

**PROJEKT BLACKJACK**

**W JĘZYKU JAVA**

***LEŚNIEWSKI MACIEJ***

***NR.ALBUMU 27624***

1. **Wprowadzenie i Ogólna Koncepcja**

Projekt jest w pełni funkcjonalną, jednoużytkownikową symulacją gry karcianej Blackjack. Aplikacja pozwala graczowi na rywalizację z komputerowym krupierem. Zaimplementowano wszystkie podstawowe zasady gry, w tym obstawianie zakładów, dobieranie kart (hit), pasowanie (stand), a także zaawansowane opcje, takie jak podwojenie stawki (double down), rozdzielenie kart (split) i ubezpieczenie (insurance).

Głównym celem projektu było nauczenie się korzystania z biblioteki JavaFX oraz wykorzystanie wcześniej zdobytych umiejętności do zbudowania w pełni funkcjonalnej gry BlackJack.

1. **Architektura Projektu**

Aplikacja została rozdzielona na 6 klas:

* **Model (Dane i Logika)**
  + **Card , Deck** – Reprezentują podstawowe obiekty gry – jedną kartę, oraz całą talię
  + **Player, Dealer** – Model gracza oraz krupiera, przechowują stan (rękę kart oraz pieniądze) oraz samą logikę gracza oraz krupiera
  + **GameLogic** – Rdzeń aplikacji, pełni rolę kontrolera. Zarządza stanem całej gry, jej przebiegiem, weryfikacją zasad itp.
* **Widok (Interfejs użytkownika(GUI))**
  + **BlackJackGUI** – Klasa odpowiedzialna za stworzenie, stylizację i zarządzanie wszystkimi elementami graficznymi okna gry przy użyciu biblioteki JavaFX
* **Zasoby**

Aplikacja korzysta z zewnętrznych plików graficznych reprezentujących daną kartę np. A\_of\_spades.png. Wszystkie karty są w formacie .png. Rewers karty jako back.png oraz ikona aplikacji icon.png. W folderze znajduje się również plik shuffle.wav z dźwiękiem tasowania talii wykorzystywanym w programie

1. **Opis Klas**

**Card**

* **Cel:** Reprezentuje pojedynczą kartę do gry.
* **Pola:**
  + private String suit: Kolor karty (np. "spades", "hearts").
  + private String rank: Figura/wartość karty (np. "K", "9", "A").
  + private String imagePath: Ścieżka do pliku graficznego reprezentującego kartę.
* **Kluczowe Metody:**
  + public Card(String suit, String rank): Konstruktor, który inicjalizuje kartę i tworzy ścieżkę do jej obrazka.
  + public int getValue(): Zwraca numeryczną wartość karty zgodnie z zasadami Blackjacka. Walety, Damy i Króle mają wartość 10. As domyślnie ma wartość 11. Pozostałe karty mają wartość zgodną z ich rangą.
  + public String getRank(): Zwraca rangę karty.
  + public String getSuit(): Zwraca kolor karty.
  + public String toString(): Zwraca tekstową reprezentację karty (np. "A of clubs").

**Deck**

* **Cel:** Reprezentuje standardową, 52-kartową talię do gry.
* **Pola:**
  + private List<Card> cardList: Lista przechowująca wszystkie obiekty Card w talii.
  + private String[] suits, private String[] ranks: Tablice przechowujące możliwe kolory i rangi kart, używane do inicjalizacji talii.
* **Kluczowe Metody:**
  + public Deck(): Konstruktor, który tworzy pełną, 52-kartową talię poprzez iterację po wszystkich kolorach i rangach, a następnie natychmiast ją tasuje.
  + public void shuffle(): Tasuje karty w talii przy użyciu metody Collections.shuffle().
  + public Card drawCard(): Usuwa i zwraca pierwszą kartę z talii. Jeśli talia jest pusta, zwraca null.

