DSA I Lab 3: Blackjack Simulator

Christine Baca [cab8xd]

[1] University of Virginia, Charlottesville VA 22904, USA

Abstract. The experiment focused on creating the ideal betting strategy for the gambling card game Blackjack. Two strategies focused on optimizing the getBet() method and informing the player the optimal amount of chips to bet in an oncoming hand based on the result of the previous hand were tested. The first strategy concentrated on raising and lowering the bet based on the previous hand's result. The second strategy emphasized only decreasing the bet when the previous hand lost and increasing when the bet amount reached zero after a specific number of games had been lost. The experiment concluded the second strategy proved to be the most optimal because of its conservative approach and reluctance to bet more chips despite any winning streaks and, overall, decreased the number of coins lost over a series of Blackjack hands.

Keywords: Blackjack, Java, MyBlackJackPlayer.java, getBet()

1 Introduction

The lab focused on making a Blackjack bot via the implantation of the methods getBet() and getMove(). Blackjack is a gambling card game that places the player against the dealer. The basic rules of Blackjack comprise of the player drawing two cards: one is known immediately to the player, and the other is discovered later in the game. Several moves are available to the player, including actions to draw more cards or to stop drawing cards. By the end of the round, whichever individual has the sum of the cards closest or equal to (but not exceeding) 21 will win the round and gain chips; the loser loses the chips they betted at the beginning of the round to the winner.

This experiment focused on, rather than the bot's ability to assess the cards given and create the optimal move, the best betting strategy to assess how many chips the player should bet per round based on the previous round's performance. Two bots with two different getBet() methods will compete in numerous games of Blackjack and their resulting final number of chips will be compared and analyzed to see which bot implemented the better getBet() or betting strategy; the getMove() method or Blackjack strategy will be constant and unchanged for both bots.

2 Methods

2.1 Blackjack getBet() Strategy #1

The first variation of getBet() uses several if statements , the variable formerChips (initially set to the integer amount of chips the player starts with in the beginning of the series of runs), and the variable bet (integer variable initially set to thirty). If the number of chips given is greater than the former amount from the previous game, the bot increases its bet for the oncoming bet. Similarly, if the number of chips is less than the former amount, then the bot decreases its bet for the oncoming hand. To compensate for the possibility of zero-chip bets in the program which will occur if the player loses a certain number of times, the bot will check if the bet is zero and, if so, add a set amount of chips to the bet.

Code Sample:

```
//The initial bet amount for this robot is 30 (int bet = 30 is declared
before the method is initialized, similarly to formerChips)
//if the number of chips is greater than the former amount, increase the
bet
        if(getChips() > formerChips) {
                bet = bet + 5; System.out.println("INCREASE");
        }
//else if the number of chips is less than the former amount, lower the
bet
        else if(getChips() < formerChips) {
                bet = bet - 5; System.out.println("DECREASE");
        }

        if (bet == 0)
                bet += 5;
//Updates formerChips value to current chip value
formerChips = getChips();
//returns integer bet
return bet;
```

2.2 Blackjack getBet() Strategy #2

The second variation of getBet() or the player's betting strategy follows the example of the first by using several if statements , the variable formerChips (initially set to the integer amount of chips the player starts with in the beginning of the series of runs), and the variable bet (integer variable initially set to thirty). But, the strategy excludes the bet-increase if statement. Instead, the bot adds a higher amount of chips to the bet once a number of losses reduced the bet amount to zero, effectively resetting the amount of chips being betted to the amount initially betted unlike the first Blackjack strategy which added only a small amount of chips to the bet when it was reduced to zero.

Code Sample:

```
//The initial bet amount for this robot is 30 (int bet = 30 is de-
clared before the method is initialized, similarly to formerChips)
        if(getChips() < formerChips) {
                bet = bet - 5; System.out.println("DECREASE");
        }
        //reset to initial bet value
        if (bet <= 0) {
                System.out.println("RESET");
                bet += 30;}
//Updates formerChips value to current chip value
formerChips = getChips();
//returns integer bet
return bet;
```

2.3 Predictions

The two getBet() methods attempt to optimize the amount betted per hand in consecutive games of Blackjack, using the information of the previous game to help the player decide what would be the smartest bet to place. The first getBet() strategy illustrates a direct relationship between the player's hand results and the resulting bets the player places. If the player is winning, the bets get higher, and, if the player is losing, the bets are lowered. This is a logical thinking if the individual believes that they are experiencing a winning or losing streak and need to increase or conserve their chips.

