Name: Syed Moiz
Class: CIS/CSC-17C
Date:12/8/24

Introduction

Title: Texas Hold'em Poker

As I built upon project 1 my reasoning for developing a poker game remained the same. Such as coding it in its most common form Texas Hold’em poker. The reason I chose this was honestly because I myself love to play poker (responsibly) with friends all the time. I also wanted to have something that I could use outside of class. I've already tested it with some of my poker buddies and they think it’s pretty neat. Aside from personal reasons I understood with a complex game like poker I could push myself to build something that used all of our requirements for the last project and this time around I made sure to include recursions, recursive sorts, hashing, trees and graphs.

The project is 1074 lines and it took me 1 week to complete the implementation, debugging, and documentation, spending 2 hours on most days. There are 6 classes 14 standalone functions and 18 member functions

Approach to development

For this project I first took a look at what I already had done from my first project and sought functions that I could improve on. Utilizing the new concepts we learned I was able to improve on community card function using recursion instead of for loops for more concise code. I also changed how I sorted players by chips, this time utilizing recursive sorting. For the other new concepts I decided to create new functions completely and I even added classes. It allowed me to realize many ways I could have improved my code from the past and taught me new ways to implement them. For help I used online forums and GeeksforGeeks to brush up on the old topics. I also rewatched old lectures and looked at examples provided to get a better understanding of the options I have to implement stuff with the new concepts. I Used visual studio to code and for version control then transferred it over to netbeans to turn in.

Game Rules

1. **Players and Cards:**

Each player is dealt two cards (known as "hole cards").

There are five community cards dealt in three stages: the Flop (3 cards), the Turn (1 card), and the River (1 card).

2. **Betting:**

Betting rounds occur before the Flop, after the Flop, after the Turn, and after the River.

Players can bet, call, raise, check, or fold during these rounds.

3. **Goal:**

Players use their hole cards and the community cards to make the best possible five-card hand.

The goal is to win chips by having the best hand or convincing other players to fold.

4. **Winning:**

The game continues until only one player remains with chips. Players can also quit the game at any point.

Code Description

1. **Card Class**:
   - Represents a single card with a suit and rank.
   - This is used to create and handle the deck of cards.
2. **Deck Class**:
   - Manages the full deck of 52 cards.
   - It can shuffle the deck and deal cards when needed.
3. **Player Class**:
   - Represents each player in the game.
   - Keeps track of their cards, chips, and actions like betting or folding.

    - Includes functions to calculate how strong a player’s hand is and track their stats.
4. **InterGraph Class**:
    - Keeps track of interactions between players, like chip exchanges.
    - Uses a graph structure to store these interactions efficiently.
5. **PlayerTree Class**:
    - Organizes players in a binary search tree based on their chip count.
    - Makes it easy to display players in order of their rankings.
6. **Game Logic**:
    - Controls the main gameplay, such as dealing cards, managing bets, and determining the winner.
    - Keeps the game moving through rounds and tracks player interactions.

Containers and Their Usage

1. **Sequences**:
    - **list**: Used to keep track of the history of actions during each betting round.
2. **Associative Containers**:
    - **set**: Stores the players who are eliminated to make sure they don’t get involved again.
    - **map**: Helps manage side pots when players go all-in with different amounts.
3. **Container Adaptors**:
    - **stack**: Used to keep track of actions like betting during the rounds.
    - **queue**: Organizes the turn order of players so everyone gets a fair chance.

Iterators

1. **Input Iterator**:
    - Goes through the list of actions in the betting history to display them.
2. **Output Iterator**:
    - Writes player statistics to a file so the game state can be saved.
3. **Forward Iterator**:
    - Moves through the array of community cards to display them during the game.
4. **Bidirectional Iterator**:
    - Used to go through the set of eliminated players if needed.

Algorithms Used

1. **Non-mutating Algorithms**:

   - **count_if**: Counts how many players are still active in the game.
   - **find**: Searches for specific actions in the betting history.
2. **Mutating Algorithms**:
   - **shuffle**: Randomizes the deck at the start of each round.
   - **remove_if**: Removes players who have no chips left.
3. **Sorting**:
   - **mergeSort**: A custom merge sort is used to rank players by their chip count in descending order.

