Project 1

STL Multiplayer Poker

https://github.com/KyleRiebeling/STL_Poker

CIS-17C 47065
Kyle Riebeling
4/23/2023

Introduction

Poker is a very common gambling game. Players will place bets to go to the "pot", which the winner earns after the round is over.

Each player pays a buy-in bet, then is dealt two cards. Based on what the two cards are and how the player feels, they can either fold and forfeit their bets for the round, raise the bets and require everyone else to do the same or give up, or call and just stay in the game and wait for the next round. This betting method is held after every card dealing round. The card dealing rounds are as follows: deal 2 cards to the players, play 3 cards on the table, place one more card on the table, and place one last card on the table.

Using the 5 cards on the table and the 2 cards in their hand, the players try to create the best hand of 5 that they can. The ranking of best hand to worst hand is as follows: one pair, two pairs, three of a kind, straight, flush, full house, and four of a kind. The player who had the highest ranking hand at the end of the last card deal wins the money in the pot. In the event of a tie in hand rankings, the winner is determined by the highest ranking card relative to what hand they have,

Summary

Project size: about 800 lines
Number of variables: about 18
Number of functions: 28

Containers used:
1.list: Lists were used to simulate the dealer's and players' hands. Using the dealCard function, one card from a shuffled deck is added to the list containers in the Table class.

2.map: A map<string,int> container is used to track the players names and money remaining. Iterators are used throughout the game loop to access players names and take out money from bets and add the pot to the winner.

3.queue : a queue was used as a container to hold the shuffled deck of cards that are dealt out. When the queue is empty, a new shuffled deck is added to the queue. This was done to prevent any of the 52 cards in the original deck from getting modified or deleted throughout the code. There is an array that holds the full unshuffled deck, then a queue that receives a shuffled version of that array.

Iterators used:
1.Random access iterator: This type of iterator is used in the random_shuffle algorithm. This is the reason I used an array over any other container for the deck of cards. The types of containers that can use random access iterators are very limited, but arrays are one.

2. Input iterators: Despite the conflict with the name, these iterators are used to output what it is accessing. I used these to output players names and money amounts. These are also used to output the contents of the list that contain card objects.

3. Output iterators: Similar to input iterators, output iterators are used for the opposite of what they are called. I used output iterators to modify the contents of the player maps.

Algorithms used:
1.count: the count algorithm is used to find if there is an instance of 2 or 3 in the facesCounted array created in the evaluateHand function. That array is passed to pair() and threeOfAKind() where the count algorithm checks if there is a 2 or 3 in any element of the array, which means there is a pair or three of a kind in the player's hand.

2.Random_Shuffle: Random_shuffle was used to shuffle the standard deck array, which was then copied into a queue for dealing a unique random deck every time the program is run or when the full deck has been dealt already.

3.max_element: max_element is used to compare the results from the evaluateHand function. It will return the index in the array of the highest hand. This is then used to push an iterator to the respective player to declare them the winner

4. Max: max is used to clear the cin stream when someone enters an input that causes cin.fail() to return true. Max is used to make sure that no amount of junk input remains after calling cin.ignore().

Pseudo Code

Welcome user and prompt for amount of players
    Make sure input is valid for amount of players

Declare a Table object with the amount of players
    Set up starting cash and names for each player

Declare a deck of cards
    Shuffle that deck of cards
    Send the shuffled deck of cards into a queue for drawing cards during the game

Take initial buy-in bets from players

Deal 2 cards to each player
    Take turns displaying their cards and then clearing terminal to keep hidden from other players

For turn 2-4

Ask for more bets/calls/folds
Display appropriate amount of dealers hard

Turn 5
Ask for any last bets/folds/calls
Reveal all players resulting hands
Compare those hands to declare winner

Once winner is determined
Add their winnings to their map variable

Prompt users to continue playing
Eliminate any players who cant afford the buy-in bet

Major Variables

Type	Variable Name	Description	Location
Int	numPlayer	Amount of players	main()
Table	pTable	The table object for the program	main()
Int	turn	Tracks the current turn of the game	Class Table
map	players	Map that carries players names and their cash	Class Table
map	activePlayers	Map that carries temporary player data during gameplay to track who is active in each round	Class Table
int	pot	Total of bets placed per round	Class Table
Int	currentPlayer	Counter used to count each player per turn	Class Table
list	player1/2/3/Cards dealerCards	Lists that hold each players cards per round	Class Table

Type	Variable Name	Description	Location
int	highBet	Holds the amount raised that needs to be matched by each player if a player decided to raise	Class Table
queue	shuffled	The shuffled queue of cards	Class DeckOfCards
int	currentCard	Counter for how many cards have been dealt	Class DeckOfCards
int[]	facesCounted	Counts how many of each face value are in a player's hand	DeckOfCards.evaluateHand()
int[]	suitsCounted	Counts how many of each suit are in a player's hand	DeckOfCards.evaluateHand()
string	face	Card's face value	Class Card
string	suit	Card's suit	Class Card

UML Diagrams