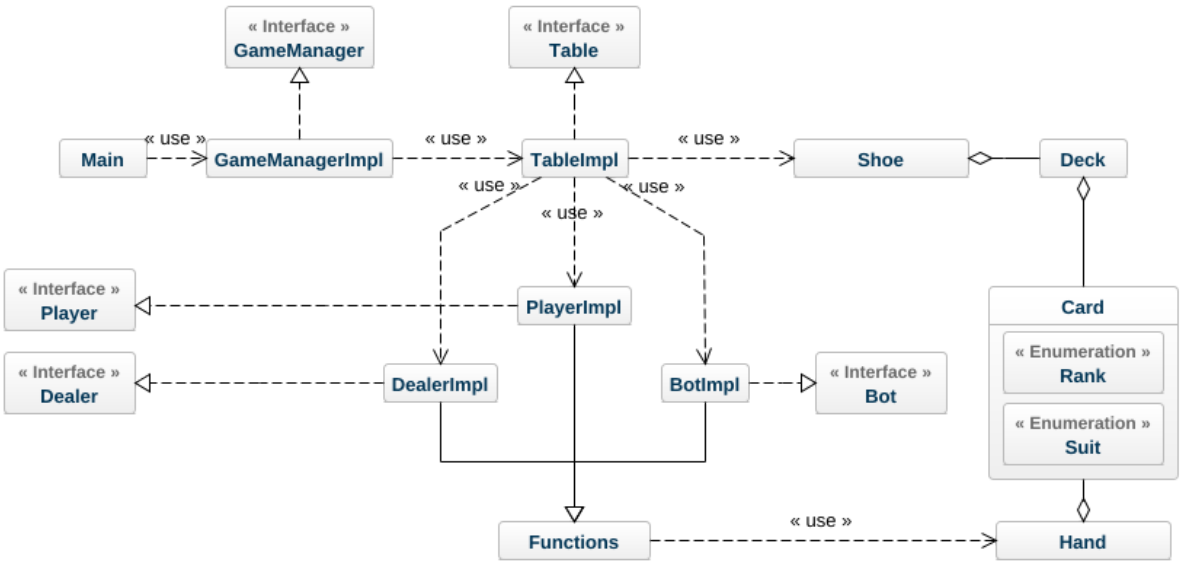
'Functions' è un'entità che raggruppa tutte le caratteristiche e le azioni comuni dei giocatori strettamente legate alla mano (Hand). Quindi questa entità è un collegamento diretto tra le entità giocatori e la 'Hand'.

Il gioco vero e proprio si svolge nel tavolo (Table) il quale conterrà le principali entità per permettere lo svolgimento della partita, ovvero i giocatori: player, dealer e bot.

Queste ultime tre entità sono composte dalla 'Hand' (diversa per ognuno) e da 'Functions' e sono caratterizzate dalle possibili giocate effettuabili da ognuno, le quali sono specificate direttamente all'interno delle loro classi.

Nel 'Table' si gestirà anche la vincita o la perdita delle puntate (bet) da parte dell'utente (player) oltre che gli opportuni controlli per la verifica del vincitore.

Infine è presente un'ulteriore classe 'GameManager' che si occupa interamente della comunicazione con l'utente, ovvero degli input e degli output che serviranno per la configurazione del gioco. Inoltre da questa entità sarà gestito anche l'intero flusso del programma e del gioco stesso.



2.1 Architettura (Design)

L’architettura dell’applicazione BlackJack è stata progettata per garantire una separazione chiara tra le diverse componenti del sistema e per facilitare la gestione delle interazioni tra di esse e la comprensione. Per questo abbiamo scelto di adottare il pattern architetturale Model-View-Controller per raggiungere questi obiettivi.

Il pattern MVC è stato utilizzato per suddividere l’applicazione in tre componenti principali: il Model, la View e il Controller. Questa suddivisione permette una gestione modulare e una facile manutenzione dell’applicazione.

Model

Il Model rappresenta la logica del programma, che determina come i dati dovrebbero essere elaborati e manipolati. E’ responsabile della gestione delle regole del gioco, della valutazione delle mani dei giocatori, del dealer e dei bot, della gestione dei turni di gioco, della gestione delle puntate ed infine del calcolo delle vincite e delle perdite. Il Model tiene traccia dello stato corrente del gioco e fornisce un’interfaccia per accedere e aggiornare le informazioni relative al tavolo da gioco. Il Model è indipendente dalla View e dal Controller, e la separazione di queste parti contribuisce ad una migliore organizzazione del codice e facilita il riuso e la manutenibilità del programma. In sostanza la parte del Model è gestita dalla nostra classe ‘Table’.

View

La View è responsabile della presentazione dell’interfaccia grafica dell’applicazione agli utenti, cioè tutto ciò che l’utente vede e con cui interagisce. Essa riceve informazioni dal Model e visualizza lo stato attuale del gioco inclusi i mazzi di carte, la mano del giocatore, del dealer, dei bot e le puntate. La View, implementata con JavaSwing nel nostro programma, è indipendente dal Model e dal Controller, in modo che possa essere sostituita o aggiornata senza influenzare il funzionamento delle altre componenti.