New Concepts Applied

1. **Recursion**:
   - A recursive function is used to display community cards one at a time.
   - Recursion is also used for in-order traversal of the binary search tree in PlayerTree.
2. **Hashing**:
   - The unordered_map in the InterGraph class stores chip exchanges for quick access.
   - Player stats are saved in a hash table to make lookups fast and easy.
3. **Trees**:
   - The PlayerTree class uses a binary search tree to rank players by their chip count.
   - The tree helps organize players efficiently and supports quick traversal.
4. **Graphs**:
   - The InterGraph class models chip exchanges between players as a graph.
   - Each player is a node, and their chip transactions are represented as edges.

Sample Input/Output

```
--------------------------------------------------
        Welcome to Texas Hold'em Poker!
--------------------------------------------------
In this game, you will be playing against AI
Players in a classic poker setting. Use your
skill and a bit of luck to win chips and
become the ultimate poker champion!!!!WOOHOO

--------------------------------------------------

Basic Rules:
1. Each player is dealt two cards, known as hole cards.
2. There are five community cards dealt in three stages:
   the Flop (3 cards), the Turn (1 card), and the River (1 card).
3. Players use their hole cards and the community cards
   to make the best possible five-card hand.
4. Betting occurs before the Flop, after the Flop,
   after the Turn, and after the River.
5. You can bet, call, raise, check, fold, or even quit the game.
6. The goal is to win chips by having the best hand
   or convincing other players to fold.
--------------------------------------------------
Let's get started!
--------------------------------------------------

Do you want to load a saved game? (y/n): n
Enter the number of human players (max 6): 1
Number of bots: 5
Enter name for player 1: Syed

Player Rankings (before the game):
Syed -> Chips: 1000
Bot 1 -> Chips: 1000
Bot 2 -> Chips: 1000
Bot 3 -> Chips: 1000
Bot 4 -> Chips: 1000
Bot 5 -> Chips: 1000

New Round Begins!
Syed's hand: 2 of Spades, 4 of Diamonds
Bot 1's hand: [Hidden]
Bot 2's hand: [Hidden]
Bot 3's hand: [Hidden]
Bot 4's hand: [Hidden]
Bot 5's hand: [Hidden]

Betting Round Begins
Syed, it's your turn. Enter your action (Bet, Raise, Call, Check, Fold): Bet
Enter bet amount: 100
The current pot is: 210 chips.

Dealing the Flop...
Community cards: Queen of Clubs, Ace of Spades, 8 of Diamonds

Betting Round 2 Begins
Syed, it's your turn. Enter your action (Bet, Raise, Call, Check, Fold): Fold
The current pot is: 640 chips.

Dealing the Turn...
Community cards: Queen of Clubs, Ace of Spades, 8 of Diamonds, 10 of Clubs

Betting Round 3 Begins
The current pot is: 890 chips.

Dealing the River...
Community cards: Queen of Clubs, Ace of Spades, 8 of Diamonds, 10 of Clubs, Queen of Spades

Final Betting Round Begins
The current pot is: 1740 chips.

Showdown! Evaluating hands...
Bot 4's hand: 9 of Clubs, 4 of Clubs
Bot 4 has a hand score of 2 based on their hand and community cards.
Bot 5's hand: 8 of Spades, 2 of Diamonds
Bot 5 has a hand score of 4 based on their hand and community cards.
Bot 5 wins the pot of 1740 chips!

Would you like to continue to the next round? (y/n): n
Exiting the game...

Player Interactions:

Player Interactions:
Bot 5 interacted with:
  - Syed (Chips: 160)
  - Bot 1 (Chips: 160)
  - Bot 2 (Chips: 160)
  - Bot 4 (Chips: 160)
  - Bot 1 (Chips: 180)
  - Bot 2 (Chips: 180)
  - Bot 4 (Chips: 180)
  - Bot 2 (Chips: 200)
  - Bot 4 (Chips: 200)
  - Bot 4 (Chips: 200)
Bot 4 interacted with:
  - Syed (Chips: 160)
  - Bot 1 (Chips: 160)
  - Bot 2 (Chips: 160)
  - Bot 5 (Chips: 160)
  - Bot 1 (Chips: 180)
  - Bot 2 (Chips: 180)
  - Bot 5 (Chips: 180)
  - Bot 2 (Chips: 200)
  - Bot 5 (Chips: 200)
  - Bot 5 (Chips: 200)
Bot 2 interacted with:
  - Syed (Chips: 160)
  - Bot 1 (Chips: 160)
  - Bot 4 (Chips: 160)
  - Bot 5 (Chips: 160)
  - Bot 1 (Chips: 180)
  - Bot 4 (Chips: 180)
  - Bot 5 (Chips: 180)
  - Bot 4 (Chips: 200)
  - Bot 5 (Chips: 200)
Bot 1 interacted with:
  - Syed (Chips: 160)
  - Bot 2 (Chips: 160)
  - Bot 4 (Chips: 160)
  - Bot 5 (Chips: 160)
  - Bot 2 (Chips: 180)
  - Bot 4 (Chips: 180)
  - Bot 5 (Chips: 180)
Syed interacted with:
  - Bot 1 (Chips: 160)
  - Bot 2 (Chips: 160)
  - Bot 4 (Chips: 160)
  - Bot 5 (Chips: 160)
```

```
Updated Player Rankings (after the game):
Bot 5 -> Chips: 2090
Bot 3 -> Chips: 1000
Bot 1 -> Chips: 960
Syed -> Chips: 900
Bot 2 -> Chips: 550
Bot 4 -> Chips: 500

Player Statistics:
Bot 4 -> Games Won: 0, Chips: 500
Bot 2 -> Games Won: 0, Chips: 550
Syed -> Games Won: 0, Chips: 900
Bot 1 -> Games Won: 0, Chips: 960
Bot 3 -> Games Won: 0, Chips: 1000
Bot 5 -> Games Won: 1, Chips: 2090
```

