CSE 202 Hearts Card Game Project Specification Winter 2025

Pooja Sounderajan, Nevasini Sasikumar, Viroopaksh Chekuri,
Akhila Yekalluri, Anusha Ravichandran

04 February, 2025

1 Introduction

For our project, we have chosen to develop an algorithmic approach to the classic trick-taking card game Hearts. Unlike AI-based strategies, our focus is on creating efficient algorithms for representing game state, selecting valid moves, and heuristically choosing the best possible move without relying on machine learning or reinforcement learning.

The objective of Hearts is to avoid acquiring penalty points by strategically playing cards in a sequence of tricks. The single player must make informed decisions based on the current state of the game, available cards, and the next set of possible moves. Our algorithm aims to efficiently model game states and implement strategic move selection based on deterministic rules and heuristics. Below are the steps visually depicted.

Game heuristics/rules:

- Heart break case
- Queen spade - 13 points penalty
- Each heart - 1 point penalty

Design Strategies:

- The one-player game and the other 3 are randomized.
- Designing for one game.

Game Snapshots:

Figure 1: Card Ranking

Figure 2: Dealing

Figure 3: Passing

Figure 4: Playing a Round

1.1 Brief Example of Play

Each round of Hearts follows a structured flow:

1. **Dealing Cards**: A standard 52-card deck is evenly distributed among four players.

2. **Trick-taking**:
    - The player with the **2 of Clubs** starts the first trick.
    - Players must follow suit if possible; otherwise, they may discard any card.
    - The highest ranked card of the lead suit wins the trick and that player leads the next trick.
    - Hearts and Queen of Spades are penalty cards that contribute to the player's score.
3. **Scoring**:
    - Hearts are worth **1 point each**.
    - The Queen of Spades is worth **13 points**.
    - The game continues until a player reaches a predefined point limit (commonly 100) and the player with the lowest score wins.

2 Game Mechanics

2.1 State Representations and Key Components

The game state in "Hearts" consists of multiple elements that track the progress and context of the game:

- **History of Cards:** A record of all cards that have been played so far.
  Data Structure: Dictionary consisting of (key, value) pairs where the key is the suit, and the value is the cards belonging to that suit that have been discarded.
  Example: $[(Hearts, 2), (Clubs, Queen), (Spades, Ace)]$
- **In-Hand Cards:** The set of cards currently held by each player.
  Data Structure: A list of tuples, where each tuple represents a card as (Suit, Value).
  Example: $[(Hearts, 3), (Spades, 7), (Diamonds, Jack)]$
- **Current Round Cards:** The cards that have been played in the ongoing trick.
  Data Structure: A list of tuples, where each tuple represents a card as (Suit, Value).
  Example: $[(Spades, 10), (Diamonds, 5), (Hearts, Jack)]$
- **Current Turn:** Indicates which player's turn it is to play.
  Example: CurrentTurn = Player 2
- **Score:** Keeps track of the current score of all players.
  Example: $[(Player1, 0), (Player2, 14), (Player3, 8), (Player4, 4)]$

2.2 Input and Output

Input: The system receives a structured representation of the state of the game. The primary input is a list of tuples, where each tuple represents a card as (Suit, Value). This input helps to determine legal moves and make decisions based on the current state of the game.
Example: $[(Hearts, 4), (Spades, 8)]$

Output: The system produces an updated score or penalty for the player after each move. The objective is to minimize the player's score, as Hearts is a game where players seek to avoid penalty points.
Example: $Score : 14$

2.3 Core Functions

- **Legal_Moves_Determination():** Identifies which moves are valid based on game rules, including suit-following constraints and restrictions on playing penalty cards early in the game.
- **Move_Selection():** Implements a greedy algorithm to select the best move, prioritizing strategies that minimize the risk of collecting penalty points.