Controller

Il Controller funge da intermediario tra il Model e l’utente. Esso riceve input dagli utenti e traduce tali input in azioni da intraprendere sul Model. Ad esempio, quanto un giocatore vuole puntare o quando un giocatore decide di “chiamare” (hit), di “stare” (stay), di fare DoubleDown o “splittare” (split), il controller comunica con il Model per aggiornare lo stato di gioco. Il Controller può inoltre gestire anche eventi esterni, come l’inserimento del nome del giocatore per inizializzare la configurazione del gioco, la selezione della modalità di gioco (singleplayer o multiplayer), l’inserimento del numero di bot con cui si vuole giocare in caso di modalità

multiplayer, l’inizio di una nuova partita o il termine di questa, la scelta di giocare su un nuovo tavolo o di uscire definitivamente dal gioco. Una volta ricevuti gli input, il Controller interpreta ed elabora tali richieste. Questa elaborazione può comportare il recupero di dati specifici o la navigazione tra diverse parti del programma. In pratica il Controller gestisce ogni interazione con l’utente acquisendo in input ogni sua decisione, inviando messaggi di errore nel caso di inserimento errato dell’input, verificato con opportuni controlli di validazione. Il Controller nel nostro programma è identificabile dalla classe ‘GameManager’.

2.2 Design dettagliato

Nella progettazione dettagliata del programma abbiamo adottato diversi aspetti di design che contribuiscono ad una struttura modulare ben organizzata. Di seguito approfondiremo alcuni elementi chiave del design, illustrando i problemi e le soluzioni proposte.

2.2.1 Neve Luca

Gestione del Mazzo e della Distribuzione delle Carte

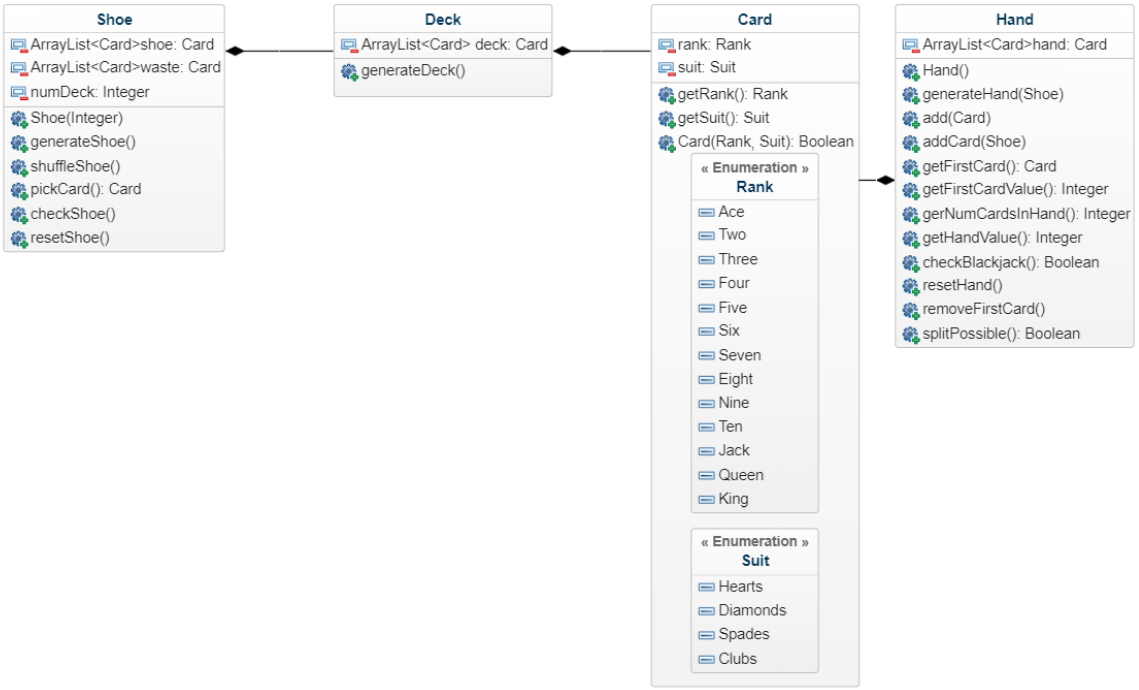
Descrizione del problema:

Uno dei problemi principali è la gestione del mazzo di carte e la sua distribuzione al player, al dealer e ai bot. E' importante garantire che il mazzo sia correttamente generato, mescolato e che le carte vengano distribuite secondo le regole del BlackJack.

Soluzione proposta:

Abbiamo definito tre classi chiave: 'Card', 'Deck', e 'Shoe'. La classe 'Card' rappresenta una singola carta con il suo seme e valore. La classe 'Deck' rappresenta un mazzo di carte e include la logica per la generazione ed il mescolamento delle carte. La classe 'Shoe' rappresenta l'insieme di un numero specificato di mazzi e fornisce funzionalità per mescolare e distribuire le carte.

Schema UML:



Utilizzo di design pattern:

In questo caso abbiamo adottato il design pattern ‘Singleton’ per la classe ‘Shoe’, poiché desideriamo una sola istanza globale di una classe contenente più mazzi. Ciò assicura che la distribuzione delle carte sia coerente in tutto il gioco.

