Five Aces - Software Specification

V1.0 Release

Authors: Kent Mencel, Albert Huang, Aakarsh Pasi, Johnny Wu, Liam Fricker, and Jimmy Le

In affiliation with the UCI EECS department and the EECS 22L class

May 20, 2024

Table of Contents

Glossary

A

Algorithm - Step-by-step procedure or formula for solving a problem or accomplishing a task, within a finite number of steps. They are sets of instructions that guide computers through the processes necessary to input, manipulate, and output data; the chess bot will use a variety of these to determine the best moves to make.

Application Programming Interface (API) - A set of rules and protocols for building and interacting with software applications. They define the methods and data formats that developers can use to communicate with programming libraries, software components, etc. They abstract complex code into simpler, reusable components.

Array - Fundamental data structure in programming that consists of a collection of elements, each identified by at least one array index or key.

Artificial Intelligence - Systems that can perform tasks which typically require human intelligence with the goal to augment/automate decision making. The chess bot that the player will play against is a prime example of this, given how it will make informed decisions based on a variety of factors (board states) and algorithms.

C

Character (char) - A data type used to store individual characters, such as letters, numbers, or symbols, and typically requires a single byte of memory.

D

Data Type - A classification that specifies the type of data a variable can hold and determines the operations that can be performed on it.

Dependencies - The external libraries, modules, or packages that a software project requires to function properly. They are pieces of code written and maintained by third parties that developers incorporate into their projects to reduce redundancy and to avoid reinventing the wheel.

Doubly Linked List - A type of data structure that consists of a sequence of elements called nodes where each node contains three components: data, a pointer to the next node, and a pointer to the previous node. It is called “doubly linked” because it allows traversal in both directions. The head of the list points to the first node and the tail points to the last.

E

Enumerate (enum) - A user-defined data type that provides the developer a way to assign symbolic names to a set of related values.

F

Function - A self-contained block of code designed to perform a specific task; it can be called by various parts of the program based on the hierarchy. They have unique names and can optionally accept parameters and return a value.

G

Graphical User Interface (GUI) - A type of user interface that allows users to interact with digital devices through graphical icons and visual indicators.

I

Input - Any data or instruction that is sent to a computer system for processing.

Integer (int) - A primitive data type used to represent whole numbers that can be positive, negative or zero.

L

Linked List - A fundamental data structure that consists of a series of connected nodes, where each node contains a data value and pointer to the next node in the sequence. The first node is the head and the last node is the tail.

M

Module - A self-contained unit of code that encapsulates a specific part of a system's functionality.

Module Hierarchy - Structured organization of code into a hierarchy of modules, reflecting the functional or logical decomposition of a system into smaller, manageable parts.

O

Output - Data that a computer system produces as a result of processing inputs.

P

Packages - A bundle of software components or modules that are grouped together under a common namespace, allowing for organized and reusable code.

Pointer - Variables that store the memory addresses of other variables. Allows for dynamic memory allocation and creation of complex data structures (linked lists, etc.).

S

Server - A server is a powerful computer or system that provides resources, data, services, or programs to other computers, known as clients, over a network. It handles requests from clients, processes them, and sends back the appropriate responses, often hosting websites, databases, or applications.

Sockets - Sockets are endpoints for sending and receiving data across a network, enabling communication between devices or processes over TCP or UDP protocols. They facilitate the exchange of data by establishing a connection between a client and a server.

Source Code - Collection of written instructions and statements written by a programmer using a human-readable programming language.

Struct - A composite data type that groups together variables under one name. These variables can be of different types.

U

User Interface (UI) - The interface through which the user interacts with the program. It includes parts of how the program responds to user inputs, and all the other elements that the user can choose to interact with.

V

Void - The absence of a data type; it can be used in function definitions to specify that the function is not returning any value explicitly.

1 Software Architecture Overview.

1.1 Main data types and structures.

Card: card structure to hold card details

- Data Type: struct
- Card suits: clubs (♣), diamonds (♦), hearts (♥) and spades (♠)
    - Clubs = 1
    - Diamonds = 2
    - Hearts = 3
    - Spades = 4
- Card Number: Ace to King
    - Ace = 1
    - Two = 2
    - King = 13
- Boolean Face Card (*optional* depends on AI implementation)
    - If card is 11,12,13,1, return true
    - Allows the computer to quickly determine whether the card is a face card

Hand:

- Data type: struct
    - Note: Cards 1 and 2 are player cards, Cards 3-7 are table cards
    - Initially Cards 3-7 are NULL
    - Cards 3-5 updated after flop
    - Cards 6-7 updated one at a time after card is flipped
- Card 1
- Card 2
- Card 3
- Card 4
- Card 5
- Card 6
- Card 7
- Evaluation
    - Best Combo: best combinations of cards
    - Combo Tiebreak Card: card used to determine tie break

Combo:

- Data type: struct
- ComboType: integer representing type of combination
- ReferenceNumber: integer representing card value that determines combo value
- Example 1: two pair of 8 will have ComboType = 3 and ReferenceNumber = 8
- Example 2: full house with 4 triplet will have ComboType = 7 and ReferenceNumber = 4
- Example 3: straight with high card 9 will have ComboType =5 and ReferenceNumber = 9

ComboType:

- Data type: int
- Larger number = better combo
- High card = 1
- Pair = 2
- Two pair = 3
- Three of a kind = 4
- Straight = 5
- Flush = 6
- Full House = 7
- Four of a kind = 8
- Straight Flush = 9
- Royal Flush = 10

Player:

- Data type: struct
- Hand: information on the 7 cards + hand evaluation
    - 2 cards from the player
    - 5 cards from the table
- Bankroll: double representing money
- Bet: double representing how much player bet

Game:

- Data type: struct
- NumPlayers: integer representing the number of players
- StartingPlayer: holds who starts the current round
    - Needs to be updated after every round to the next player
- PlayerTurn: determines whose turn it is
- BigBlind: double representing money
- SmallBlind: double representing money
- Ante: double representing money
- Pot: double representing total money

PlayerTurn

- Data type: int
- Determines which player it is
- Potential Implementation
    - Player1 = 1, Player2 = 2, Player3 = 3, Player4 = 4. Etc…

GUI buttons

- All the buttons/input needed for a poker game
- Raise

        - Asks the user amount to raise by
        - Minimum & Maximum raise set by game state
            - Texas Hold'em can be played in different betting structures, including No-Limit, Pot-Limit, and Fixed-Limit. In No-Limit, there's no maximum bet, and players can go "all-in" betting all their chips. Pot-Limit restricts players to raising up to the current size of the pot, and Fixed-Limit establishes fixed bet sizes.
            - If a player goes all-in with less than the minimum bet or raise, the next players can still raise. This creates a side pot that the all-in player cannot win. Multiple side pots can be created if several players go all-in.
                - Note: if someone is all in, maximum is capped at all in value.
                - Example: player 1 is allowed to all in $200 even though current bet is $300
        - All In option
- Fold
    - Player loses bet
- Check
- Muck / Show
    - When the round ends players have 10 seconds to show or hide their hands
- Print Status
    - Allows the players to see who is winning and who is losing
        - Refer to PrintWinnings function

1.2 Major software components.

I. Main
   A. Connects all the files together
   B. Communicates to the server

II. GUI
   A. Nice looking GUI interface with a table
   B. GUI Buttons
   C. Cards & Currency
   D. (optional) animations
      1. Flipping cards
      2. Betting
      3. Emotes
      4. Chat

III. Game Setup
   A. Initialize Game
   B. Customize the Game through user input

IV. Game
   A. File containing functions to ensure Game operates

V. Hand
   A. File containing hand information and functions

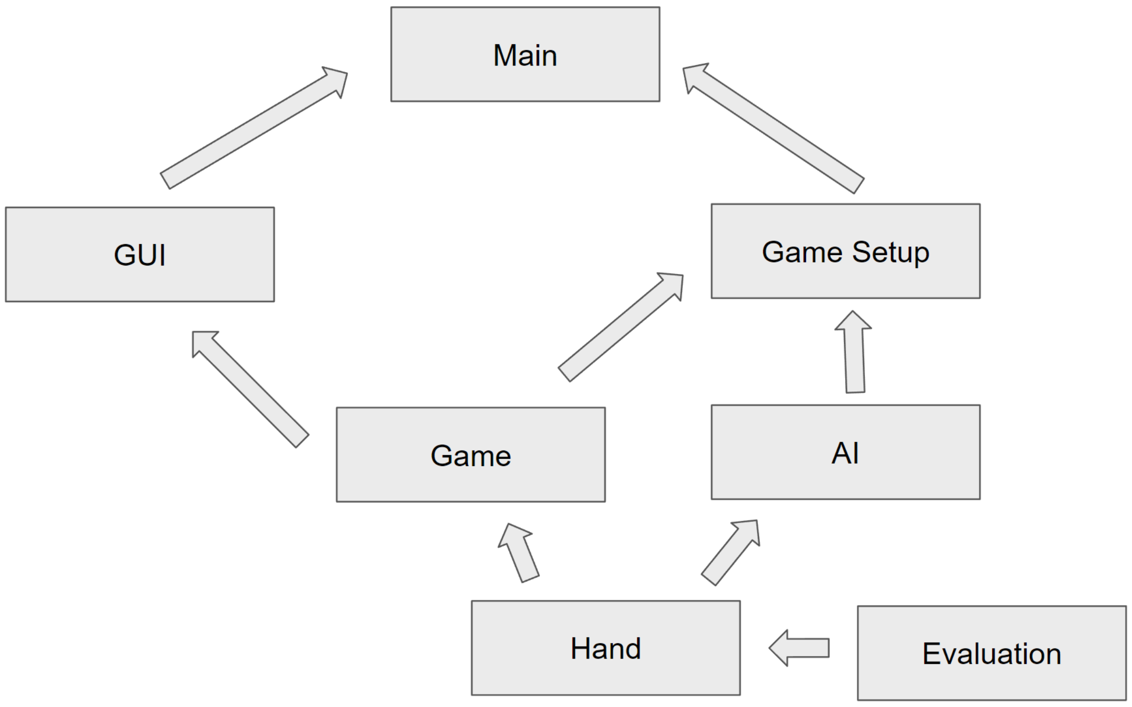
VI. Evaluation

A. Evaluation function for game state and winning chances

VII. AI algorithm

A. Algorithm for computer actions

Diagram of module hierarchy:



1.3 Module interfaces.

Main

-Module Dependencies:

- Provides: poker game
- Requires: all files

-Exported Functions:

- Main function

- Server Connection

GUI

-Module Dependencies:

- Provides: graphical interface and user input
- Requires: Game

-Exported Functions:

- Void chat(String text)
    - Arguments: String text
    - Result: displays the text on a chat interface

Game Setup

- void Initialize(int NumberPlayers, int NumberAI, double SmallBlind, double BigBlind, double AnteAmount, double TimePerTurn, double BetLimit, int NumberDecks)
    - int NumberPlayers
    - int NumberAI
    - double SmallBlind
    - double BigBlind
    - double AnteAmount
    - double TimePerTurn
    - double BetLimit
    - int NumberDecks

Game

- void StartGame(Game pokerGame, int NumberPlayers, int NumberAI)
- void DeleteGame(Game pokerGame)
- void Flop(Game pokerGame)
- void Deal()
- void NextCard()
- Void PrintWinnings(Game pokerGame)
- Void Borrow(Player PlayerID, double borrowAmount)
- Void Bet(Player PlayerID, double betAmount)
- Void Check(Game PokerGame, Player PlayerID)
- Void Raise(Game PokerGame, Player PlayerID, double RaiseAmount)
- Void Fold(Game PokerGame, Player PlayerID)
- Void ExitGame(Game PokerGame, Player PlayerID)

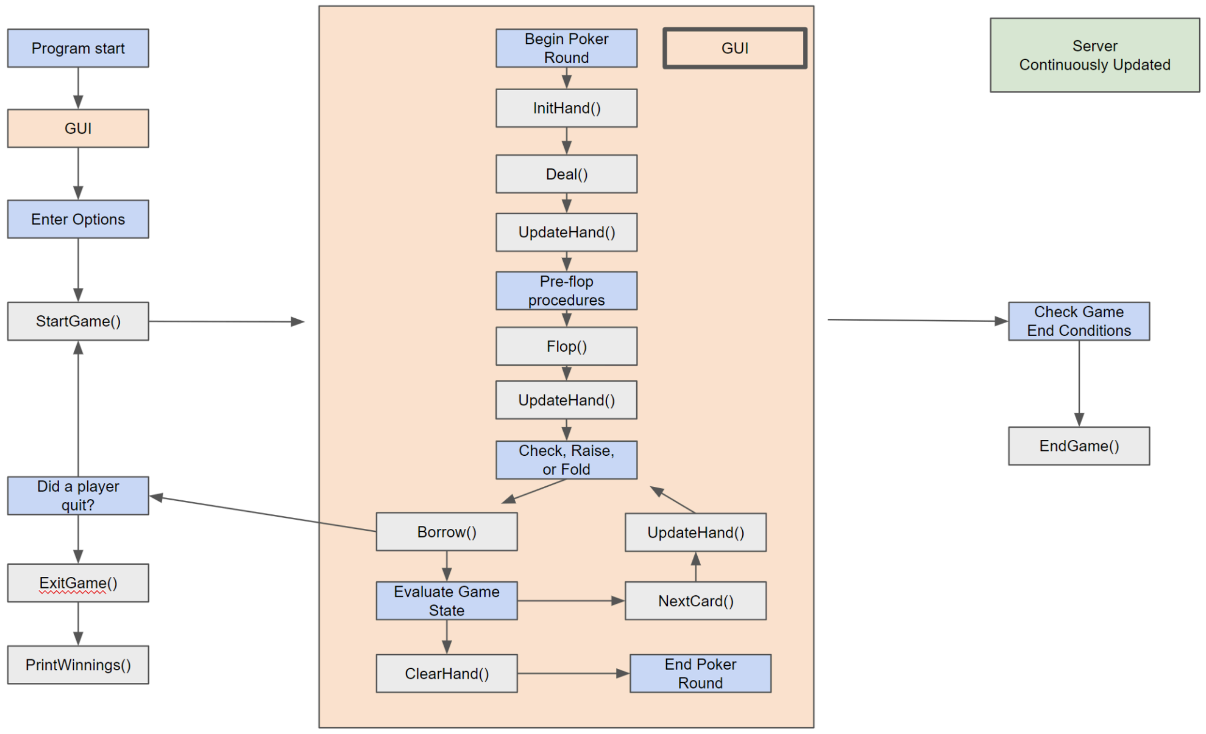
Hand

- Void InitHand()
- Void ClearHand()
- Void UpdateHand(Hand hand, Card card)