Checkoff Sheet

Recursions

- **Recursive Function for Community Cards Display**:
  - **Location**: recursiveComcard function.
  - **Purpose**: Displays the community cards recursively by iterating through the array.

Recursive Sorting

- **Merge Sort for Sorting Players by Chips**:
  - **Location**: merge and mergeSort functions.
  - **Purpose**: Sorts players based on their chip count in descending order using recursion.

Hashing

- **Player Interaction Tracking with Hash Tables**:
  - **Location**: InterGraph class.
  - **Purpose**: Tracks interactions between players using a hash table (unordered_map)

Trees

- **Binary Search Tree for Player Rankings**:
    - **Location**: PlayerTree class.
    - **Purpose**: Organizes players by chip count in a binary search tree for efficient ranking.

Graphs

- **Graph for Tracking Chip Exchanges**:
    - **Location**: InterGraph class.
    - **Purpose**: Represents chip exchanges between players using a graph structure.

Pseudo-Code

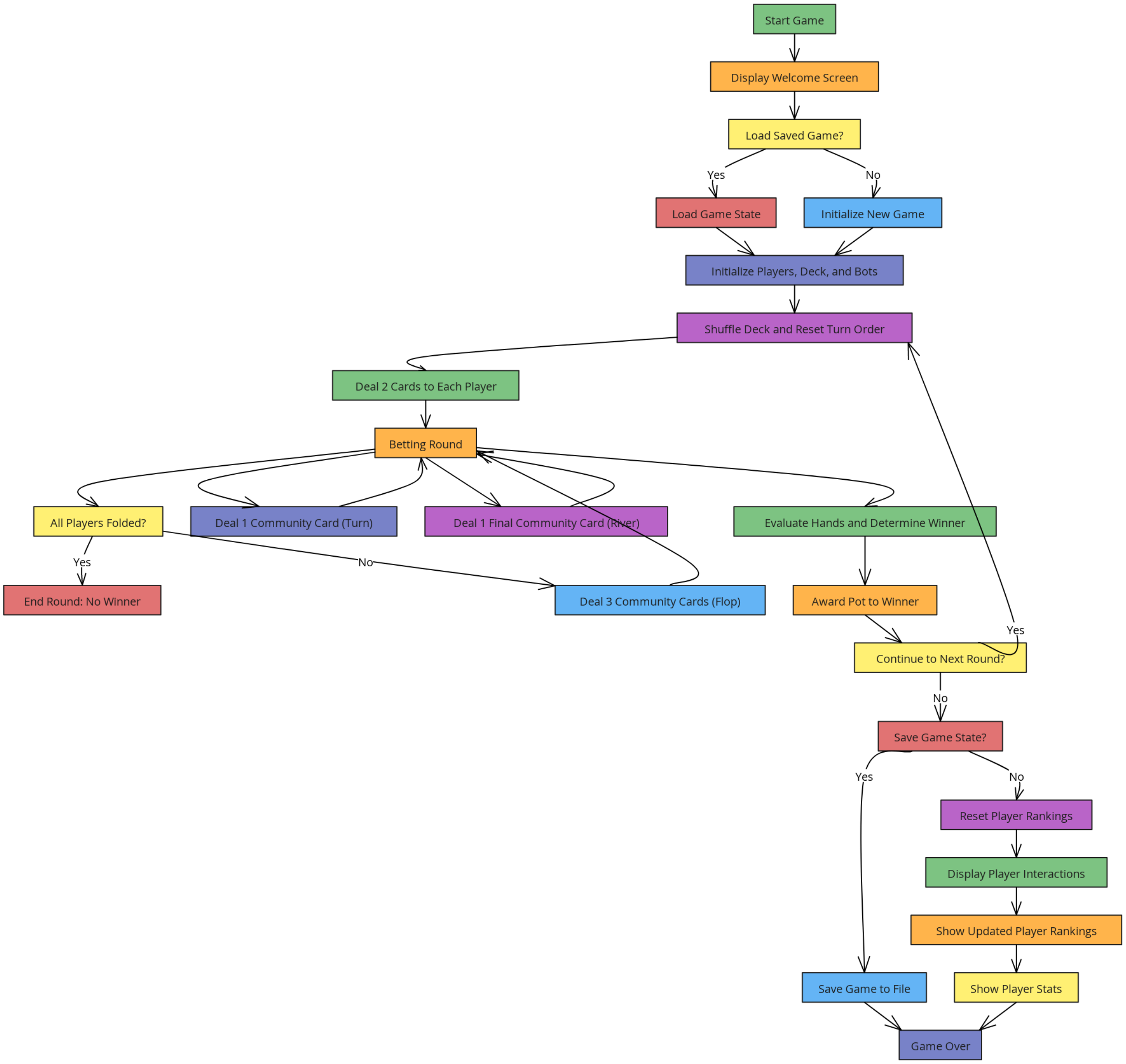
```
// Constants
DEFINE MAX_PLAYERS = 6
DEFINE MAX_CARDS = 52

// Class: Card
CLASS Card:
   suit (string)  // e.g., Hearts, Spades
   rank (string)  // e.g., Ace, King, 2
   METHOD init(suit, rank):
      SET suit and rank

// Class: Deck
CLASS Deck:
   cards[MAX_CARDS]  // Array to store all cards
   topCardIndex      // Tracks the index of the next card to deal
   METHOD init():
      CREATE 52 cards with suits and ranks
      SET topCardIndex = 0
