CSCI477 Elements of Games and Game Development
Course Project 2 – C++ Implementation on a Simple Poker Game

Version 4

Assignment Overview

With this project, you will develop a simplified poker game which can be played by humans or an artificial intelligence. Phase 1 of the project will have you implement the game and an Alpha AI, which performs per a set of supplied AI rules. Phase 2 of the assignment will have you implementing a Beta AI that should be superior to the Alpha AI. To prove this, you will create a Monte Carlo simulation of the software using the two types of AI and play them against each other with the expectation that the Beta player will beat the Alpha Player.

Phase 1 - The Game

The game that will implement is called Blockhead Poker. The rules for the game are attached below.

Basic Implementation

The game will be implemented using CLion and C++ using object oriented design. Code should meet the supplied coding standards.

The goal of this phase is to get the poker game to work in C++ using human players and/or an Alpha AI. The student should write a game driver that meets the rules given for Blockhead Poker. The I/O for the game may be console I/O (i.e. cin and cout) or the student may create a Windows or Graphic front end.

To keep things somewhat uniform, the student should use the following objects implementing the game. Instance variables should be private or protected (note, if I have forgotten something, let me know). If there is a reason to diverge from these objects, you will need to justify this in the documentation (and probably discuss it with your instructor).

enum PlayerType { HUMAN, ALPHA, BETA } - the type of player a specific player is (i.e. a human or a type of AI)

Object: Card - represents a single card in a deck
Public Methods:
Card(string cardName, int cardValue) or setCard(string cardName, int cardValue)
string getName() – get name of the card
int getValue() – get value of card per game rules
bool isFaceup() and *void setFaceup(bool)* - true if all players can see card

Object: Bet – Represent a single player's single bet.
Public Members:
Bet(int amount, int player)

int getPlayer() – player who made bet (0 or 1)
int getAmount() - amount of bet if it is a raise.

Object: Game

Public Methods:

b*ool playGame(PlayerType p0, PlayerType p1, int &chips0, int &chips1, bool reportFlag)* – Plays a game between a player of type p0 and p1. Chips is the number of chips each player respectively has at start and end of the game. reportFlag is just a flag to turn on and off I/O within the game (so you can do Monte Carlo runs without a lot of output). The method returns true if someone wants to quit the program.

Object: Hand – represents a hand or partial hand of cards

Public Methods:

void clear() – clears the hand
void addCard (Card card) – add a card to the hand
int getCount() – how many cards are in the hand.
Card getCard(int n) – Gets the n'th card in the hand.
Hand getVisible() - gets the visible part of a hand as a new hand
int evaluate() – what is the value of the hand

Object: BetHistory – The history of bets in a single round (you need it for human and AI decision making).

Public Methods:

void clearHistory() - clears the bet history
void addBet(Bet bet) – amount of bet
int getCount() – number of bets in history
Bet getBet(int n) – get the n'th bet in the history

Object: Player – This represents either a human or AI player. This is a base abstract class. You then should have derived classes for human, Alpha AI players, and Beta AI players (or more). The human player class will basically be I/O functions that presents info to the game player and gest decisions from a game player. The AI classes should make their own decisions without human input.

Public Methods:

Player(int id, int chips) – Initialize the player with player identifier (0 or 1) and starting chips.
int getBet(Hand opponent, BetHistory bh, int bet2player, bool canRaise, int pot) – This should be an abstract method that passes all of the necessary domain information of the game to the player. Other information about the player like the players hand should already be part of the Player object. The method should then return the bet made by the player. This bet represents the amount to be put in the pot, so it would include the amount bet2player which is the previous players raise. A bet of -1 is a command to quit (only comes from the human player). A bet of 0 is a fold IF there is a bet to the player (otherwise, it's just a call).
int getID() – get id number
void clearHand() – clear the players hand
void dealCard(Card c) – Add card to the players hand

Hand getHand() – get players hand
void addChips(int chips) – add (or subtract) chips from player
int getChips() – get players chip count

Object: HumanPlayer – This is a derived class of Player that presents domain information to the current human player through I/O and then allows the player to input their bet. Code should be implemented both to communicate to the game player the current status of the game (i.e. current hands showing, pot, bet history, etc.) and to validate the bets of the human player before returning the proper bet value. This uses the getBet() method.

Object: AlphaPlayer – This is a derived class of Player that evaluates domain information for the current AI player and decides on a player bet. This uses the getBet() method. The Alpha AI should use the attached rules in deciding the bet for the AI player.

Object: BetaPlayer – For Phase 1, this should be dummied out.

Deliverables

- The following files and directories must be included:
  - A short Word (1 to 2 page) document describing the work you are including and how you went about solving the specific assignment. Include a paragraph on any problems or interesting solutions you came up with.
  - A short test plan and implementation.
  - Your project directory and subdirectories, including all source code, data and a working executable.
- All files will be delivered in a single zip file containing any necessary directory structure. Usually, this just requires adding a file or two to your project directory, and then zipping it all up.