Gestione delle Mani dei giocatori

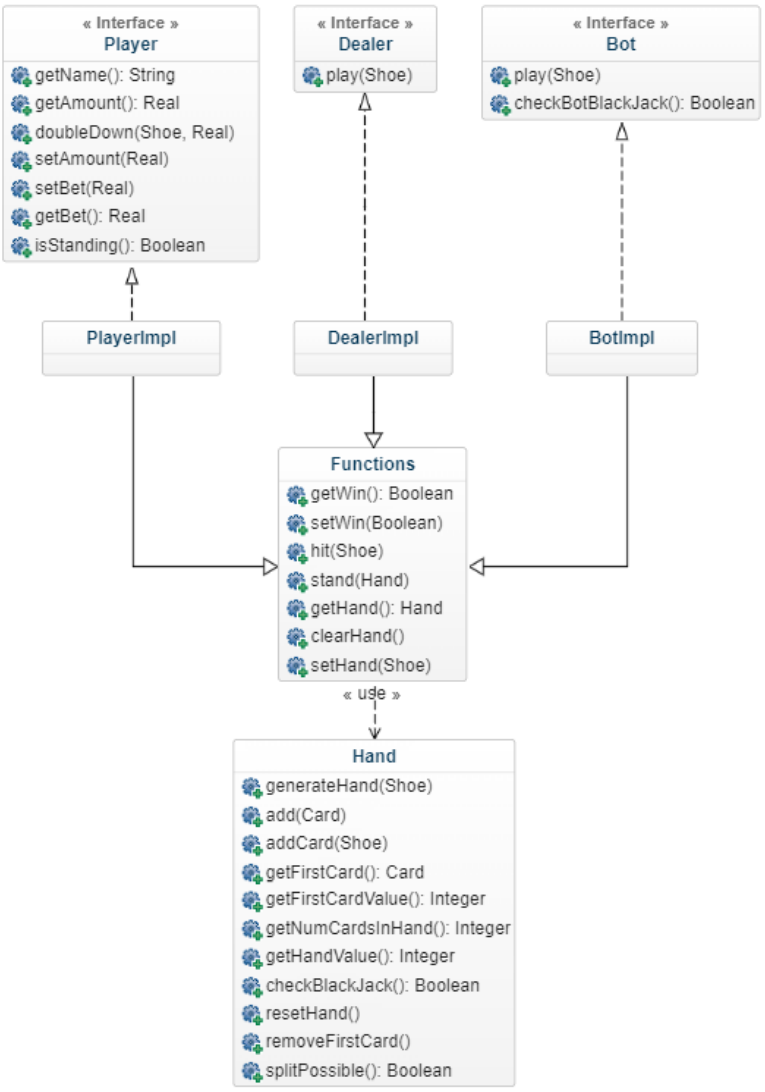
Descrizione del Problema:

Nel gioco del BlackJack è fondamentale gestire le mani dei giocatori in modo accurato. Ogni giocatore può avere una o più mani, ed ogni mano può contenere una serie di carte. E’ necessario garantire che le mani vengano create correttamente, che vengano distribuite, che il valore della mano sia calcolato in modo appropriato e che le operazioni di gioco, come ‘Hit’, ‘Stay’, ‘Double Down’ e ‘Split’, siano gestite in modo coerente.

Soluzione proposta:

Per gestire le mani dei giocatori abbiamo adottato un approccio basato su classi che utilizzano l’ereditarietà e il principio di “Single Responsibility", ovvero il principio che garantisce la progettazione di classi in modo che abbiano un’unica responsabilità chiara e ben definita. Abbiamo creato le classi ‘Hand’, ‘Player’, ‘Dealer’ e ‘Bot’, che rappresentano rispettivamente una mano di carte, un giocatore umano (utente), e due giocatori automatici (dealer e bot). Queste classi sono derivate da una classe comune ‘Functions’, che contiene funzionalità comuni tra i giocatori per la gestione delle carte e delle operazioni di gioco.

Schema UML:



Utilizzo di design pattern:

In questa parte del design possiamo vedere l'utilizzo del principio di "Single Responsibility", in cui ogni classe ha un compito ben definito (gestione delle carte, operazioni di gioco, calcolo del valore della mano). Inoltre, l'ereditarietà è utilizzata per condividere funzionalità comuni tra le diverse classi. Per di più possiamo notare l'uso di un pattern di progettazione "Strategy" implicito per il calcolo del valore della mano, dove la classe 'Hand' utilizza una strategia per calcolare il valore della mano in base al numero di assi presenti e al loro valore.

2.2.2 Benvenuti Daniele

Gestione globale dei giocatori

Descrizione del problema:

E' necessario gestire le giocate dei giocatori, inclusi i controlli delle giocate valide, le decisioni di hit o stand, e le opzioni di giocate come il double down e lo split. Inoltre per quanto riguarda il giocatore utente è necessario gestire il sistema delle puntate e del saldo del giocatore.

Soluzione proposta:

Abbiamo sviluppato le classi 'Player', 'Bot e 'Dealer' che estendono la classe 'Functions'. Ogni giocatore ha un proprio oggetto 'Hand', classe che rappresenta le carte che possiede. La classe 'Functions' fornisce funzionalità comuni ai tre tipi di giocatori, come l'aggiunta di carte ed il calcolo del punteggio della mano.

Utilizzo di design pattern:

In questo caso abbiamo implementato una forma semplificata del design pattern 'Strategy' nella gestione delle decisioni di gioco. Il metodo 'play(Shoe)' della classe 'Dealer' ed il metodo 'play(int, Shoe)' della classe 'Bot' fungono da strategia per il comportamento di gioco del dealer e dei bot. Ciò consente di separare la logica di gioco dai diversi tipi di giocatori e facilita l'aggiunta di nuovi comportamenti di gioco.