The second strategy creates a more conservative approach; the player only reduces their chips and increases the betting amount after a certain amount of losses occur; the second strategy allows less risky gambles for the player but disallows them from ever "winning big" or receiving a high yield of chips from a single hand.

The experiment focuses on comparing the two getBet() methods and deciding whether playing accordingly to luck or acting generally conservative when betting chips proves more efficient for final winnings after a series of hands.

3 Experiment:

The procedure for the experiment runs each of the two variations of Blackjack players through ten simulations of a thousand hands each and records the resulting bets gained and lost. The average and standard deviation of these hands will be calculated and recorded. The data will be analyzed based on which bot performed better and the differences between the bots' respective averages and standard deviations.

3.1 Experiment Table:

	Bot #1	*Bot #2*
No. of Simulation Runs	10	10
No. of Hands Per Simulation Run	10,000	10,000

Total Hands	100,000	100,000

General Information	
Initial Bet (int bet)	30
Initial Chips Given (Note: The same initial value as int formerChips)	3,000

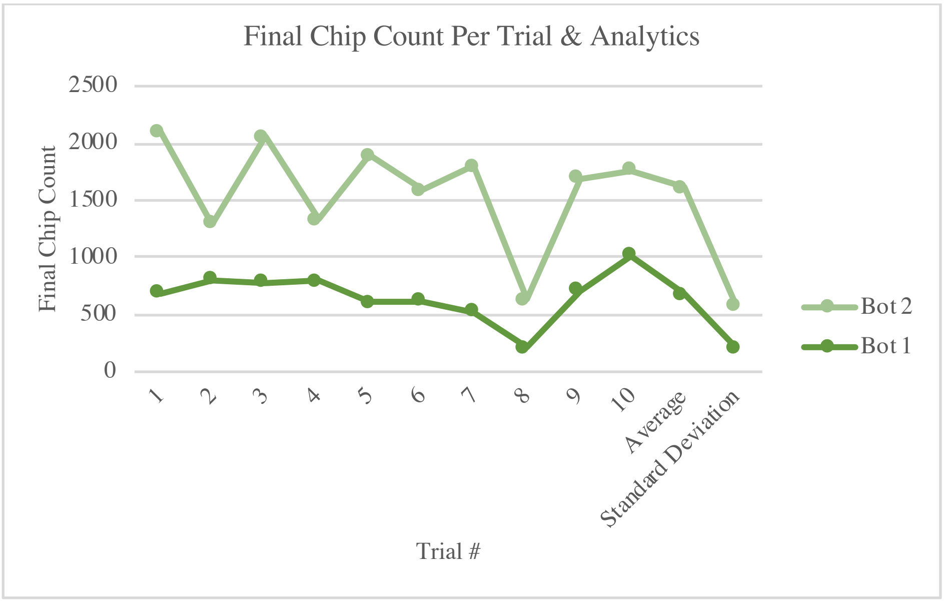
4 Results

4.1 Table Results

Bot Comparison for Final Number of Chips and Analytics of Overall Results

Trial #	Bot 1 Final Chip Count	Bot 2 Final Chip Count
1	685	1414
2	804	496
3	778	1278
4	788	547
5	606	1287
6	619	968
7	529	1267
8	212	419
9	708	983
10	1012	747
Average	**674**	**941**
Standard Deviation	**210**	**370**

4.2 Graph Results



5 Conclusion

The experiment concludes that the second and more conservative strategy proves more efficient than the more direct first strategy for the getBet() method in consecutive games of Blackjack. The results show that the second strategy repeatedly scored a higher final chip count than the first strategy. The first strategy focused on increasing and decreasing the bets based on the previous hand's performance while the second strategy only decreased the chips when previous hands were lost and added chips when the bet amount reached zero after a certain amount of losses. Because the second strategy likely betted less chips overall than the first strategy, the player conserved more chips when experiencing losses and ended with a higher amount of chips than if the player attempted to bet more chips based on previous wins. The second strategy holds a higher average number of chips than the first strategy by almost a third. The same relationship holds for the standard deviations of the two methods with the second strategy having a higher standard deviation than the first strategy. Errors of the experiment include bugs that allowed the player to lose more chips than initially held and possible logic errors that may have skewed the program's intentions and respective results. Future experiments could test more complex methods of calculating the player's next bet and account for possible card counting.