Basic Java 2 - Blackjack Analysis

Timofey Maslyukov

University of Virginia, Charlottesville VA 22903, USA

gaf4sw@virginia.edu

Abstract. The issue of creating a program to maximize blackjack profits was analyzed in this paper. In order to find the optimal method of playing blackjack two methods were used. The first method played blackjack in the most utilized way by looking at the cards dealt and making the best possible move from there. The second method looked if the previous bet was a win or loss and if a loss doubled the bet. The results from conducting each method 10 times with 1000 runs showed the second method to be more efficient. Higher chip counts were found to be given with a high standard deviation or risk. The most optimized way to play would still be to use the second method but use the first method in order to help the program make more wins instead of taking a long time to garner a win.

1 Introduction

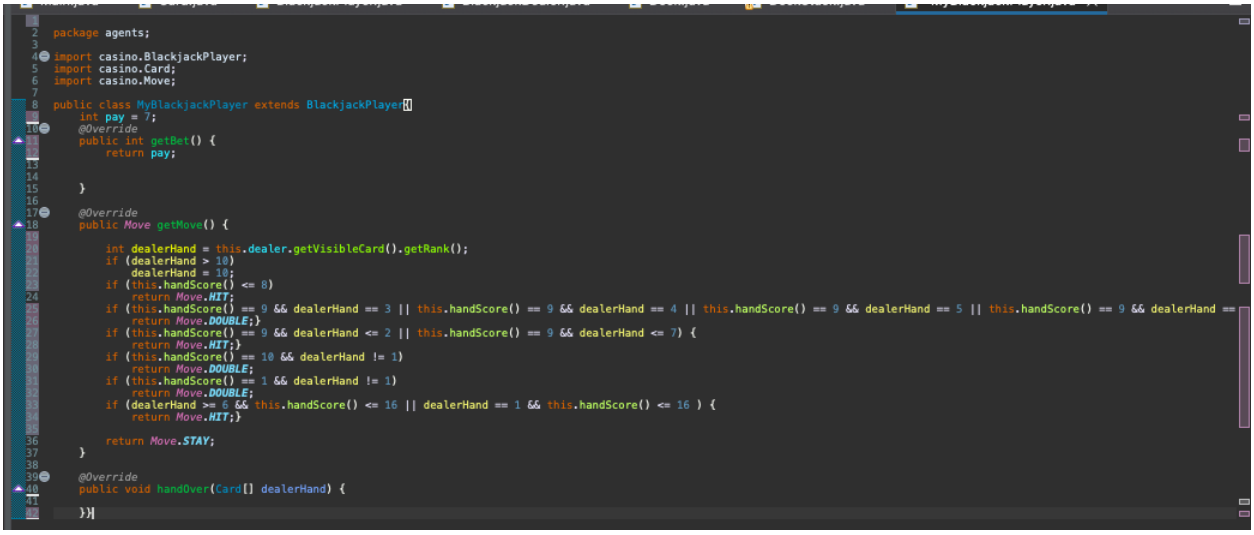
1.1 Problem

The issue presented in this analysis is to discover an effective method of creating a program which maximizes the profits in a game of blackjack. Blackjack in its essence is rigged in favor of the dealer and only using extra maximizing/ cheating can definitive gains be made. Programming a card counting would be very hard so it was not ultimately used in this analysis.

1.2 Method 1

The first method used to maximize profits is by following a card which supposedly provides the best mathematical chance of winning. For example when the player was dealt a face card or an ace a double move would be done. By looking at the dealer's dealt card the program was able to make its choice on the type of move it should make. This allowed it the best chances to win at the standard game. The same bet was done every time as well with a bet of 10. Over 1000 runs the program would show a standard average on the amount it would win or lose as well.

Table 1. Shows the code for the first method



```
package agents;

import casino.BlackjackPlayer;
import casino.Card;
import casino.Move;

public class MyBlackjackPlayer extends BlackjackPlayer{
    int pay = 7;
    @Override
    public int getBet() {
        return pay;


    }

    @Override
    public Move getMove() {

        int dealerHand = this.dealer.getVisibleCard().getRank();
        if (dealerHand > 10)
            dealerHand = 10;
        if (this.handScore() <= 8)
            return Move.HIT;
        if (this.handScore() == 9 && dealerHand == 3 || this.handScore() == 9 && dealerHand == 4 || this.handScore() == 9 && dealerHand == 5 || this.handScore() == 9 && dealerHand ==
            return Move.DOUBLE;}
        if (this.handScore() == 9 && dealerHand <= 2 || this.handScore() == 9 && dealerHand <= 7) {
            return Move.HIT;}
        if (this.handScore() == 10 && dealerHand != 1)
            return Move.DOUBLE;
        if (this.handScore() == 1 && dealerHand != 1)
            return Move.DOUBLE;
        if (dealerHand >= 6 && this.handScore() <= 16 || dealerHand == 1 && this.handScore() <= 16 ) {
            return Move.HIT;}

        return Move.STAY;
    }

    @Override
    public void handOver(Card[] dealerHand) {

    }}
```