2.2.3 Comelli Pietro

Gestione del tavolo da gioco (Table)

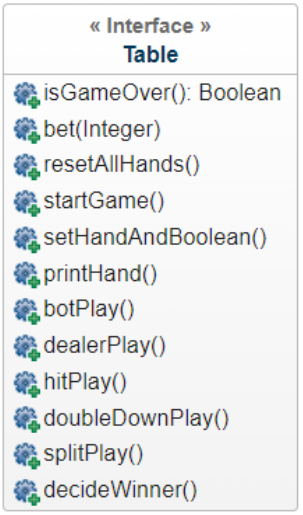
Descrizione del problema:

Il problema affrontato riguarda l'implementazione della classe 'TableImpl', che gestisce la logica del gioco BlackJack. Questa classe deve gestire i giocatori, i bot, il dealer, le puntate, le fasi del gioco e la determinazione dei vincitori. L'obiettivo è garantire che ogni fase venga gestita correttamente e in un ordine ben definito.

Soluzione proposta:

Abbiamo progettato 'TableImpl' come il nucleo centrale del gioco, ovvero come la classe che gestisce tutto lo sviluppo e il flusso di gioco. Inizialmente abbiamo valutato l'opzione di separare la gestione dei giocatori dei bot e del dealer in classi separate. Tuttavia abbiamo optato per un design più centralizzato con 'TableImpl' per migliorare la coerenza e la manutenibilità del codice.

Schema UML:



Utilizzo dei pattern:

Abbiamo scelto di gestire le diverse fasi del gioco in modo gerarchico e prevedibile utilizzando il pattern TamplateMethod. All'interno di 'TableImpl', il metodo 'startGame()' rappresenta il punto di ingresso per il gioco e definisce l'ordine sequenziale delle fasi di gioco. Abbiamo individuato le parti del gioco che potevano variare o essere personalizzate a seconda delle regole o delle situazioni, e le abbiamo incapsulate in diversi metodi come 'playerPlay' e le varie giocate che il giocatore utente può fare, 'botPlay', 'dealerPlay' e 'decideWinner'. Questo tipo di approccio semplifica l'estensione del gioco in futuro, consentendo l'aggiunta di nuove fasi o regole senza influenzare la struttura di base. L'uso del Template Method consente di separare la struttura del gioco dalla sua implementazione specifica, promuovendo una maggiore flessibilità e manutenibilità. Questo approccio favorisce anche il riuso del codice riducendo la duplicazione associata alle diverse fasi del gioco.

Gestione dell'interfaccia con l'utente

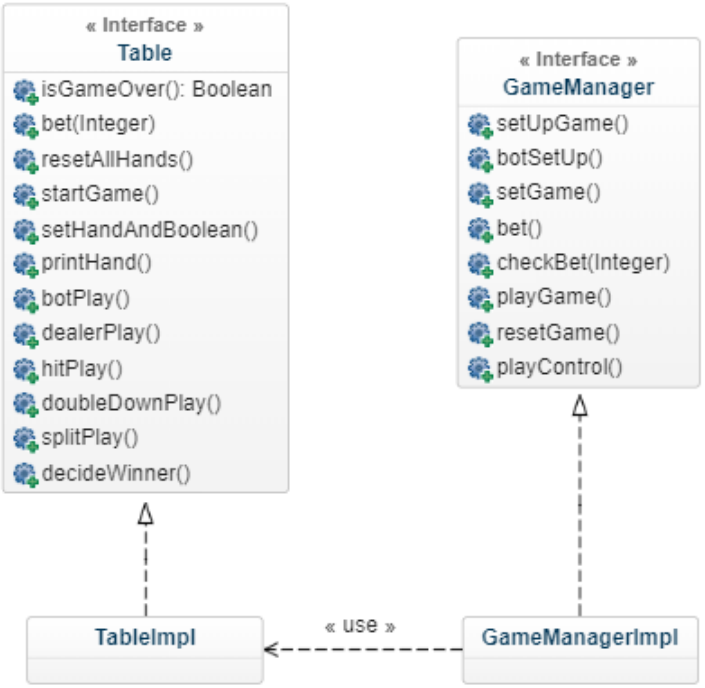
Descrizione del problema:

Il problema principale affrontato nella progettazione di 'GameManager' riguarda l'interazione fluida con l'utente, l'inizializzazione accurata del gioco ed il controllo del suo progresso (flusso).

Soluzione proposta:

Abbiamo creato 'GameManager' come un'entità esterna alla logica del gioco, ma responsabile della gestione dell'interazione con l'utente e dell'organizzazione generale del gioco. Questa classe inizializza il gioco, richiede gli input del giocatore, controlla il flusso e gestisce il reset del gioco. Abbiamo utilizzato una struttura gerarchica di metodi per guidare il flusso del gioco, inclusi metodi come 'setUpGame()', 'setGame()', 'playGame(), 'playControl()', 'botSetUp()', 'bet()', 'checkBet()' e 'resetGame(). Questi metodi consentono una gestione modulare delle diverse fasi del gioco e semplificano il processo di risoluzione dei problemi e la manutenzione futura.