**Player**

* **Cel:** Klasa bazowa reprezentująca uczestnika gry. Przechowuje stan ręki, pieniędzy, zakładów oraz statystyk.
* **Pola:**
  + protected List<Card> hand, protected List<Card> secondHand: Listy kart gracza. secondHand jest używana tylko po wykonaniu opcji "Split".
  + private int money, private int bet, private int insuranceBet: Pola przechowujące stan finansowy gracza.
  + private int roundsPlayed, totalWon, totalLost, blackjacks, wins: Pola do śledzenia statystyk gry.
  + Pola cachedHandValue, isHandValueValid (i ich odpowiedniki dla drugiej ręki) służą do optymalizacji – wartość ręki jest obliczana tylko raz po dodaniu nowej karty.
* **Kluczowe Metody:**
  + public void addCard(Card card), public void addCardToSecondHand(Card card): Dodają kartę do odpowiedniej ręki i unieważniają cache wartości ręki.
  + public int getHandValue(), public int getSecondHandValue(): Obliczają i zwracają wartość punktową ręki. Implementują kluczową logikę Blackjacka, gdzie Asy mogą mieć wartość 11 lub 1, aby uniknąć przekroczenia 21 punktów.
  + public boolean isBusted(): Sprawdza, czy wartość ręki przekracza 21.
  + public boolean canDoubleDown(): Sprawdza, czy gracz może podwoić stawkę (ma dokładnie dwie karty i wystarczająco pieniędzy).
  + public boolean canSplit(): Sprawdza, czy gracz może rozdzielić rękę (ma dwie karty o tej samej randze i wystarczająco pieniędzy).
  + Metody get i set do zarządzania pieniędzmi, zakładami oraz statystykami.

**GameLogic**

* **Cel:** Centralna klasa zarządzająca całą rozgrywką. Łączy ze sobą gracza, krupiera i talię, implementując zasady gry.
* **Pola:**
  + private Player player, private Dealer dealer, private Deck deck: Obiekty kluczowe dla rozgrywki.
  + private boolean dealerTurnCompleted, playingSecondHand, firstHandCompleted: Flagi kontrolujące aktualny stan i fazę rundy.
* **Kluczowe Metody:**
  + public void startGame(): Rozpoczyna nową rundę, rozdając po dwie karty graczowi i krupierowi.
  + public void prepareNextRound(): Resetuje stan rąk, zakładów i tworzy nową, potasowaną talię przed kolejną rundą.
  + public void playerHits(), public void playerDoubleDown(), public void splitHand(), public void buyInsurance(): Metody obsługujące akcje gracza, modyfikując stan gry (np. dobierając kartę, zmieniając zakład).
  + public void dealerTurn(): Inicjuje turę krupiera, wywołując jego metodę play().
  + public String getBlackjackResult(): Sprawdza, czy na początku rundy wystąpił Blackjack i zwraca odpowiedni komunikat.
  + public String calculateRoundResult(): Po zakończeniu tury krupiera, ta metoda porównuje ręce gracza i krupiera, oblicza wynik, aktualizuje stan konta gracza i generuje podsumowanie tekstowe rundy.
  + private String resolveHandOutcome(...): Prywatna metoda pomocnicza do calculateRoundResult, która rozstrzyga wynik dla pojedynczej ręki (co jest istotne przy rozdzieleniu).

**BlackJackGUI**

* **Cel:** Tworzenie i zarządzanie graficznym interfejsem użytkownika (GUI) przy użyciu JavaFX.
* **Struktura UI:**
  + **BorderPane root**: Główny kontener okna.
  + **Górny panel (VBox topPanel)**: Wyświetla karty i wynik krupiera.
  + **Środkowy panel (VBox centerPanel)**: Wyświetla karty (jedną lub dwie ręce) i wynik gracza.
  + **Dolny panel (VBox bottomPanel)**: Zawiera panel zakładów oraz panel przycisków akcji.
* **Kluczowe Metody:**
  + public void start(Stage primaryStage): Główna metoda klasy Application, która inicjalizuje całe UI, tworzy kontrolki, ustawia ich style i przypisuje im obsługę zdarzeń.
  + private void updateUI(): Kluczowa metoda odświeżająca cały interfejs, aby odzwierciedlał aktualny stan GameLogic. Aktualizuje etykiety, wyświetla odpowiednie karty i zarządza widocznością oraz stanem (aktywny/nieaktywny) przycisków.
  + private void animateDealerTurnAndEndRound(): Realizuje animację odwrócenia zakrytej karty krupiera, a następnie sekwencyjnie uruchamia logikę jego tury i kończy rundę. Zapewnia płynne przejście między turą gracza a rozstrzygnięciem.
  + private ImageView loadCardImage(Card card), private ImageView loadBackCardImage(): Metody wczytujące pliki graficzne kart z zasobów aplikacji.
  + private void toggleBetting(boolean inGame): Przełącza interfejs między trybem obstawiania a trybem gry, aktywując i deaktywując odpowiednie przyciski.
  + private Button createChipButton(...), private Button createGameButton(...): Metody pomocnicze tworzące stylizowane przyciski, co ułatwia utrzymanie spójnego wyglądu i redukuje powielanie kodu.
  + private void showResultAlert(...), private void showStatsWindow(): Tworzą i wyświetlają okna dialogowe z wynikiem rundy lub statystykami gry.