   METHOD shuffle():
      RANDOMLY shuffle the cards
   METHOD dealCard():
      IF cards are available:
         RETURN the next card
      ELSE:
         THROW an error
```

```
    METHOD reset():
        RESET topCardIndex to 0

// Class: InterGraph
CLASS InterGraph:
    playerChipExchanges (map)  // Tracks chip exchanges between players
    METHOD addInter(player1, player2, chips):
        ADD chip exchange between player1 and player2
    METHOD display():
        PRINT all interactions between players

// Class: Player
CLASS Player:
    name, chips, folded, gamesWon, handsPlayed, handsWon
    hand[2] (to store two cards)
    METHOD init(name):
        SET initial chips, stats, and player name
    METHOD takeAction(currentBet, pot, actionHistory, communityCards,
communitySize):
        IF the player is a bot:
            DECIDE action (bet, raise, call, fold) based on logic or randomness
        ELSE:
            PROMPT the user for an action
    METHOD receiveCard(card, index):
        STORE card in hand at the given index
    METHOD showHand(hideCards):
        IF hideCards:
            PRINT hidden hand
        ELSE:
            PRINT actual cards
    METHOD evaluateHand():
        RETURN a random score for the hand
    METHOD savePlayerState(file):
        WRITE player data to a file
    METHOD loadPlayerState(file):
        READ player data from a file
    METHOD displayPlayerStatistics():
        PRINT stats like games won and chips

// Class: PlayerTree
```