Schema UML:



Utilizzo dei pattern:

In ‘GameManager’ non abbiamo applicato pattern specifici, poiché la classe si concentra principalmente sulla gestione generale del gioco e dell’interazione con l’utente. Tuttavia, l’approccio gerarchico di metodi utilizzato per il flusso di gioco e il controllo delle azioni potrebbe essere considerato come una forma semplificata del pattern Command o Chain of Responsibility, dove ciascun metodo svolge un ruolo determinante nel flusso di gioco. Questo approccio garantisce ordine e coerenza nell’evoluzione del gioco.

3.1 Testing automatizzato (Sviluppo)

Il testing automatizzato è una componente essenziale che consente di garantire che il software funzioni correttamente e di rilevare eventuali regressioni nelle funzionalità esistenti quando si apportano modifiche al codice.

Nel contesto di questo progetto, abbiamo adottato un approccio rigoroso al testing automatizzato, utilizzando la suite di test JUnit per verificare le funzionalità principali del nostro gioco di BlackJack.

Abbiamo sviluppato una serie di test automatizzati all'interno della classe 'BlackJackGameTest', che fa parte del nostro progetto. La classe inizia con alcune importazioni necessarie per JUnit e altre classi utilizzate nel test. Il metodo 'setUp()' viene eseguito prima di ogni test e viene utilizzato per inizializzare le variabili o gli oggetti necessari per l'esecuzione dei test. Questi test coprono un'ampia gamma di scenari, dalle operazioni di scommessa e gestione delle mani dei giocatori alle regole del gioco, compreso il comportamento dei bot, del dealer e del player. Di seguito sono riportati alcuni dei punti salienti dei nostri test:

- **Test delle funzionalità chiave:** Abbiamo testato le funzionalità chiave del gioco, come il calcolo del punteggio e del numero di carte nelle mani, il controllo dei limiti di scommessa e il comportamento dei giocatori durante il gioco. Questi test ci assicurano che il cuore del nostro gioco funzioni in modo affidabile.
- **Test delle regole del gioco:** Abbiamo verificato che il nostro gioco rispetti le regole del BlackJack, come il controllo del BlackJack, il bust dei giocatori (quando il loro punteggio supera il 21) e del dealer, e il conteggio dei punti delle mani.
- **Test dei bots:** Nel caso in cui siano presenti dei bot nel gioco abbiamo incluso test specifici per assicurarci che i bot seguano le regole e prendano decisioni adeguate durante il gioco, includendo test che verifichino la fine del turno dei bot.
- **Test del dealer:** Abbiamo testato il comportamento del dealer, in particolare la logica del dealer nel chiedere carta o stare in base al suo punteggio.
- **Test della scelta del vincitore:** Abbiamo verificato con specifici casi se la scelta del vincitore all'interno del nostro programma è corretta, in base al punteggio delle mani dei giocatori, testando casi di vittoria, sconfitta o pareggio del player.

La classe contiene diversi metodi di test, ognuno dei quali è annotato con'@Test', indicando che è un metodo di test. Ecco cosa fa ciascun metodo di test:

- **testShoe():** Verifica il numero di carte presenti nella 'Shoe' dopo che è stata creata.
- **testBet():** Verifica se la scommessa 'bet' è gestita correttamente per un tavolo 'table' e se l'ammontare del giocatore cambia come previsto.
- **testCheckBetLimit():** Verifica se il controllo del limite di scommessa (checkBetLimit()) funziona come previsto.

- **testHandValueCalculation():** Verifica il calcolo del valore di una mano di carte.
- **testBlackJack():** Verifica se una mano è un blackjack.
- **testPlayerHit():** Verifica il comportamento di un giocatore quando fa “hit”, ovvero se dopo la chiamata della giocata il giocatore ha più di due carte in mano.
- **testPlayerStay():** Verifica il comportamento del giocatore quando decide di stare (stay).
- **testSplit():** Verifica il comportamento della funzione “split”, ovvero che le due mani del giocatore sono inizializzate correttamente con due carte in mano ognuna.
- **testDoubleDown():** Verifica il comportamento della funzione “doubledown”, ovvero che dopo la chiamata della giocata il giocatore ha in mano una carta in più e controlla il booleano che indica la fine del turno del giocatore.
- **testBotsPlay():** Verifica il comportamento dei bot durante il gioco.
- **testDealer():** Verifica il comportamento del dealer durante il gioco.
- **testDecideWinner():** Verifica il calcolo del vincitore in base alle mani del giocatore, del dealer e dei bot.

In generale, i nostri test automatizzati forniscono una solida copertura delle funzionalità essenziali del nostro gioco. Ciò ci consente di essere sicuri che il nostro software funzioni correttamente e di identificare rapidamente qualsiasi regressione nel comportamento del gioco quando apportiamo aggiornamenti al codice.

Riteniamo che questa suite di test automatizzati sia sufficiente per garantire la stabilità delle funzionalità core del nostro gioco. Tuttavia in futuro, potremmo espandere i nostri test per coprire scenari più complessi o aggiungere funzionalità al gioco.

3.2 Metodologia di lavoro

Per la realizzazione di questo programma c'è stata una suddivisione del lavoro ben definita, abbiamo stabilito a monte i compiti che ogni membro del gruppo sarebbe dovuto andare a sviluppare.

3.2.1 Neve Luca

Creazione e gestione delle carte da gioco, gestione dell'oggetto shoe e delle mani dei giocatori.