Evaluation

- void evaluate(Hand hand)

1.4 Overall program control flow.



1.5 Automated Client: Poker Bot

AI

- void MakeAIMove(Game PokerGame, Player PlayerID, int AI_IQ)
- double calculateWinningChances(Game PokerGame, Player PlayerID, int AI_IQ)

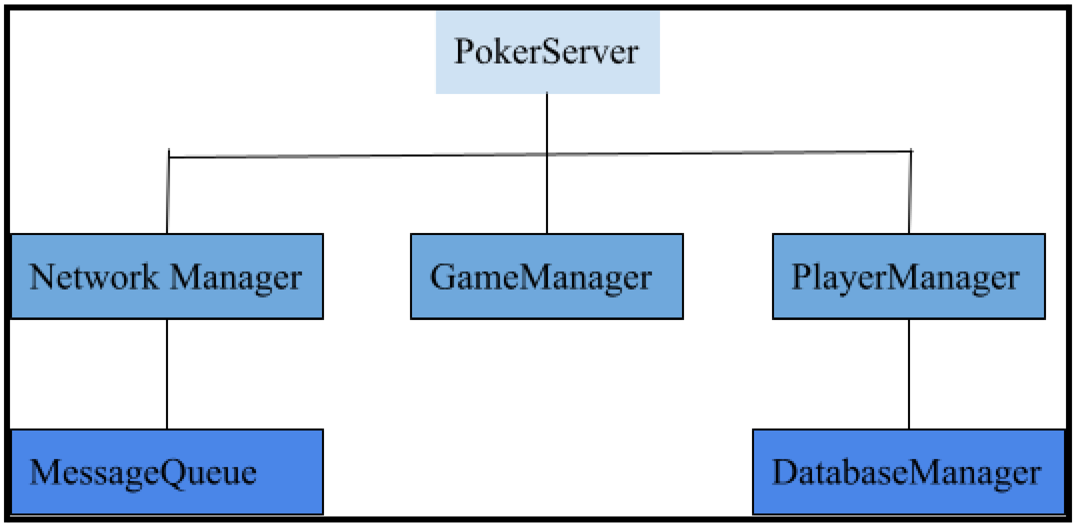
2 Poker Server Software Architecture Overview

2.1 Main data types and structures

- PlayerData: Struct to store player information (name, balance, hand, etc.)
- GameState: Struct to store the current state of the game (community cards, pot, current player, etc.)
- MessageQueue: Queue to store incoming and outgoing messages from clients.

2.2 Major software components

- NetworkManager: Handles network communication with clients (receiving/sending messages).
- GameManager: Manages the game logic and state transitions (dealing cards, betting rounds, etc.).
- PlayerManager: Manages player information and actions (joining, leaving, betting, etc.).
- DatabaseManager: Handles storage and retrieval of game data (player details, game histories, etc.).



2.3 Module interfaces

NetworkManager:

- void handleIncomingMessage(Message m);
- void sendMessageToClient(Message m, ClientID id);

GameManager:

- void startNewGame();
- void processPlayerAction(PlayerAction a);
- GameState getGameState();

PlayerManager:

- void addPlayer(PlayerData p);
- void removePlayer(PlayerID id);
- PlayerData getPlayerData(PlayerID id);

DatabaseManager:

- void saveGameState(GameState s);
- GameState loadGameState();
- void savePlayerData(PlayerData p);
- PlayerData loadPlayerData(PlayerID id);

2.4 Overall program control flow

1. Server starts and initializes NetworkManager, GameManager, PlayerManager, and DatabaseManager.
2. NetworkManager listens for incoming client connections.
3. When a new client connects, NetworkManager notifies PlayerManager to add a new player.
4. PlayerManager retrieves player data from DatabaseManager (if existing player) or creates a new entry.
5. GameManager starts a new game when enough players have joined.
6. NetworkManager receives player actions from clients and passes them to GameManager.
7. GameManager processes player actions, updates the game state, and notifies clients of the new state via NetworkManager.
8. When the game ends, GameManager saves the final game state and player data to DatabaseManager.
9. The cycle continues with new games being started as players join/leave.

3 Installation.

3.1 System requirements, compatibility.

The recommended system requirements are:

- OS: Linux (64 bit)
- Processor: Anything that can run at at least 2 GHz
- Memory: 2 GB of RAM
- Graphic Card: Integrated graphics or dedicated GPU with basic capabilities
- Compatibility: Any computer, just needs to install or access Linux somehow

3.2 Unpacking and configuration

The developer must have access to a machine running Linux OS or access to a server with Linux OS installed on it.