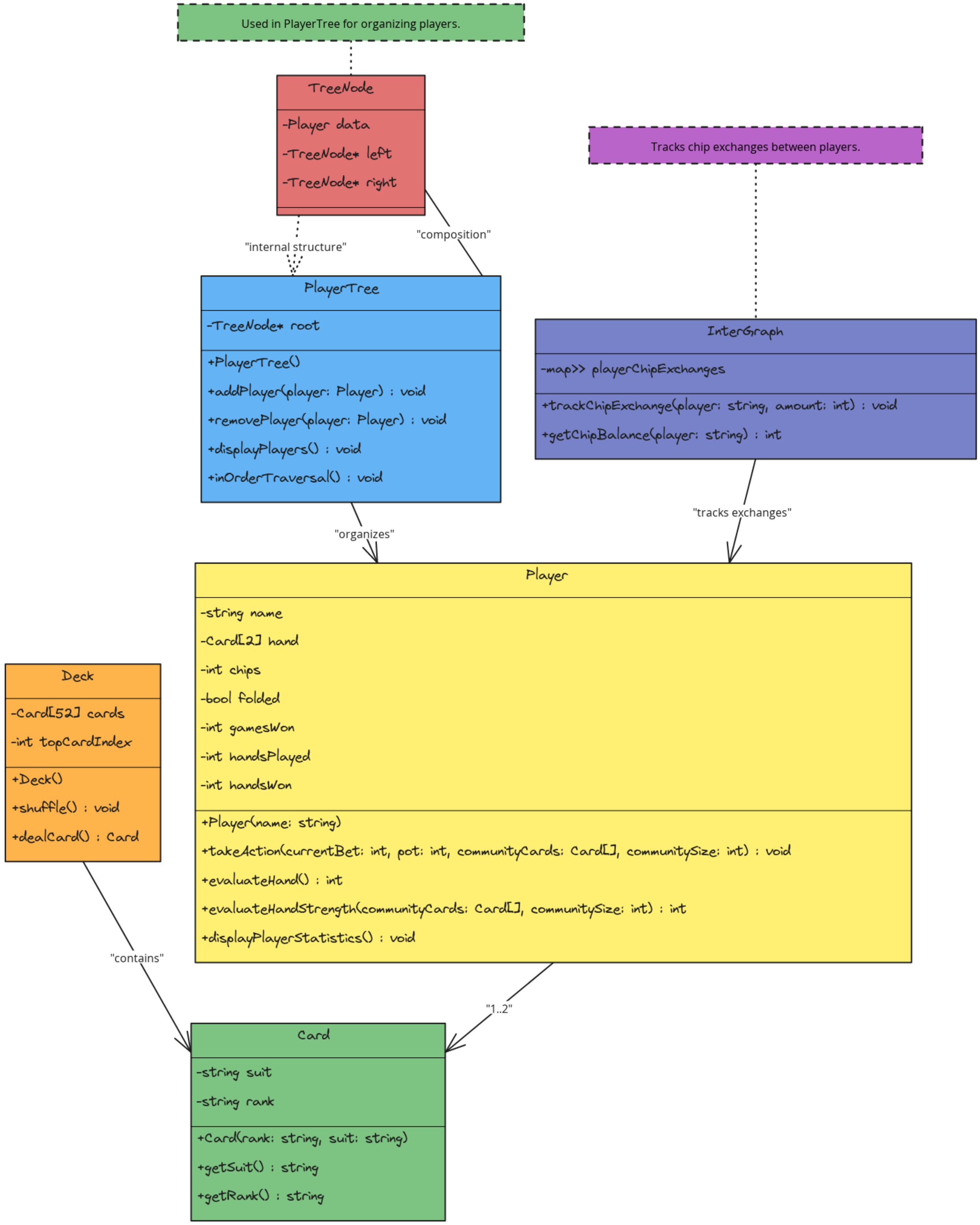
```
CLASS PlayerTree:
   root (TreeNode)
   METHOD addPlayer(player):
      INSERT the player into the tree sorted by chip count
   METHOD displayPlayers():
      PRINT all players in descending order of chips

// Function: WelcomeScreen
PRINT introduction and game rules

// Function: saveGameState(players, numPlayers)
SAVE all player stats to a file

// Function: loadGameState(players, numPlayers)
LOAD player stats from a saved file

// Function: gameLoop(players, numPlayers, deck, interactions)
START the game setup
WHILE more than one player still has chips:
   SHUFFLE the deck
   DEAL two cards to each player
   EXECUTE betting rounds
   DEAL community cards (flop, turn, river)
   IF all players fold:
      END round with no winner
   ELSE:
      DETERMINE winner based on hand strength
      UPDATE pot and player stats
      REMOVE players with zero chips
   PROMPT to continue or save the game

// Function: main
CALL WelcomeScreen()
INITIALIZE deck, players, and other variables
ASK the user if they want to load a saved game
IF yes:
   CALL loadGameState()
ELSE:
   SET UP a new game with human players and bots
CALL gameLoop()
```

```
AFTER the game:
    DISPLAY final player rankings and stats
```