Per la realizzazione della shoe è stato deciso di adottare un approccio paragonabile ad una matrioska.

Classe 'Card'

Per prima cosa sono state create le singole carte, composte da numero seme e valore. La classe 'Card' contiene al suo interno due tipi di dato *Enum* per identificare il seme e il valore della carta.

Classe 'Deck'

Tra l'oggetto shoe e l'oggetto card è stato inserito l'oggetto deck che, anche se non strettamente necessario ai fini della realizzazione del programma, fa in modo che il codice risulti più chiaro e lineare.

In questo modo quando viene creato l'oggetto shoe basta solo specificare il numero di deck da cui è composto in modo tale da aggiungere questi ultimi nella shoe invece delle singole carte.

Classe 'Shoe'

Il contenitore più grande di questa matrioska è l'oggetto shoe composta da un numero *n* di deck (di default 4), questa entità viene generata con due *for* annidati tra loro, il più esterno controlla il numero di deck che vengono inseriti mentre il *for* più interno controlla l'inserimento delle singole carte del deck.

Al termine di ogni mano, le carte vengono inserite in una seconda lista presente nell'entità shoe chiamata *waste*, in modo da avere sotto controllo sia le carte che devono uscire che quelle che sono già uscite. Quando il numero di carte utilizzate supera il numero delle carte ancora presenti nella shoe tutte le carte presenti nel waste vengono re-inserite nella shoe e viene tutto rimescolato in modo da rendere più difficile il famoso 'conteggio delle carte' nel BlackJack (ovvero cercare di prevedere le carte che usciranno basandosi sulla conoscenza delle carte già uscite).

Classe ‘Hand’

Dalla shoe verranno estratte le carte da consegnare alle mani dei giocatori. Per gestire la mano di ognuno viene utilizzata una lista di oggetti al cui interno sono presenti le carte, così da facilitare la rimozione, l’inserimento e la ricerca di ogni carta nella mano interessata.

La classe ‘Hand’ oltre a permettere di aggiungere e rimuovere carte, è dotata di due metodi di ‘controllo’ per stabilire se:

- La mano contiene un BlackJack
- Se ci sono le condizioni necessarie per effettuare uno split

In questa classe è presente anche un metodo che restituisce il valore presente nella mano, un ostacolo in fase di scrittura del codice è stato riuscire a dare due valori diversi agli assi (che possono valere 1 o 11). La soluzione adottata è stata aggiungere 10 in caso di presenza di un asso a patto che il valore totale non superi 21. Se dopo che l’asso è stato fatto valere 11 il giocatore chiede carta e si supererebbero i 21 allora l’asso torna al suo valore originario di 1.

Classe ‘Function’

L'obiettivo di questa classe è riuscire a raccogliere insieme le funzionalità di gioco in comuni tra le entità ‘player’, ‘dealer’ e ‘bot’. La classe ‘Function’ sarà l’unica classe che comunicherà in maniera diretta con la classe ‘Hand’.

Le principali funzionalità presenti sono: chiedere una carta, passare il turno, restituire la mano, resettare la mano e generare la mano. Oltre le funzionalità esclusivamente riguardanti le mani ci sono anche due metodi riguardo alla vittoria dell’entità chiamate setWin() e getWin().

Questa classe verrà estesa dalle classi‘player’,’dealer’ e ‘bot’ che erediteranno i metodi di quest’ultima.

Classe ‘Bot’

Oltre ad occuparmi della gestione delle carte, mi sono occupato anche dell’entità ‘bot’. Se il giocatore decide di giocare in modalità ‘multiplayer’ (giocabile solo da terminale) potrà scegliere il numero di bot automatici che giocheranno con lui (da 1 a 4).

Queste entità sono comprese di soli due metodi:

- Play: ovvero il metodo che permette loro di giocare e gestisce le decisioni che dovranno prendere.
- checkBotBlackJack: controlla se nella mano del bot sia presente un BJ.

Per far sì che non giochino con regole rigorose, come il dealer, ho cercato di implementare il metodo ‘play’ (descritto più dettagliatamente nella sezione successiva) dando un po’ di casualità nelle scelte da fare ma sempre seguendo una logica.

3.2.2 Benvenuti Daniele

Classe ‘DealerImpl()’:

La classe ‘DealerImpl()’: rappresenta il ruolo del dealer nel gioco. Questa classe è progettata per gestire le azioni del dealer come la distribuzione delle carte e la decisione sulle sue azioni da intraprendere in base alle regole del gioco.

Le azioni intraprese dal dealer sono state implementate dal metodo ‘play()’ in questo modo: il dealer pesca le carte dal mazzo finché il valore della sua mano è inferiore a 17, se il punteggio della sua mano è superiore a 21 il dealer perde (“sballa”) e viene stampato un messaggio a schermo.

Classe ‘GUIView’:

Creazione e gestione della GUI (Graphical User Interface).