- The poker program is inside the '**poker_V1.0_src.tar.gz**' file that contains all the source files needed to run the program.
- They can extract the archive by running "**gtar xvzf poker_V1.0_src.tar.gz**".

3.3 Building, compilation, installation

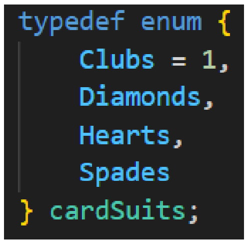
- Go into the source file directory by using "cd **poker_V1.0_src**"
- The developer can then run "**make**" and the poker executable should be in the "**bin**" directory.
- To run the poker program, the user enters the following command in the linux terminal: '**./poker_program**'. Then, all configurations of the poker game can be done via the start settings in the program.
- Alternatively, the user can also run "**make clean all test**" to compile and run the program.

4 Documentation of packages, modules, interfaces (30 points)

4.1 Detailed description of data structures

Card Suit

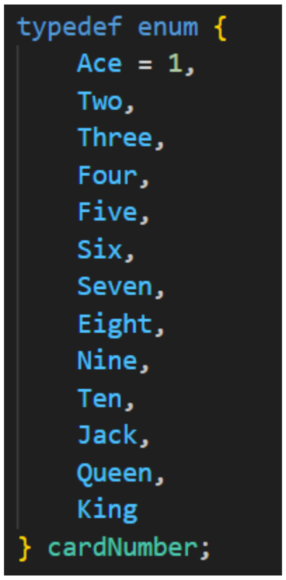
- Data Type: enum
- Contains different card suits



```
typedef enum {
    Clubs = 1,
    Diamonds,
    Hearts,
    Spades
} cardSuits;
```

Card Number

- Data Type: enum
- Contains different card numbers

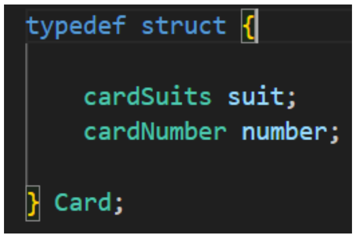


```
typedef enum {
    Ace = 1,
    Two,
    Three,
    Four,
    Five,
    Six,
    Seven,
    Eight,
    Nine,
    Ten,
    Jack,
    Queen,
    King
} cardNumber;
```

Card: card structure to hold card details

- Data Type: struct
- Card suits: clubs (♣), diamonds (♦), hearts (♥) and spades (♠)
    - Clubs = 1
    - Diamonds = 2
    - Hearts = 3
    - Spades = 4
- Card Number: Ace to King
    - Ace = 1
    - Two = 2

    - King = 13
- Boolean Face Card (*optional* depends on AI implementation)
    - If card is 11,12,13,1, return true
    - Allows the computer to quickly determine whether the card is a face card



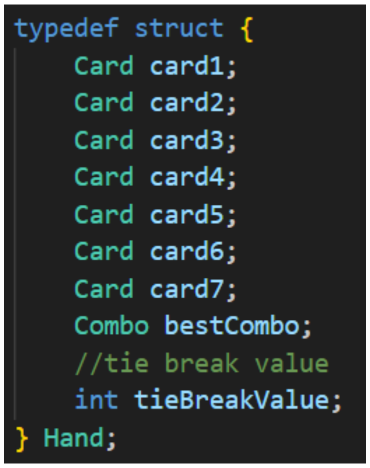
```c
typedef struct {

    cardSuits suit;
    cardNumber number;

} Card;
```

Hand:

- Data type: struct
    - Note: Cards 1 and 2 are player cards, Cards 3-7 are table cards
    - Initially Cards 3-7 are NULL
    - Cards 3-5 updated after flop
    - Cards 6-7 updated one at a time after card is flipped
- Card 1
- Card 2
- Card 3
- Card 4
- Card 5
- Card 6
- Card 7
- Evaluation
    - Best Combo: best combinations of cards
    - Combo Tiebreak Card: value of card used to determine tie break

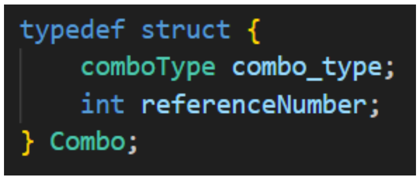


```c
typedef struct {
    Card card1;
    Card card2;
    Card card3;
    Card card4;
    Card card5;
    Card card6;
    Card card7;
    Combo bestCombo;
    //tie break value
    int tieBreakValue;
} Hand;
```

Combo:

- Data type: struct
- ComboType: integer representing type of combination
- ReferenceNumber: integer representing card value that determines combo value
- Example 1: two pair of 8 will have ComboType = 3 and ReferenceNumber = 8
- Example 2: full house with 4 triplet will have ComboType = 7 and ReferenceNumber = 4