Flow Chart


Start Game
Display Welcome Screen
Load Saved Game?
Yes
No
Load Game State
Initialize New Game
Initialize Players, Deck, and Bots
Shuffle Deck and Reset Turn Order
Deal 2 Cards to Each Player
Betting Round
All Players Folded?
Deal 1 Community Card (Turn)
Deal 1 Final Community Card (River)
Evaluate Hands and Determine Winner
Yes
End Round: No Winner
No
Deal 3 Community Cards (Flop)
Award Pot to Winner
Yes
Continue to Next Round?
No
Save Game State?
Yes
No
Reset Player Rankings
Display Player Interactions
Show Updated Player Rankings
Save Game to File
Show Player Stats
Game Over

UML Class Diagram


Used in PlayerTree for organizing players.
TreeNode
-Player data
-TreeNode* left
-TreeNode* right
Tracks chip exchanges between players.
"internal structure"
"composition"
PlayerTree
-TreeNode* root
+PlayerTree()
+addPlayer(player: Player) : void
+removePlayer(player: Player) : void
+displayPlayers() : void
+inOrderTraversal() : void
InterGraph
-map>> playerChipExchanges
+trackChipExchange(player: string, amount: int) : void
+getChipBalance(player: string) : int
"organizes"
"tracks exchanges"
Player
-string name
-Card[2] hand
-int chips
-bool folded
-int gamesWon
-int handsPlayed
-int handsWon
+Player(name: string)
+takeAction(currentBet: int, pot: int, communityCards: Card[], communitySize: int) : void
+evaluateHand() : int
+evaluateHandStrength(communityCards: Card[], communitySize: int) : int
+displayPlayerStatistics() : void
Deck
-Card[52] cards
-int topCardIndex
+Deck()
+shuffle() : void
+dealCard() : Card
"contains"
"1..2"
Card
-string suit
-string rank
+Card(rank: string, suit: string)
+getSuit() : string
+getRank() : string