La classe ‘GUIView’ ha il compito di creare tutte le componenti dell’interfaccia grafica, che sono rispettivamente:

- WelcomePanel: Pannello di benvenuto, visualizzabile per pochi secondi.
- StartPanel: Pannello di inizio gioco, premere ‘GIOCA’ per iniziare a giocare o ‘ESCI’ per uscire dal gioco.
- BetPanel: Pannello per l’input della puntata, con una casella di testo per digitare la puntata per la giocata, e un tasto per puntare (con relativo controllo degli input).
- TurnPanel: Pannello per la visualizzazione della giocata, questa presenta due macro sezioni, una per la gestione della mano del Dealer, e l’altra per la gestione della mano del Player, con i relativi output a schermo delle varie informazioni delle mani. La visualizzazione delle carte del Dealer viene effettuata attraverso un metodo che passa al pannello delle carte del dealer la semplice immagine della carte, al contrario la gestione della mano del Player è gestita dalla classe ‘GUIHand’ e le relative carte vengono visualizzate utilizzando più metodi.
- continuePlayingPanel: Pannello per visualizzare il gameOver.

Oltre alla creazione dei vari pannelli, questa classe implementa i metodi per visualizzare tali pannelli, per abilitare/disabilitare i pulsanti e per visualizzare messaggi.

Inoltre implementa i seguenti pulsanti:

- playButton : per iniziare a giocare.

- exitButton: per uscire dal gioco.
- dealButton: per iniziare la giocata.
- betField e betButton: per prendere la puntata e puntare sulla giocata.
- yesButton: per continuare a giocare.
- noButton: per uscire dal gioco.

Classe ‘GUIHand’:

La classe ‘GUIHand’ ha il compito creare (tramite il metodo ‘setupPanel’) e di gestire la mano del Player, implementado pulsanti per giocare con la propria mano, questi sono:

- hitButton: per chiedere carta.
- standButton: per stare.
- splitButton: per Splittare la mano.
- doubleDownButton: per eseguire un doubleDown.

Questa classe inoltre implementa metodi per gestire la visualizzazione dei pulsanti e la visualizzazione di messaggi.

Classe ‘TableImplGUI’:

La classe ‘TableImplGUI’ è una variante della classe ‘TableImpl’ che implementa i metodi per consentire la corretta visualizzazione e giocabilità dell’interfaccia grafica.

Come per ‘TableImpl’ questa classe è responsabile di gestire le funzionalità principali del gioco. Mentre alcuni metodi differenziano soltanto della modalità di visualizzazione dell’output, alcuni metodi sono del tutto differenti, come quelli per richiamare i metodi per la visualizzazione delle carte oppure quelli per la gestione della visualizzazione dei pulsanti.

Per motivi estetici e funzionali, non è stato possibile implementare i Bot nell’interfaccia grafica, i quali vengono invece implementati in ‘TableImpl’. Sfortunatamente per il momento sarà possibile giocare al BlackJack con interfaccia grafica solo in SinglePlayer.

Classe ‘GameManagerGUI’:

La classe ‘GameManagerGUI’ consente di giocare al BlackJack con Interfaccia Grafica. Questa classe implementa metodi che consentono all’utente un corretto flusso di gioco.

Attraverso la PlayControl vengono implementati i pulsanti e questi ultimi consentono di passare da un metodo ad un altro. Quando si clicca un Bottone o si inserisce testo, vengono sempre eseguiti dei controlli, tramite i metodi getChoice(), getStart() e getBet(), infatti, viene controllato ogni tipo di input, in modo tale che non ci siano errori di visualizzazione o bug grafici. In questa classe poi vengono mostrati tutti i pannelli implementati precedentemente dalla classe ‘GUIView’.

3.2.3 Comelli Pietro

Classe ‘PlayerImpl’:

La classe ‘PlayerImpl’ rappresenta il giocatore ed è progettata per gestire le azioni e lo stato del giocatore durante il gioco, inclusi elementi come l’importo del saldo, le scommesse (puntate), il nome del giocatore, la mano con cui iniziare a giocare e alcune delle possibili giocate da effettuare durante la partita.

I metodi principali sono:

- doubleDown(): questo metodo consente al giocatore di raddoppiare la puntata corrente. Riceve come parametri l’importo della puntata e un oggetto ‘Shoe’ che rappresenta il mazzo di carte. Il metodo raddoppia la puntata, sottrae l’importo dal saldo del giocatore e pesca una carta dal mazzo.
- setAmount(): questo metodo viene utilizzato per impostare il saldo del giocatore. Aggiunge o sottrae l’importo specificato dal saldo del giocatore.
- setBet(): setta l’importo della puntata.
- getBet(): restituisce la puntata.

Classe ‘TableImpl’:

La classe ‘TableImpl’ rappresenta il tavolo di gioco del BlackJack ed è quindi responsabile di gestire le funzionalità principali del gioco:

- Implementazione dei meccanismi per gestire i turni dei giocatori, del dealer e dei bot e stabilire il vincitore di ogni round.

- Creazione delle funzioni per gestire le puntate dei giocatori, inclusi i controlli di validazione degli input.

- Implementazione dei metodi per consentire al giocatore di prendere decisioni sulle giocate possibili da effettuare durante il suo turno, come ‘Hit’, ‘Stay’, ‘DoubleDown’ e ‘Split’. Nel caso dello split ho dovuto gestire il turno del giocatore con due mani differenti. Per fare ciò ho creato due oggetti giocatore ‘Player’ e ‘PlayerSplit’. Il primo gioca sempre ed è il giocatore ufficiale dell’utente, mentre il secondo interviene solo in caso si scelga la giocata dello split. In questo caso i due giocatori avranno due mani differenti gestite nel seguente modo: la seconda carta della mano originale del ‘Player’ verrà rimossa dalla mano di quest’ultimo ed assegnata come prima carta alla mano del ‘PlayerSplit’, di conseguenza i due giocatori pescheranno una carta a testa per finire di comporre la mano con cui iniziare a giocare. Ovviamente ho inserito anche metodi per la gestione e il controllo del turno per ogni mano del giocatore in caso di split, con l’inserimento di variabili booleane che tengono

conto del turno di ognuno.