1. **Przebieg Rozgrywki**
2. **Start Gry**

Wygląd okna po włączeniu gry

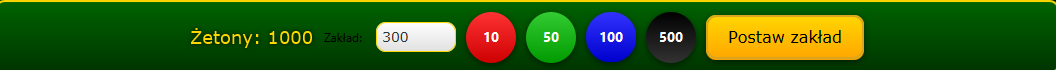
Po uruchomieniu aplikacji, gracz ma 1000 żetonów. Interfejs jest w trybie do obstawiania, guziki żetonów do obstawiania są aktywne, reszta oprócz statystyk oraz resetowania rozgrywki jest nieaktywna

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

1. **Obstawianie**

Gracz wprowadza kwotę zakład wpisując ją w polu lub klikając w odpowiednie żetony lewe kliknięcie myszki dodaje podaną liczbę a prawe odejmuje. Następnie po wybraniu kwoty gracz wciska przycisk *Postaw zakład* i gra się rozpoczyna



1. **Rozdanie**

GameLogic.startGame() jest wywoływane. Zaczyna się również prosta animacja (progres bar) tasowania kart a w tym czasie odtwarzany jest dźwięk tasowania. Gracz i krupier otrzymują po dwie karty. Jedna z kart krupiera jest zakryta.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez AI może być niepoprawna.

1. **Sprawdzenie Blackjacka**

System sprawdza, czy gracz lub krupier mają Blackjacka, Jeśli tak, runda kończy się natychmiast, a wynik jest wyświetlany, jeśli odkryta karta krupiera to as wtedy możliwe jest wcześniejsze wykupienie ubezpieczenia przez gracza.

1. **Tura Gracza**

Jeśli nie ma Blackjacka, interfejs przechodzi w tryb gry. Gracz ma do dyspozycji przyciski:

* **Dobierz kartę (Hit):** Dobiera jedną kartę.
* **Pass (Stand):** Kończy swoją turę.
* **Double Down:** Podwaja zakład, dobiera dokładnie jedną kartę i kończy turę.
* **Split:** Jeśli ma dwie takie same karty, rozdziela je na dwie osobne ręce.
* **Ubezpieczenie:** Dostępne, gdy odkryta karta krupiera to As.

Obraz zawierający tekst, zrzut ekranu, gra

Zawartość wygenerowana przez AI może być niepoprawna.

1. **Tura krupiera**

Po zakończeniu tury gracza (przez Pass, Double Down lub dobranie karty, wywoływana jest metoda animateDealerTurnAndEndRound(). Zakryta karta krupiera jest animowanie odwracana, a następnie krupier dobiera karty, aż osiągnie co najmniej 17 punktów.

Obraz zawierający tekst, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

1. **Wynik rundy**

calculateRoundResult() porównuje wyniki, aktualizuje saldo gracza i wyświetla podsumowanie w oknie dialogowym.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

1. **Nowa runda**

Interfejs wraca do trybu obstawiania, a metoda prepareNextRound() jest wywoływana w tle, aby przygotować nową talię do kolejnej gry.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Zawartość wygenerowana przez AI może być niepoprawna.

Żetony na bieżąco się aktualizują oraz pokazane okno ze statystykami rozgrywki aktualizującymi się co rundę

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

W takiej sytuacji możemy kupić ubezpieczenie od blackjacka krupiera, guzik *Ubezpieczenie* jest wtedy aktywny

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Kolejna sytuacja w której uaktywnia się przycisk split i możemy rozdzielić karty na 2 ręce

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, gra

Zawartość wygenerowana przez AI może być niepoprawna.

Wynik przeprowadzonego splita

1. **Wnioski**

Projekt z powodzeniem realizuje założenia, stworzona została prosta gra opierająca się na prawdziwym kasynowym Blackjacku, posiada graficzny interfejs użytkownika, jest stabilna, wszystkie zaplanowane aspekty zostały zrealizowane

Minusy i ewentualne rzeczy do poprawy:

* Brak mechanizmu zapisywania rozgrywki, stan konta i statystyki nie są zapisywane
* Tryb jednoosobowy – gra została zaplanowana na jednego użytkownika rywalizującego z krupierem, z powodu braku czasu nie zostało zrealizowane rozszerzenie projektu o tryb wieloosobowy np. hotseat lub tryb sieciowy
* Ewentulne rozdzielenie formatowania CSS do osobnego pliku, przez to że nie zostało to rozdzielone kod jest lekko nieczytelny, lecz z powodu małej ilości tychże poleceń nie jest to aż tak irytujące