Grading

Game Implementation	50%
Code Quality	20%
Documentation	20%
Test Plan and Results	10%

Phase 2 – A Better (Beta) AI

The goal of this phase is to create an AI (Beta) that is consistently better than the Alpha AI and utilize Monte Carlo techniques to prove your new Beta AI is superior to the Alpha AI.

Part 1 – Building the Monte Carlo simulator

The goal of this part is to implement a Monte Carlo simulation that test the AI in playing against another Alpha Player. The simulation should play two Alpha Player's against each other for 100 games, then average the results for each player for those 100 games, and present those results.

There should be no user input other than possibly some initial conditions (like number of games to run in the Monte Carlo simulation).

Part 2 – Creating a Beta AI

The goal of this phase is to create a Beta Player using a Beta AI that can consistently beat a Alpha Player. The Beta Player may use any of the techniques learned in class but must meet these constraints…

- The AI needs to implement some level of fuzzy or random logic.
- The AI cannot cheat (i.e. it cannot look at its opponents face down card, evaluate the opponents hand with the facedown card or have other information a normal player would not have).
- The student is encouraged to actually write several versions of his Beta AI and play them against each other in order to create the best AI possible. As such, the player may want to add additional AI players.

Deliverables

- The following files and directories must be included:
  - A short Word (1 to 2 page) document describing the work you are including and how you went about solving the specific assignment. Include a paragraph on any problems or interesting solutions you came up with.
  - A short test plan and implementation using Monte Carlo techniques.
  - Your Visual Studio project directory and subdirectories, including all source code, data and a working executable.
- All files will be delivered in a single zip file containing any necessary directory structure. Usually, this just requires adding a file or two to your project directory, and then zipping it all up.

Grading

Game Implementation	50%
Code Quality	20%
Documentation	20%
Test Plan and Results	10%

Taking it Further

The below questions are presented for consideration if the student would like to take this project further.

- If you wanted to use real poker hands, how might you evaluate the hands?
- If the domain information contained data about previous hands, could the AI make use of that information, and how might it use it?
- How might you implement a learning AI?
- How might the AI be changed if the game was played with 6 players?

Blockhead Poker Game Rules

Blockhead poker is a very simplified version of stud poker. Below are the rules to the game.

1. The game is played between two players with a 52 card standard deck and poker chips.
2. A game is composed of 20 hands.
3. At the start of the game, each player will begin the game with 1000 chips.
4. The winner of the game is the player with the most chips after 20 hands.
5. At the start of each hand, the 52 cards are shuffled together, and three cards are dealt to each player, one face down and two face up. In addition, each player will contribute 10 chips (the buy in) into the pot.
6. A face up card may be viewed by both players, a face down card may be viewed only by the owning player.
7. A bidding round will commence. The first player will open the bidding in the first round and subsequent odd numbered rounds. The second player will open the bidding in the second round and all even number rounds.
8. Another card will be dealt face up to both player, and another bidding round commence with the player that opened the bidding in the first round opening the bidding in the second round.
9. A final card will be dealt face up to both player, and another bidding round commence with the player that opened the bidding in the first round opening the bidding in the second round.
10. The turned down card is displayed and the player with the best hand collects the pot. If the hands are a tie, the pot is carried over to the next round.
11. Winning hands are not decided like normal poker. Instead, each card has a value, and the five cards are added together to get the total value of the hand. The hand with the largest value is the winner.
    a. Card points are as follows:
       Ace – 1
       2-10 – 2-10
       Jack, Queen, King – 10
    b. Therefore the maximum hand value is 50.
12. A bidding round is handled in the following manner.
    a. Players alternately raise/call/fold until one player folds or both players call one after the other.
    b. The opening player may open with a call (actually a check) or raise. A call at this point means that no bet is made. A raise may be of any value from 1 to 10. This amount is put into the pot.
    c. The next player may do one of the following if the prior player called:
        i. Call, the round stops.
        ii. Raise – The bet may from 1 to 10. That amount is put in the pot. Note, a player may not raise if there have been three (3) prior raises in the bidding round.
    d. The next player may do one of the following if the prior player raised:
        i. Fold – The hand immediately ends and the pot goes to the prior player.
        ii. Call – The player puts into the pot the number of chips equal to the prior players raise.

iii. Raise – The player may raise any value from 1 to 10 chips. The amount of the raise plus the amount of the prior players raise is put into the pot. Note, a player may not raise if there have been three (3) prior raises in the bidding round.

e. Note, for the purpose of this assignment, a player may have negative number of chips (i.e. he borrows from the bank).

Some Discussion on Betting

Some of you may be rusty on the basics of Poker betting and similar games. Betting games are a major genre of games, and are beautifully designed for interesting gameplay and developing interesting AI's. This is because the betting process is relatively simple, but holds a great wealth of gameplay, strategy and skill. I thought I'd give a little more background on betting and how it applies to the project.