- Implementazione del controllo del flusso del gioco, inclusi controlli per verificare lo sballamento dei giocatori ed un metodo che controlla se è avvenuto BlackJack (sia nel caso di una singola mano del giocatore sia in caso split, quindi più di una mano).

Classe ‘GameManagerImpl’

La classe ‘GameManagerImpl’ è responsabile della comunicazione con l’utente ed è il gestore principale del ciclo di gioco, ovvero gestisce interamente gli input e gli output della configurazione del gioco e si occupa di far eseguire tutti i metodi e le funzioni nel giusto ordine e nel giusto momento, coordinando la classe ‘TableImpl’, la quale si occupa solamente dell’esecuzione. E’ questa la classe che dirige il flusso e l’ordine dei metodi e delle funzioni da eseguire nella classe ‘TableImpl’.
Per creare questa classe ho implementato diversi metodi che si occupano ognuno di funzionalità differenti:

- Scelta della modalità di gioco (singolo o multiplayer) da parte dell’utente.
- Inserimento del numero di bot in caso venga selezionata la modalità multiplayer.
- Possibilità dell’utente di iniziare a giocare oppure di terminare la partita.
- Inserimento della puntata da effettuare.
- Flusso di gioco, ovvero un elenco ordinato di tutte le funzioni da eseguire per il corretto svolgimento del gioco, richiamando anche diversi metodi della classe ‘TableImpl’.
- Possibilità da parte dell’utente di selezionare un nuovo tavolo oppure di uscire definitivamente dal gioco.
- Scelta delle possibili giocate da effettuare da parte del giocatore.

In tutti questi metodi sono presenti controlli per la validazione dell’input, con messaggi d’errore o eccezioni in caso di inserimento errato dell’input.

3.3 Note di Sviluppo

Neve Luca

- **Stream:** utilizzando una lista per contenere le carte nelle mani, ho ritenuto comodo utilizzare una stream per calcolare il valore totale della mano.
- **Lambda:** l'unica lambda utilizzata è risultata efficace nell'algoritmo del calcolo del valore della mano per filtrare gli assi (se presenti) all'interno di essa.
- **Algoritmo per la giocata dei bot:** Per gestire la giocata dei bot e non farli prendere decisioni in maniera fissa (come il dealer) ho optato per un algoritmo che aumenti la casualità delle giocate dei bot mantenendo sempre una certa discrezione nel decidere l'azione da compiere: finchè il bot ha un punteggio in mano minore di 11 chiederà sempre carta, se il suo punteggio è compreso tra 12 e 15 verrà scelto in maniera casuale se chiedere carta o se stare(tramite la generazione casuale di un numero randomico tra 0 e 1 importando la libreria java.util.Random).

Benvenuti Daniele

- **Algoritmo per il controllo della scelta:** Per garantire un corretto funzionamento della GUI, soprattutto per quanto riguarda il controllo degli input dei pulsanti, ho implementato un metodo in 'GameManagerGUI' chiamato getChoice(), che testa in attesa di input finché un tasto non viene premuto. Quando questo accade esce dal ciclo e assegna alla variabile choice il messaggio ricevuto, che può essere != null in caso di tasta d'azione, e = int come nel caso della getBet().

Comelli Pietro

- **Algoritmo particolare:** ho creato un particolare algoritmo per quanto riguarda la giocata dello split. Vengono creati due giocatori separati: Player e PlayerSplit. Sono partito dalla mano del Player separando le due carte nella sua mano, mantenendo la prima ed assegnando la seconda come prima carta della nuova mano del PlayerSplit, entrambi i giocatori richiederanno poi una carta per completare la mano per iniziare a giocare. A questo punto ci sarà un algoritmo che gestisce in maniera adeguata il turno dei due giocatori tenendo conto che entrambi potranno continuare a fare nuove giocate.

4 Conclusioni

In conclusione, il nostro progetto di sviluppo del gioco del Blackjack ha richiesto un approccio rigoroso alla progettazione e all'implementazione. Abbiamo suddiviso il lavoro in modo efficiente tra i membri del team, ciascuno dei quali ha contribuito a diverse parti del programma. Il risultato è un'applicazione che simula fedelmente le regole e le dinamiche di gioco del Blackjack.

Abbiamo adottato un architettura Model-View-Controller (MVC) per garantire una separazione chiara tra le diverse componenti del sistema. Il Model gestisce la logica di gioco, la View si occupa della presentazione dell'interfaccia grafica agli utenti e il Controller funge da intermediario tra il Model e l'utente, consentendo agli utenti di interagire con il gioco.

Abbiamo utilizzato una serie di design pattern che hanno contribuito a rendere il nostro codice più modulare, flessibile e manutenibile.

Abbiamo anche sviluppato una suite di test automatizzati utilizzando JUnit per verificare le funzionalità principali del gioco. Questi test ci hanno permesso di identificare e correggere rapidamente eventuali errori e di garantire la stabilità del software.