a). 1947711

b). presenza MZ

c). Junle Ye

d). decisioni di progettazione:

1). Gestione del profilo utente:

i). Nickname:

- Il nickname del giocatore viene memorizzato come una stringa nella classe Giocatore.
- È fornito un metodo setNickName(String nickName) per consentire ai giocatori di impostare un nuovo nickname.
- Quando viene cambiato il nickname, viene notificato all'observer attraverso il metodo notifyNicknameUpdated().

ii). Avatar:

- L'avatar del giocatore è rappresentato come un oggetto ImageIcon.
- Viene fornito un metodo setAvatar(ImageIcon avatar) per permettere ai giocatori di impostare un nuovo avatar.
- Quando l'avatar viene cambiato, viene notificato all'observer attraverso il metodo notifyAvatarUpdated().

iii). Partite Giocate, Vinte e Perse:

- Le informazioni sulle partite giocate, vinte e perse sono memorizzate come interi nella classe Giocatore.
- Metodi getter e setter sono forniti per accedere e modificare queste

informazioni.

- Non sono previste notifiche specifiche quando questi valori vengono modificati, ma potrebbero essere implementate se necessario.

iv). Livello:

- Il livello del giocatore è memorizzato come un intero nella classe Giocatore.
- È fornito un metodo setLivello(int livello) per consentire ai giocatori di impostare un nuovo livello.
- Quando il livello viene cambiato, viene notificato all'observer attraverso il metodo notifyLevelUp().

v). Esperienza (Exp):

- L'esperienza del giocatore è memorizzata come un intero nella classe Giocatore.
- È fornito un metodo setExp(int exp) per consentire ai giocatori di impostare la loro esperienza.
- Quando l'esperienza viene modificata, viene notificato all'observer attraverso il metodo notifyExpUp().

vi). Dimensione della Mano (handSize):

- La dimensione della mano del giocatore è memorizzata come un intero nella classe Giocatore.
- È fornito un metodo getHandSize() per ottenere la dimensione corrente della mano del giocatore.

- È fornito un metodo setHandSize(int handSize) per consentire di impostare la dimensione della mano del giocatore.

vii). Dimensione Iniziale della Mano (initialHandSize):

- La dimensione iniziale della mano del giocatore è memorizzata come una costante finale nella classe Giocatore.
- Quando si desidera reimpostare la dimensione della mano del giocatore al suo valore iniziale, viene fornito il metodo resetHand(), che restituisce la dimensione iniziale della mano.

2). Gestione di una partita completa con un giocatore umano contro 1/2/3 giocatori artificiali:
Per giocare con un giocatore umano contro 1 giovatore artificiale, ho creato una classe GameLoop2:

- **Inizializzazione del gioco**: Il costruttore della classe GameLoop2 prende in input il giocatore umano e il giocatore NPC. Viene inizializzato il layout del gioco, caricato lo sfondo e le immagini delle carte.
- **Gestione degli eventi del mouse**: La classe implementa l'interfaccia MouseListener per gestire gli eventi del mouse sui componenti del gioco, come le carte e le etichette dei mazzi. Quando un giocatore preme su una carta o su una delle etichette dei mazzi, vengono eseguite le azioni corrispondenti.
- **Avvio del gioco**: Il metodo gameStart() gestisce l'estrazione delle carte dal mazzo e l'assegnazione delle carte ai giocatori. Durante il turno del giocatore umano, è possibile selezionare una carta dalla propria mano e giocarla. Il turno passa quindi al giocatore NPC, che esegue la sua mossa automaticamente.

- **Animazione del turno del giocatore NPC**: Il metodo npcAnimation() gestisce l'animazione del turno del giocatore NPC. Durante il proprio turno, il giocatore NPC può selezionare una carta dalla propria mano e giocarla. Se possibile, il giocatore NPC utilizzerà una carta scartata dal giocatore umano, altrimenti pescherà una nuova carta.