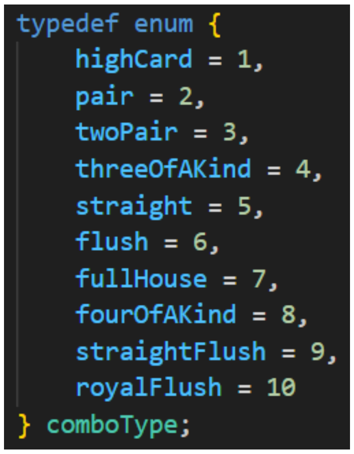
- Example 3: straight with high card 9 will have ComboType =5 and ReferenceNumber = 9



```
typedef struct {
    comboType combo_type;
    int referenceNumber;
} Combo;
```

ComboType:

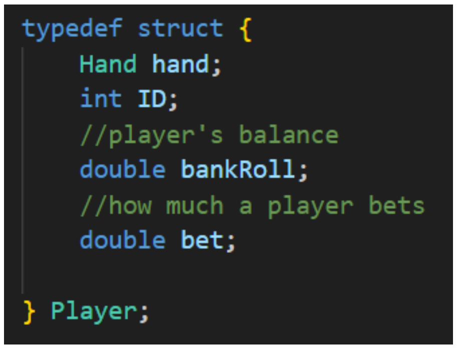
- Data type: int
- Larger number = better combo
- High card = 1
- Pair = 2
- Two pair = 3
- Three of a kind = 4
- Straight = 5
- Flush = 6
- Full House = 7
- Four of a kind = 8
- Straight Flush = 9
- Royal Flush = 10



```
typedef enum {
    highCard = 1,
    pair = 2,
    twoPair = 3,
    threeOfAKind = 4,
    straight = 5,
    flush = 6,
    fullHouse = 7,
    fourOfAKind = 8,
    straightFlush = 9,
    royalFlush = 10
} comboType;
```

Player:

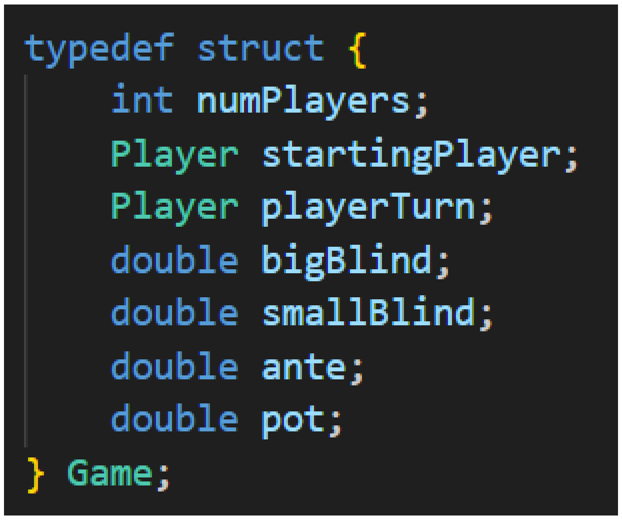
- Data type: struct
- Player ID: int starting at 1 and increasing by one for next player
- Hand: information on the 7 cards + hand evaluation
    - 2 cards from the player
    - 5 cards from the table
- Bankroll: double representing money
- Bet: double representing how much player bet


```
typedef struct {
    Hand hand;
    int ID;
    //player's balance
    double bankRoll;
    //how much a player bets
    double bet;

} Player;
```

Game:

- Data type: struct
- NumPlayers: integer representing the number of players
- StartingPlayer: holds who starts the current round
    - Needs to be updated after every round to the next player
- PlayerTurn: determines whose turn it is
- BigBlind: double representing money
- SmallBlind: double representing money
- Ante: double representing money
- Pot: double representing total money


```
typedef struct {
    int numPlayers;
    Player startingPlayer;
    Player playerTurn;
    double bigBlind;
    double smallBlind;
    double ante;
    double pot;
} Game;
```

GUI buttons

- All the buttons/input needed for a poker game
- Raise
    - Asks the user amount to raise by
    - Minimum & Maximum raise set by game state
        - Texas Hold'em can be played in different betting structures, including No-Limit, Pot-Limit, and Fixed-Limit. In No-Limit, there's no maximum bet, and players can go "all-in" betting all their chips. Pot-Limit restricts

        - players to raising up to the current size of the pot, and Fixed-Limit establishes fixed bet sizes.
        - If a player goes all-in with less than the minimum bet or raise, the next players can still raise. This creates a side pot that the all-in player cannot win. Multiple side pots can be created if several players go all-in.
            - Note: if someone is all in, maximum is capped at all in value.
            - Example: player 1 is allowed to all in $200 even though current bet is $300
    - All In option
- Fold
    - Player loses bet
- Check
- Muck / Show
    - When the round ends players have 10 seconds to show or hide their hands
- Print Status
    - Allows the players to see who is winning and who is losing
        - Refer to PrintWinnings function

4.2 Detailed description of functions and parameters

Main

-Module Dependencies:

- Provides: poker game
- Requires: all files

-Exported Functions:

- Main function
- Server Connection

GUI

-Module Dependencies:

- Provides: graphical interface and user input
- Requires: Game

-Exported Functions:

- Void chat(String text)
    - Arguments: String text
    - Result: displays the text on a chat interface

Game Setup

-Module Dependencies:

- Provides: Setup of poker game
- Requires: Game, AI

-Exported Functions:

- void Initialize(int option, int option2, int option3)
    - Arguments: **option1**, **option2**, **option3**
    - Result: initialize the game state
    - Note: option1, option2, option3, etc… are parameters that define the following
        - Number of players
        - Small blind amount
        - Big blind amount
        - Ante amount (if enabled)
        - Timer (if enabled)
        - Bet limit option
            - No limit
            - Pot limit
            - Fixed limit
        - AI (if enabled)
        - Number of decks (if enabled)
            - Default: 1 deck
            - In custom games it is possible to use more than one deck

Game

-Module Dependencies:

- Provides: a lot of functions to ensure game operation
- Requires: Hand

-Data Structures

- Game
- Player

-Exported Functions:

- void Flop(Game pokerGame)
    - Arguments: Game pokerGame
    - Result:
        - 1) displays the flop, or initial three cards
        - 2) iterate from 1 to total number of players and update Hand for all Players
- void NextCard()
    - Arguments: void
    - Result: adds another card (the 4th and 5th card on the table)
- void Deal()
    - Arguments: void
    - Result: deals two cards to each player
- void StartGame(Game pokerGame, int NumberPlayers, int NumberAI)
    - Arguments: Game pokerGame, int NumberPlayers, int NumberAI
        - NumberAI = zero by default
    - Results:
        - 1) collect big blind, small blind, and ante (if enabled)

            - If player does not have enough, ask whether player wants to borrow
            - Remove the player if the player runs of of money and does not borrow more
        - 2) deal two cards to each player
        - 3) ask for pre-flop bets
        - 4) flop
- void DeleteGame(Game pokerGame)
    - Arguments: Game pokerGame
    - Result: delete the game and associated memory and data
- Void PrintWinnings(Game pokerGame)
    - Arguments: Game pokerGame
    - Result: displays the following
        - 1) how much each player started with
        - 2) how much each player has currently
        - 3) percentage gain of each player
        - 4) number of "wins"
        - 5) number of rounds played
- Void Borrow(Player PlayerID, double borrowAmount)
    - Argument: double borrowAmount, Player PlayerID
    - Result: adds the borrowAmount to Bankroll in struct Player
- Void Bet(Player PlayerID, double betAmount)
    - Arguments: Player PlayerID, double RaiseAmount
    - Result: adds to pot & updates Player struct
- Void Check(Game PokerGame, Player PlayerID)
    - Argument: Game PokerGame, Player PlayerID
    - Result: proceeds to the next player or turn
- Void Raise(Game PokerGame, Player PlayerID, double RaiseAmount)
    - Argument: Game PokerGame, Player PlayerID, double RaiseAmount
    - Result: checks if Raise is valid and calls the Bet() function
- Void Fold(Game PokerGame, Player PlayerID)
    - Arguments: Game PokerGame, Player PlayerID
    - Result: the player folds this turn
- Void ExitGame(Game PokerGame, Player PlayerID)
    - Arguments: Game PokerGame, Player PlayerID
    - Results:
        - 1) the player exits the game
        - 2) modify Game PokerGame to reflect changes
        - 3) calls printWinnings() to show how the player did

Hand

-Module Dependencies:

- Provides: functions ensuring operation of Hand
- Requires: evaluation
    - NOTE: Hand, Card, Combo, ComboType structures are defined in Evaluation

-Exported Functions:

- Void InitHand()
    - Argument: none
    - Result: creates and initializes Hand structure
- Void ClearHand()
    - Argument: none
    - Result: resets Hand structure
- Void UpdateHand(Hand hand, Card card)
    - Argument: Card card
    - Result: adds the card to Hand hand

Evaluation

-Module Dependencies:

- Provides: combo definitions and evaluations
- Requires: nothing

-Data structures

- Struct Hand
- Struct Card
- Struct Combo
- Enum ComboType

-Exported Functions:

- void evaluate(Hand hand)
    - Argument: none
    - Result: returns best combo

AI

-Module Dependencies:

- Provides: AI decision making
- Requires: Hand

-Exported Functions:

- void MakeAIMove(Game PokerGame, Player PlayerID, int AI_IQ)
    - Arguments: Game PokerGame, Player PlayerID, int AI_IQ
    - Result: make the best AI move in the current game state according to how smart the AI should be
- double calculateWinningChances(Game PokerGame, Player PlayerID, int AI_IQ)
    - Arguments: Game PokerGame, Player PlayerID, int AI_IQ
    - Result: returns winning chance percentage

4.3 Detailed description of the communication protocol

NetworkManager:

- void handleIncomingMessage(Message m):
    - Message m: The incoming message from a client.
- bool sendMessageToClient(Message m, ClientID id):
    - Message m: The message to be sent.
    - ClientID id: The ID of the client to send the message to.
    - Returns true if the message was sent successfully, false otherwise.

