Adam Lopes

Professor Kanemoto

CPSC-39

6 December 2024

Texas Hold’em Report

My final project is a Texas Hold’em style poker game application. The goal of this project was to make the game run and play like a proper poker game. While this code is definitely not perfect, it is functional. This project makes use of several data structures such as linked lists and hashmaps. One of the main use cases of the linked list is in the Deck class. The deck in the game is a linked list which makes use of the efficient removal time of a card from the deck. A hashmap is used for the player hands, allowing efficient O(1) access of a players hand.

This project makes use of several algorithms some of which I will detail here. Some of these algorithms were done by ChatGpt. When making these algorithms I asked prompts such as “How can I implement a betting round in this project?” The first algorithm I will showcase is determineWinner, which decides the winner of the hand.



Steps to the algorithm:

1. **Initialize Variables**:

* Set bestScore to 0 to track the highest hand score seen so far.

1. **Loop Through Each Player**:

* Iterate over all players in the playerHands map, which associates players with their hands.

1. **Evaluate Hand Score**:

* For each player, use HandEvaluator.evaluateHand to calculate the combined score of their hand and the community cards.

1. **Check if the Current Player Has the Highest Score**:

* If the player's score is **greater than** bestScore:
  + Update bestScore to the new higher score.
  + Clear the topPlayers list, removing previous players with lower scores.
  + Add the current player to topPlayers as the new leader.

1. **Check for Ties**:

* If the player's score is **equal to** bestScore:
  + Add the player to the topPlayers list, indicating a tie for the highest score.

**5. Determine the Outcome**:

* After the loop, check the size of the topPlayers list:
  + **If topPlayers.size() == 1**:
    - There is a single winner.
    - Get the winning player (topPlayers.get(0)).
    - Announce the winner’s name and their score.
    - Add the pot to the winner’s balance using winner.addWinnings(pot).
    - Reset the pot to 0.
  + **If topPlayers.size() > 1**:
    - Multiple players are tied for the best score.
    - Announce that a tie has been detected.
    - Call the BreakTie method, passing topPlayers to resolve the tie.
* The time complexity for this algorithm is O(n)

The second algorithm I will showcase is the breakTie Algorithm. This algorithm, as you can probably guess functions to break a tie if two players have the same score. For the sake of simplicity, this algorithm breaks a tie by examining the high cards in a player's hands.



### **1. Initialize Variables**

* Define a Player object winner, initially set to null, to track the player with the best high card.
* Define a Card object bestHighCard, initially set to null, to track the highest card among the tied players.

### **2. Loop Through the Tied Players**

* Iterate over each player in the tiedPlayers list:
  + Retrieve the current player’s hand from the playerHands map.
  + Use the getHighCard method to determine the player's highest card, combining their hand and the community cards.

### **3. Compare High Cards**

* Compare the current player’s high card with bestHighCard:
  + **If winner is null** (first iteration) or the current player's highCard is higher than bestHighCard:
    - Update winner to the current player.
    - Update bestHighCard to the current player's high card.

### **4. Announce the Winner**

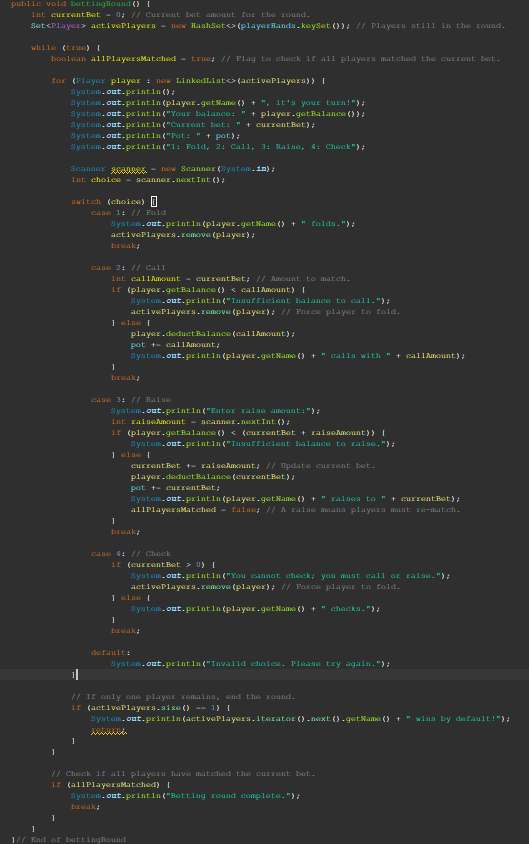
* After the loop, check if a winner was determined:
  + If winner is not null, print a message announcing the winner’s name and their high card.
  + Add the pot amount to the winner’s balance using winner.addWinnings(pot).

### **5. Reset the Pot**

* Set the pot to 0 to finalize the round.

The time complexity of this algorithm is O(n).

Finally I will showcase the bettingRound method.



1. **Initialize Variables**
   * Set currentBet to 0, representing the current bet amount for the round.
   * Create a Set named activePlayers containing all players currently in the game (playerHands.keySet()).
2. **Start Betting Loop**
   * Enter a while loop to manage the betting process. This loop will continue until all players have matched the current bet or one player remains.
3. **Initialize Match Flag**
   * Define a boolean allPlayersMatched as true. This tracks whether all players have matched the current bet.
4. **Iterate Through Active Players**
   * Use a for loop to iterate over the activePlayers set.
5. **Display Player Options**
   * For the current player:
     + Print their name, balance, the current bet, and the pot size.
     + Display action options: Fold, Call, Raise, or Check.
6. **Capture Player Choice**
   * Use a Scanner to read the player's choice.
7. **Handle Player Choice**
   * Based on the choice:

**a. Fold (1)**

* + - Remove the player from activePlayers and print that they folded.

**b. Call (2)**

* + - If the player’s balance is less than the currentBet, they are forced to fold (removed from activePlayers).
    - Otherwise, deduct the currentBet from the player’s balance, add the amount to the pot, and print that they called.

**c. Raise (3)**

* + - Prompt the player to enter a raise amount.
    - If the raise amount plus the currentBet exceeds their balance, notify them of insufficient funds.
    - Otherwise, update currentBet, deduct the total bet from the player's balance, add it to the pot, and print that they raised. Set allPlayersMatched to false because a raise requires all players to re-match.

**d. Check (4)**

* + - If the currentBet is greater than 0, notify the player that checking is not allowed and remove them from activePlayers.
    - Otherwise, allow the player to check without adding to the pot.

1. **Check for Single Player Remaining**
   * After the current player’s turn, check if activePlayers has only one player remaining:
     + If true, announce that the remaining player wins the round by default and return, ending the method.
2. **End of Player Iteration**
   * After the for loop, check if allPlayersMatched is still true:
     + If true, print a message that the betting round is complete and break out of the while loop.
3. **Repeat or Exit**
   * If allPlayersMatched is false, the while loop repeats to allow remaining players to take another turn.
4. **End Method**
   * Exit the while loop when all players match the current bet or only one player remains, signaling the end of the betting round.

There are many things I would like to change or add to this project. I would like to add basic AI to some of the players so that the game could be played as intended. I would also like to add features such as blinds, a better tiebreaker, GUI, and settings through the terminal.