Specyfikacja techniczna systemu do licytacji brydżowej

Wojciech Romanowski

Czerwiec 2025

Spis treści

1 Wprowadzenie

Dokument opisuje specyfikację techniczną systemu do licytacji brydżowej, który umożliwia rozgrywanie wirtualnych partii brydża z pełnym procesem licytacji i obliczania wyników.

2 Opis klas

2.1 Player

Klasa reprezentująca gracza.

- Pola:

  - `String nick` - nazwa gracza

- Metody:
  - `Player(String nick)` - konstruktor tworzący gracza o podanym pseudonimie

2.2 Team

Klasa reprezentująca drużynę (parę graczy).

- Pola:
  - `String name` - nazwa drużyny
  - `Player player1, player2` - gracze w drużynie
  - `List<Integer> scores` - lista wyników z poszczególnych rund
  - `int totalScore` - suma punktów drużyny
- Metody:
  - `Team(Player p1, Player p2, String name)` - konstruktor tworzący drużynę
  - `void addScore(int score)` - dodaje wynik do historii drużyny
  - `int average_score()` - oblicza średni wynik drużyny
  - `void displayStats()` - wyświetla statystyki drużyny

2.3 Round

Klasa reprezentująca rundę gry.

- Pola:
  - `List<Bidding> biddings` - lista odzywiek w licytacji
  - `Contract contract` - kontrakt ustalony w rundzie
  - `int nsScore, ewScore` - wyniki drużyn NS i EW
  - `int roundNumber` - numer rundy
- Metody:
  - `Round(int roundNumber)` - konstruktor tworzący rundę
  - `void addBidding(Bidding bid)` - dodaje odzywkę do licytacji

2.4 Game

Klasa zarządzająca całą grą.

- Pola:
  - `Team teamNS, teamEW` - drużyny North-South i East-West
  - `int scoreNS, scoreEW` - aktualne wyniki drużyn
  - `List<List<Bidding» biddingSequence` - sekwencje licytacji
  - `List<Round> rounds` - lista rund
  - `Table table` - układ graczy przy stole
  - `int currentRound, maxRounds` - aktualna i maksymalna liczba rund
  - `boolean gameFinished` - czy gra zakończona
  - `boolean nsVulnerable, ewVulnerable` - status "vulnerable"drużyn
  - `LocalDateTime startTime, endTime` - czas rozpoczęcia i zakończenia
- Metody:
  - `Game(Team ns, Team ew, Table table, int maxRounds)` - konstruktor
  - `void add_bidding(int round, Bidding bid)` - dodaje odzywkę
  - `void displayBidding()` - wyświetla historię licytacji
  - `boolean nextRound()` - rozpoczyna nową rundę
  - `void updateVulnerability()` - aktualizuje status vulnerable
  - `void displayScoreHistory()` - pokazuje historię wyników
  - `void displayRoundSummary()` - podsumowuje rundę
  - `Round getCurrentRound()` - zwraca aktualną rundę
  - `void startNewRound()` - inicjalizuje nową rundę
  - `String getGameInfo()` - zwraca podstawowe informacje o grze
  - `String getFullGameData()` - zwraca pełne dane gry do zapisu

2.5 Scoreboard

Klasa reprezentująca tablicę wyników.

- Pola:
  - `int round` - numer rundy
- Metody:
  - `void displayScores(Team ns, Team ew, int scoreNS, int scoreEW)` - wyświetla wyniki

2.6 Table

Klasa reprezentująca stół do gry.

- Pola:
  - `Player north, east, south, west` - gracze na pozycjach
- Metody:
  - `void assignPositions(Player n, Player e, Player s, Player w)` - przypisuje graczy
  - `Team getNSTeam()` - zwraca drużynę NS
  - `Team getEWTeam()` - zwraca drużynę EW

2.7 Bidding

Klasa reprezentująca odzywkę w licytacji.

- Pola:
  - `Player player` - gracz składający odzywkę
  - `String bid` - treść odzywki
- Metody:
  - `Bidding(Player player, String bid)` - konstruktor
  - `String toString()` - reprezentacja tekstowa odzywki

2.8 Contract

Klasa reprezentująca kontrakt brydżowy.

- Pola:
  - `Team declaringTeam, undeclaringTeam` - drużyny
  - `int level` - poziom kontraktu
  - `String suit` - kolor
  - `int result` - wynik (nadróbki/niedóróbki)
  - `boolean doubled, redoubled` - czy kontra/rekontra
- Metody:
  - `Contract(Team d, Team u, int level, String suit, int result, boolean doubled, boolean redoubled)` - konstruktor
  - `int calculateScore(boolean isDeclarerVulnerable)` - oblicza wynik

  - `int calculateContractPoints(int level, String suit)` - oblicza punkty za kontrakt
  - `int colorValue(String color)` - wartość koloru
  - `int calculatePenalty(int undertricks, boolean doubled, boolean redoubled, boolean vulnerable)` - oblicza kary

2.9 History

Klasa zarządzająca historią gier.

- Pola:
  - `List<Game> gamesHistory` - lista rozegranych gier
  - `Scoreboard scoreboard` - tablica wyników
  - `private static final String HISTORY_FILE` - plik historii
- Metody:
  - `History()` - konstruktor
  - `void add_game(Game game)` - dodaje grę do historii
  - `void displayGamesList()` - wyświetla listę gier
  - `void displayGameDetails(int gameIndex)` - pokazuje szczegóły gry
  - `private void saveHistoryToFile()` - zapisuje historię do pliku
  - `private void loadHistoryFromFile()` - wczytuje historię z pliku

2.10 GameManager

Główna klasa zarządzająca rozgrywką.

- Pola:
  - `private List<Player> players` - lista graczy
  - `History history` - historia gier
  - `private Scanner scanner` - do odczytu wejścia
- Metody:
  - `GameManager()` - konstruktor
  - `void start()` - główna pętla programu
  - `private void playNewGame()` - rozpoczyna nową grę
  - `private void setupPlayers()` - konfiguruje graczy
  - `Game setupGame()` - przygotowuje grę
  - `void playBidding(Game game)` - przeprowadza licytację

- `private void createContract(Game game, String lastBid, boolean doubled, boolean redoubled)` - tworzy kontrakt
- `boolean isValidBid(String bid)` - sprawdza poprawność odzywki
- `boolean isHigherBid(String newBid, String lastBid)` - porównuje odzywki
- `int parseResult(String resultStr)` - parsuje wynik
- `void browseHistory()` - przegląda historię
- `void displayGameSummary(Game game)` - podsumowuje grę
- `void info()` - wyświetla informacje pomocnicze

2.11 BridgeBidding

Klasa główna programu.

- Metody:
  - `void main(String[] args)` - punkt wejścia programu

3 Opis funkcjonalności

Program implementuje następujące funkcjonalności:

- Tworzenie graczy i drużyn
- Przeprowadzanie licytacji zgodnie z zasadami brydża
- Obliczanie wyników kontraktów z uwzględnieniem kontr i rekontr
- Śledzenie statusu vulnerable drużyn
- Zapisywanie i wczytywanie historii rozgrywek
- Generowanie statystyk i podsumowań