- **Reset del gioco**: Il metodo resetGame() reimposta il gioco dopo la fine di un turno o di una partita. Vengono rimosse le carte dai mazzi e dai pannelli dei giocatori, e vengono reimpostate le variabili di stato del gioco.

- **Avvio dell'animazione iniziale**: Il metodo startAnimation() avvia l'animazione iniziale del gioco. Durante questa animazione, vengono inizializzate le mani dei giocatori, il mazzo viene mescolato e viene gestito l'inizio del gioco.

Per giocare con due o tre giocatori artificiali ho creato altre 2 classi: GameLoop3 e GameLoop4 seguendo le stesse logiche con l'aggiunto dei panel, label e nuovi giocatori artificiali (npc2 e npc3).

3). Uso appropriato di MVC, Observer Observable e di altri design pattern:

i). **MVC**: nel progetto ho creato 3 package:

- **Model**: Dentro package Model ci sono tutte le classi che rappresentano dati dal database o altre fonti di dati.
    - **Enumerazione Rank**: Rappresenta i valori delle carte in un mazzo da gioco, come ad esempio ASSO, DUE, TRE, etc. Ogni carta ha associato un valore numerico.
    - **Enumerazione Suit**: Rappresenta i semi delle carte in un mazzo da

gioco, come CUORI, QUADRI, FIORI, PICCHE.

- **Classe Carta**: Rappresenta una singola carta nel mazzo, con un valore (Rank), un seme (Suit) e un'immagine associata.
- **Classe Regole**: Gestisce le regole del gioco, includendo funzionalità come la gestione del mazzo, il disegno delle carte, il controllo del turno dei giocatori e la terminazione del gioco.
- **Classe Giocatore**: Gestisce l'audio all'interno dell'applicazione. È implementata come un singleton, il che significa che può essere istanziata una sola volta per garantire un'unica gestione dell'audio nell'intera applicazione
- **Classe AudioManager**: Rappresenta un giocatore nel gioco. Implementa l'interfaccia Subject, che consente la notifica degli observer quando lo stato del giocatore cambia.

- **View**: Si trovano classi che rappresentano elementi dell'interfaccia utente (UI) come finestre, pannelli, pulsanti, etichette.
  - **Classe AudioManagerButton**: è responsabile della rappresentazione di un pulsante per controllare lo stato dell'audio nel programma.
  - **Classe GameFrame**: Rappresenta la finestra principale del gioco.
  - **Classe GameInterface**: Rappresenta l'interfaccia grafica del gioco.
  - **Classe GameLoop2/3/4**: Rappresenta un ciclo di gioco per un gioco di carte giocato contro un personaggio non giocante (NPC). Questo ciclo di gioco fornisce una rappresentazione grafica del gioco e gestisce gli eventi del mouse per interazioni con l'interfaccia utente.
  - **Classe JTrash**: Rappresenta il punto di ingresso dell'applicazione.

Quando l'applicazione viene avviata, il metodo main crea un'istanza della classe LoginPhase, che a sua volta avvia l'applicazione.

- **Classe LoginPhase**: Rappresenta la fase di login dell'applicazione.
- **Classe Profilo**: Rappresenta l'interfaccia per la visualizzazione e la modifica del profilo del giocatore.
- **Classe MyButton**: Rappresenta un bottone personalizzato per l'applicazione.
- **Classe MyFrame**: Rappresenta un frame personalizzato per l'applicazione.

