SIMULATING THE MONTE CLARO GAME (BLACKJACK) USING ARRAYLISTS

Abstract:

In this project, I explored the concepts of ArrayLists in Java to simulate the famous MonteClaro Game: Blackjack. In the game where a player and a dealer have to pick (or deal) a card from the deck (card stack) continuously, where we might not be able to predict the number of cards a player/ dealer will pick each turn to achieve their goal total sum (16<=goal<21 and 17 respectively), the concepts of array list is very important here as it models the real world situation of the ever-dynamic card picking and card deck for the real world card.. In an arraylist, the size of the list is not fixed and can be manipulated to fit the situation of the card in the hands of the player and dealer at any point in time . We can remove cards from the array list as players play the cards as well as add new cards as players deal (or pick) cards from the deck. I simulated this exact situation using the dynamics of arraylists.

Methods:

In this project, I made a lot of classes to represent each object or entity in the game- the card deck (Deck), the hands representing the player and dealer (Hand), the card (Card) which is the object that will be played, the game itself as the whole play goes(Blackjack). Instead of using the java ArrayList package, I rather implemented the whole ArrayList logic in my code and built all the functions(add, remove, resizing) myself. Hence, the whole simulation was based on my own written ArrayList program. I did this in my class I wrote my code for 3 different simulations: Simulation (S1) where each register has its own line and people can choose which line to wait in as they arrive; Simulation (S2), where people wait in a single line and customers checkout in a one-after-the-other sequence; Simulation (S3), where there are 2 lines and the simulation for each line is like Simulation (S1), for half of the cash registers. Each queue was represented with an ArrayList. For the first simulation, I created a specific number of array lists to represent the specific number of queues I'd like to have, then each customer in the queue was represented with an object Customer . For the second simulation (S2), I created a single array list and removed and created customer objects to the list as needed. For the third simulation, I created 2 array lists to represent the two queues in this context.

Results:

In my experiments, I ran a simulation of my Blackjack game by writing a java program called Simulation and Blackjack that allowed me to run the game for specific scenarios. This scenario involves the simulation of the blackjack game for 1000 and 10000 rounds respectively. I wanted to analyze what percentage of the game was won by the dealer and the player respectively and also determine what percentage of the game resulted in a draw for each scenario. My main goal was to see how much these

metrics (game distribution) changed with the number of simulations per scenario. I also wanted to determine how random the game process is and whether it agrees with the real world situation where the game winning chances are probabilistic.

No of Simulation Rounds	Number of Games Won by The Player	Number of Games Won by The Dealer	Number of Games Resulting in Draw
1000	411 (41.10%)	490 (49.00%)	99 (9.90%)
10000	4030 (40.30%)	4980 (49.80)	990 (9.90%)

_Simulation Result Statistics of The Blackjack Game for 1000 Rounds

From the result of the experiments, I think it makes sense that the dealer(otherwise known as the casino itself) won the majority of the game (49 and 49.80%) for this high number of rounds. This is mainly due to the fact that the odds are usually in the favour of the dealer because it plays second and is able to observe the player's move before strategizing its own.. The casino usually has a house edge advantage (or second mover advantage) in the game which makes it very possible for the dealer to win the game even when the player uses their optimal strategy. From the way my Deck Class is implemented, the randomness of the shuffle method and the way the card is dealt to the player and dealer enabled me to model the real world situation of the house edge advantage for the dealer.

Also, it seems as the number of simulation rounds increases, the dealer tends to win more games, probably due to the fact that their strategies implement over the player's strategies as more and more rounds are played. This fits closely with the fact the probability of the dealer winning increases as the size of the game increases.

Extensions:

For my extension, I decided to create a new method in my Blackjack class that simulates a kind of interaction between the player and the program. Here, the player has a chance to enter some dynamic inputs (hit / stand)- into the program so as to control their own fate (as they would in real life) rather than base it on some random probabilistic model. Hit means the player still wants to choose a card from the deck; Stand means they do not want to choose a card. This is very important as it gives the player the ability to make decisions instead of being forced to get dealt with two cards from the deck.

For this, I created a new class called Interactive that acts like a bridge between the player and the simulation. This class functions by receiving the player's inputs, validating the input so that we don't run into errors while running the game, and providing an immediate response / feedback on the state of the game.

Here, I also used my interactive class in my Simulation code which allows the player to manually choose whether they want to play an interactive game or they just want to run an automatic simulation. I did this by making changes to the game method in the Blackjack class to accept an InteractivePlayer object when the game is in interactive mode. This gives flexibility to the program's user whether they want to play with control or just simulate the whole game. The user also has the option to opt out of the program when it is interactive. They can choose to stop playing the game or continue playing if they still feel like doing so.

My extension improved the program because it allows the player to be able to turn the tables if they play strategically enough with the computer. If they observe their card is closer to 21, they might choose to hit / stand rather than being confined to hit every time because the deck automatically deals a card to them.

To run this simulation, you can use the **java -ea Blackjack** on the terminal

Acknowledgements:

I consulted a lot of Youtube videos on the use of ArrayList implementation and related data structures. I also went to the professor's office hours and the TA hours (Ben) for a better explanation on ArrayList and related java syntax. I also studied the code from the class notes to get a better understanding of the related concepts. I also discussed code concepts with Emmanuel Buabeng during my project testing and some implementations.