- **Update_Score():** Adjusts player scores after each trick based on the cards collected, adding penalty points for hearts and the Queen of Spades.
- **Update_Game_State():** Maintains and updates the state of the game after each move, ensuring the history of played cards, in-hand cards, and current round progress is accurately tracked.
- **Player_Selection():** Determines whose turn it is to play next based on the game's turn-order rules.

3 Algorithmic Approach

Our project will focus on building an algorithm that efficiently handles the following aspects:

- **Game State Representation**: A compact and efficient way to track card distribution, trick history, and remaining cards.
- **Valid Move Selection**: An algorithm to determine legal moves based on game rules.
- **Heuristic-Based Strategy**: A deterministic approach to choosing moves based on a set of predefined heuristics rather than AI-based learning.

4 Game State Representation

The game state will be modeled mathematically as follows:

Let:

- $P = P_1, P_2, P_3, P_4$ represent the four players.
- H_i be the hand of player P_i, where $H_i \subseteq D$ (deck D of 52 cards).
- T_k be the set of cards played in the k-th trick.
- L_k be the lead suit in trick k.
- W_k be the winner of trick k.
- S_i be the current penalty score of player P_i.

State Representation:

- **Hands**: $H = (H_1, H_2, H_3, H_4)$.
- **Trick History**: $T = (T_1, T_2, ..., T_n)$.
- **Lead Suit**: $L = (L_1, L_2, ..., L_n)$.
- **Scores**: $S = (S_1, S_2, S_3, S_4)$.
- **Turn Order**: A function $f : P \to P$ determining the next player.

5 Move Selection Algorithm

To ensure valid and efficient move selection, we will implement the following:

1. **Check for Lead Suit Constraints**: Ensure the player follows suit if possible.
2. **Determine Legal Moves**: Consider restrictions on playing penalty cards.
3. **Select Move Using Heuristics**:
    - **Avoid High-Risk Cards**: Avoid playing high penalty cards unless necessary.
    - **Safe Card Play**: Play lower-ranked cards to minimize penalty risk.

6 Constraints

In our approach, we impose the following constraints:

- Each player's score, denoted as S_i, satisfies $0 \leq S_i \leq 26$.
- The sum of all player scores must be exactly 26, i.e.,

These constraints ensure that the total penalty points distributed among players remain balanced as per the game's rules.

7 Objective

The objective of our algorithm is to minimize the score of the controlled player (our agent) for a single iteration of the game. Mathematically, we aim to minimize:

where S_{we} represents the penalty score of our player. The optimization is applied within the constraints of the game mechanics and heuristic-based decision-making.

8 Limitations of Algorithms

Due to computational limitations, the following algorithmic approaches are not feasible:

- **Exhaustive Search**: Enumerating all possible game states and outcomes is infeasible due to the large branching factor and memory constraints.
- **Monte Carlo Tree Search (MCTS)**: While powerful, MCTS requires extensive rollouts that exceed reasonable computational time and storage capacity.
- **Dynamic Programming for Full Game Search**: Given the state space complexity, storing and updating values for all subproblems would be computationally prohibitive.

Instead, we will focus on efficient heuristic-based approaches that provide good approximations without excessive computational overhead.

9 Possible Algorithms

Given the constraints and objective, the following heuristic-based algorithms are considered:

- **Greedy Algorithms**: Select the move that minimizes immediate penalty while maintaining long-term flexibility.
- **Rule-Based Heuristics**: Implement specific rules such as avoiding the Queen of Spades when possible, prioritizing low-value cards early, and monitoring opponent behavior.
- **Priority-based Move Selection**: Assign priority values to different moves based on expected penalties and strategic positioning within the game.

These algorithms will be designed to ensure efficient decision-making without requiring large-scale simulations or exhaustive search trees.

10 Model Extensions

If time permits, we may extend our project to include:

- **Advanced heuristics**: Refining strategies based on observed player tendencies.
- **Dynamic Adaptation**: Adjusting strategies mid-game based on score trends.
- **Parameterized Difficulty Levels**: Varying risk tolerance for different player models.