Basically, in any betting game, you have some type of game in which a winner of the game takes a pot that all of the players contribute to. In that context, a basic strategy is to make the pot as large as possible if you have a good chance of winning the pot, but keep the pot as small as possible if you probably are going to lose. One interesting aspect of betting games is that a player may choose to quit the game and contributing to the pot, but then he/she automatically loses. This is where the bluff comes in where you try to convince all the other players that they are not going to win the game, and therefore they drop out before the game is resolved.

In Phase 2 of the project you will be asked to write an AI that makes these decisions. That means you need to get a good feel for how betting is done, and how humans make betting decisions. If you don't know how to bet, do some research and maybe get some friends together with match sticks and play some Blockhead poker.

So how does betting work? This varies from game to game, but basically in a game there will be one or more rounds of betting (in Blockhead Poker there are three rounds). In each round each player may bet in sequence until the betting round ends. A betting round will end when…

- One player folds, then the other player wins the pot that the hand is over.
- A player calls (i.e. he bets an amount equal to the raise to him).
- Both players check.

Note that a player may not raise after 3 raises so the betting will end either as a call or a fold. So what are the possible betting actions? They are…

Check - This may only be done at the start of the betting round and no bets have been made. This is a 0 bet.

Raise - The player bets an amount equal to the raise made by the previous player PLUS his/her own raise (which must be somewhere between 1 and the max raise).

Call - The player bets an amount equal to the previous players raise.

Fold - The player bets nothing but the hand is finished and the other player wins what is in the pot.

Notes:

- A player cannot check if the previous bet was a raise (i.e. bet2player > 0).
- A player cannot fold unless the previous bet was a raise (i.e. bet2player > 0).
- A player cannot raise if there previously have been three raises (i.e. canraise == false).

Note, the AI/player doesn't have to count the raises. The game engine should do this and and tell the player with the canraise flag.

- A player cannot call if the previous bet was a check (i.e. a bet2player == 0)
- The previous bet will always be a check (bet2player == 0) or a raise (bet2player > 0).

Any other situation (a call or a fold) would have stopped betting.

The method GetBet() is called every time it is a players turn to make a bet. The method does two and only two things. It tells the player the status of the game (i.e. his hand, his opponents showing hand, the size of the pot, the bet to the player, and whether the player can raise) and it returns the bet the player makes. Note, it does not change the size of the pot, end the game, make multiple bets or anything else with the game itself. This is all handled by the other game engine code.

So what is the value that is supposed to be returned from GetBet()? Here is a summary of what it may be…

-1 - The human player wants to exit the computer program. This is passed back to the game engine, and the game exits.
0 - If bet2player == 0, this means check.
If bet2player > 0, this means fold
Amount equal to the previous players raise (i.e. bet2player) - call
Amount greater than the previous players raise (i.e. bet2player)- raise

Note, bet2player contains the previous players raise if it is greater than 0. As such, the value returned should be bet2player PLUS current players raise.

I hope this helps. Is this complex? Yes and no. It took me a while to write it out and there are a number of permutations, BUT as requirements for code in a game (or any computer program), this is pretty simple stuff.

Rules for Alpha Type AI

1. A delta value is calculated by getting the value of the AI players hand and subtracting the value of the opponent's visible hand.
2. If there are still cards to draw…
    a. If there have been no bets prior…
        i. If delta > 10, raise 10
        ii. If delta > 5, raise 5
        iii. If delta > 0, raise 1
        iv. Else call
    b. Else there have been prior bets
        i. Calculate a pot_factor which is the size of the pot divided by 10 (int value)
        ii. If the prior bet is a call
            1. If delta > 5-pot_factor, raise 10
            2. If delta > 0-pot_factor, raise 1
            3. Else call
        iii. If the prior bet is a raise and is less than 1+pot_factor*2
            1. If delta > 8-pot_factor, raise 10
            2. If delta > -2-pot_factor, raise 1
            3. If delta > -4-pot_factor, call
            4. Else fold
        iv. else
            1. If delta > 10-pot_factor, raise 10
            2. If delta > 0-pot_factor, raise 1
            3. If delta > -2-pot_factor, call
            4. Else fold
3. Else last betting round
    a. If there have been no bets prior…
        i. If delta > 10, raise 10
        ii. If delta > 5, raise 5
        iii. Else call
    b. Else there have been prior bets
        i. Calculate a pot_factor which is the size of the pot divided by 10 (int value)
        ii. If the prior bet is a call
            1. If delta > 10-pot_factor, raise 10
            2. Else call
        iii. If the prior bet is a raise and is less than 1+pot_factor*2
            1. If delta > 6-pot_factor, raise 10
            2. If delta > 2, call
            3. Else fold
        iv. else
            1. If delta > 8-pot_factor, raise 10
            2. If delta > 4, call
            3. Else fold