GameManager:

- bool startNewGame(vector<PlayerID> playerIds):
    - vector<PlayerID> playerIds: The IDs of the players to include in the new game.
    - Returns true if the new game was started successfully, false otherwise.
- void processPlayerAction(PlayerAction a):
    - PlayerAction a: The action performed by a player (e.g., bet, fold, call).
- GameState getGameState():
    - Returns the current state of the game.

PlayerManager:

- bool addPlayer(PlayerData p):
    - PlayerData p: The data of the player to be added.
    - Returns true if the player was added successfully, false otherwise.
- bool removePlayer(PlayerID id):
    - PlayerID id: The ID of the player to be removed.
    - Returns true if the player was removed successfully, false otherwise.
- PlayerData getPlayerData(PlayerID id):
    - PlayerID id: The ID of the player whose data is requested.
    - Returns the PlayerData struct for the specified player.

DatabaseManager:

- bool saveGameState(GameState s):
    - GameState s: The game state to be saved.
    - Returns true if the game state was saved successfully, false otherwise.
- GameState loadGameState(GameID id):
    - GameID id: The ID of the game state to be loaded.
    - Returns the GameState struct for the specified game.
- bool savePlayerData(PlayerData p):
    - PlayerData p: The player data to be saved.
    - Returns true if the player data was saved successfully, false otherwise.
- PlayerData loadPlayerData(PlayerID id):
    - PlayerID id: The ID of the player whose data is to be loaded.
    - Returns the PlayerData struct for the specified player.

5 Development plan and timeline (10 points)

5.1 Partitioning of tasks

End of Week 1: Basic Blueprint and outline of idea

End of Week 2: Rudimentary Code Development of blueprint and partitioning of responsibilities

End of Week 3: Projected Basic Poker Engine Completion and Presentation Readiness

- Basic Poker Game
- Having a start menu
- Developing a responsive and intuitive GUI
    - This includes a GUI for the start menu as well as the board
    - This also includes having buttons for the various functions such as “draw”
    - Having the board show where you can move pieces to

End of Week 4: Communication Protocol Development complete and Completion of Poker Game.

- Program needs to work correctly and needs to be able to be executed.
- The GUI has to be compatible
- Complete User Manual
- Complete Software Specification
- Complete Documentation

5.2 Team member responsibilities

The main four sections we will be partitioning our work into are

- The Underlying Poker Engine & AI
- The GUI
- Handling of the Communication Protocols

Based on the intensities of each section we will be dividing our group members to work on each section:

The Underlying Poker Engine & AI (Albert Huang & Johnny Wu)

The responsibilities of the people working on the Poker Engine will include

- Develop a Poker Engine
    - Actions:
        - Raise
        - Call
        - Check
        - Fold
        - Borrow
    - Supports multiple players and AI
- Various support/utility functions

- Successfully integrate with GUI and communication protocol

The responsibilities of the people working on the AI will include

- Developing a CPU that can play against a player without a huge wait time
- Developing a more Advanced CPU that can bluff
- Develop multiple AI IQ levels
- Calculate the percentage chance of winning

The GUI (Liam Fricker & Jimmy Le)

The responsibilities of the people working on the GUI will include

- Developing a responsive and intuitive GUI
    - This includes a GUI for the start menu as well as the board
    - This also includes having buttons for the various functions such as "draw"
    - Having the board show where you can move pieces to
- Having the visuals meet the expectations of a professional Chess Engine
- Adding minor quirks such as Piece Movement and Capture Tweens and Sound Effects.
- Adding Ease of Access.

Communication Protocols (Kent Mencel & Aakarsh Pasi)

The responsibilities of the people working on the communication protocol will include

- Displaying the GUI on the website.
- Constantly listen to requests to receive the data
- Send requests for the player that just made a move
- Reject requests that do not make sense

Back matter

Copyright.

The Five Aces Poker Game is a product of University of California, Irvine Irvine, CA 92697, United States.

Copyright © 2024 EECS 22L. All rights reserved.
No part of this manual may be reproduced or transmitted in
any form or by any means, electronic or mechanical, including
photocopying, recording, or any information storage and retrieval
system, without permission in writing from the Professor. Details
on how to seek permission, further information about the
Professor's permissions policies and our arrangements with
organizations such as the EECS department, can be found at
the website: https://engineering.uci.edu/dept/eecs.
This book and the individual contributions contained in it are
protected under copyright by the Professor (other than as may be
noted herein).

References

https://upswingpoker.com/poker-rules/
https://bicyclecards.com/how-to-play/basics-of-poker
https://www.pokernews.com/poker-rules/texas-holdem.htm
https://bicyclecards.com/how-to-play/texas-holdem-poker

Index