- **Controller**: Contiene le classi che gestiscono la logica dell'applicazione e coordinano le interazioni tra la vista e il modello. Queste classi spesso si occupano di elaborare gli input dell'utente, eseguire operazioni sui dati e aggiornare la vista o il modello di conseguenza.
  - **Classe Controller**: Gestisce gli esiti del gioco e aggiorna le statistiche del giocatore in base alle partite vinte o perse.
  - Interfaccia ProfileChangeListener: Definisce un metodo di callback per notificare gli ascoltatori quando avviene una modifica del profilo, come un cambiamento nel nome utente o nell'avatar.
  - **Interfaccia Observer**: Definisce i metodi che devono essere implementati da tutte le classi che vogliono essere osservatori di un soggetto
  - **Interfaccia Subject**: Definisce i metodi per gestire gli osservatori e notificare loro determinati eventi. Le classi che implementano questa interfaccia agiscono come soggetti osservati.

ii). **Observer Observable**:

- **Interfaccia Observer**: definisce i metodi che gli oggetti interessati implementeranno per ricevere gli aggiornamenti dai soggetti osservati.
- **Interfaccia Subject**: funge da Observable. I metodi setObserver e i metodi di notifica (notifyNicknameUpdated, notifyAvatarUpdated, notifyLevelUp, notifyExpUp) permettono agli oggetti interessati (osservatori) di registrarsi per ricevere aggiornamenti e di essere notificati quando avvengono cambiamenti
- **Classe Giocatore**: è la classe concreta che implementa Subject per gestire la registrazione degli osservatori e la notifica degli eventi.
- **Classe GameInterface**: è la classe concreta che implementa Observer e riceverà gli aggiornamenti dai soggetti osservati e risponderà di conseguenza.

iii). **Singleton:** Il design pattern Singleton viene utilizzato quando si desidera garantire che una classe abbia una sola istanza e fornire un punto globale di accesso a quella istanza.

- **Classe AudioManager** ha un costruttore privato per impedire la creazione di istanze della classe dall'esterno:
  - La variabile statica privata instance mantiene l'unica istanza di AudioManager.
  - Il metodo getInstance() restituisce l'unica istanza di AudioManager, creandola se non è stata ancora creata.
  - I metodi play() e stop() vengono utilizzati per controllare la riproduzione e l'arresto dell'audio.

- L'utilizzo del design pattern Singleton in questo modo garantisce che vi sia una sola istanza di AudioManager nell'applicazione e fornisce un punto centrale per gestire l'audio in tutto il gioco.

4). Adozione di Java Swing per la GUI: nel progetto viene usato **Java Swing** per implementare e gestire la GUI.

5). Utilizzo appropriato di stream:

- il metodo **areAllCardsNotCardBackIcon(List<JLabel>** labels, ImageIcon cardBackIcon) delle classe Regole utilizza lo stream per effettuare una verifica su tutte le etichette delle carte nella lista labels:

i). Viene chiamato il metodo stream() sulla lista labels, che converte la lista in uno stream.

ii). Poi, viene chiamato il metodo noneMatch, che accetta un predicato. Questo metodo restituisce true se nessun elemento dello stream soddisfa il predicato, altrimenti restituisce false.

iii). Il predicato passato al metodo noneMatch controlla se l'icona di ogni etichetta della carta è uguale all'icona del dorso della carta (cardBackIcon).

6). Riproduzione di audio sample: ho creato una classe **AudioManager** per la riproduzione di audio sample:

- I metodi play() e stop() vengono utilizzati per controllare la riproduzione e l'arresto dell'audio.
- Con l'aggiunto di un **LineListener** al **Clip** si riproduce automaticamente la musica in loop infinito.

7). Animazioni ed effetti speciali: In classe GameLoop2, GameLoop3, GameLoop4 ho aggiunto dei timer nei metodi per realizzare animazione semplice.

- **Metodo startAnimation()**: Avvia l'animazione iniziale del gioco. Durante questa animazione, vengono inizializzate le mani dei giocatori, il mazzo viene mescolato e viene gestito l'inizio del gioco. Il timer controlla la progressione dell'animazione fino a quando tutte le carte necessarie sono state distribuite ai giocatori.
- **Metodo npcAnimation()**: Gestisce l'animazione del giocatore NPC durante il proprio turno.
- **Metodo swapCard()**: Gestisce l'animazione di scambio delle carte tra la carta appena pescata e le carte del giocatore NPC durante il gioco.

f). altre note progettuali e di sviluppo