1.3 Method 2

The second method would be to change bet amounts from game to game based on the result of the previous game. The player would hit until they reached 16 no matter the cards they or the dealer were dealt. Known as the Martingale betting method theoretically if a player was given infinite money they should always be in a net positive since they always bet twice their loss. The starting bet was 1 chip and was doubled after every loss

Table 2. Shows the code for the second method

```
package agents;

import casino.BlackjackPlayer;

public class MyBlackjackPlayer extends BlackjackPlayer{
    int hand_before = 1000;
    int hand_after = 1000;
    int dealerHand2 = 0;
    int counter = 1;
    int pay = 1;

    @Override
    public int getBet()
    {

        System.out.println("gt move hand b4 " + hand_before + " hand aftr " + hand_after);
        if (counter == 1) {
        if (hand_before < hand_after)
            pay = pay * 2;
        if (hand_before > hand_after)
            pay = 1;
        if (hand_before == hand_after)
            pay = 1;}
        System.out.println("I bet " + pay);
    return pay;


    }

    @Override
    public Move getMove() {
        /* Hits until we get a score of 16 or better */
        if (this.handScore() <= 16)
            return Move.HIT;
        hand_after = this.getChips();

        return Move.STAY;
    }

    @Override
    public void handOver(Card[] dealerHand) {
        int dealerHand2 = this.dealer.getVisibleCard().getRank();
        if (dealerHand2 > 10)
            dealerHand2 = 10;
        hand_before = this.getChips();
```

2 Methods

2.1 Method of analysis

In order to test each of these methods a standard analysis experiment was done. Each method was tested 10 times with 1000 games in each run. After the 10 runs were completed and documented, the results were put into a table as well as averaged by the total sum of all the runs divided by the number of runs. The standard deviation was calculated by using a standard deviation calculator.

Table 3. This table shows the methods used as the number of runs

Method	Number of runs
Method 1	10
Method 2	10

3 Results

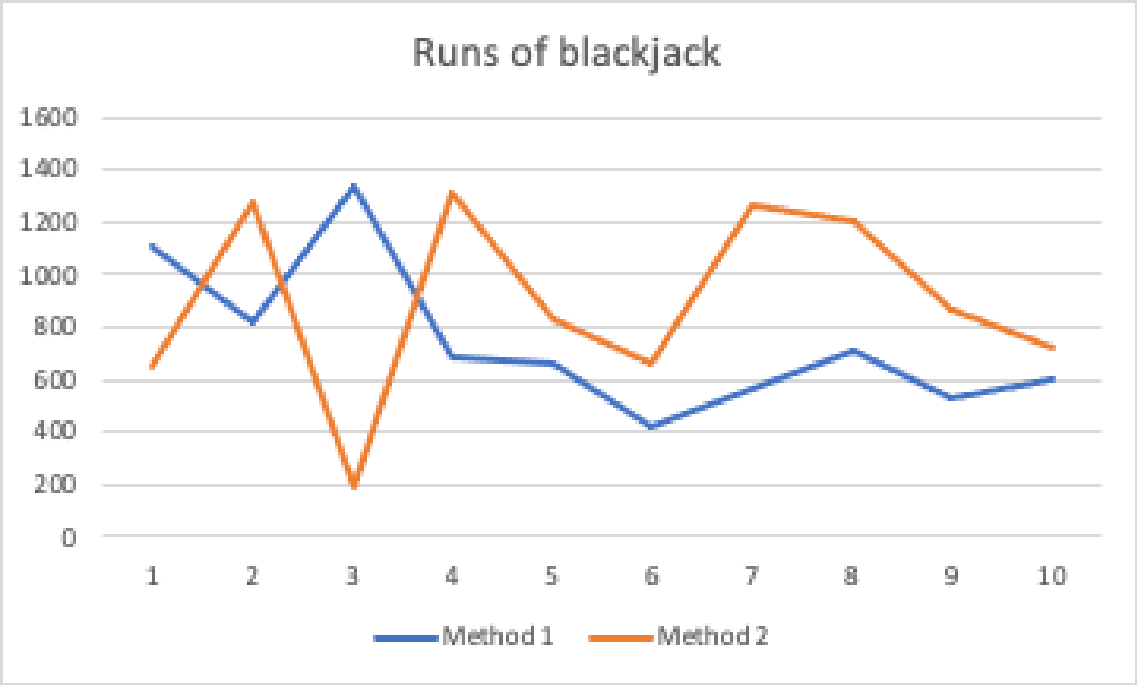
3.1 Results of different methods

The results showed that the standard method of betting which looked at the dealt cards and used the best bet produced lower results then doubling the bet with each loss. The average for the first method was 743.1 with a standard deviation of 265.3. While the second method produced an average of 897.3 and a standard deviation of 344.2. This showed that the 2nd method produced higher results of chip count on average with a higher risk than the first method.

Table 4. This table shows the results of each method's test and its average and standard deviation

	1	2	3	4	5	6	7	8	9	10	Average	Standard Deviation
Method 1	1110	816	1336	688	659	426	560	710	528	598	743.1	265.3
Method 2	653	1274	195	1308	825	660	1263	1207	861	727	897.3	344.2

Table 5. This graph shows chip total at the end of each run compared across methods



4 Conclusion

4.1 Which method was better

In conclusion the second method of doubling on losses produced better results. I believe this since the average was so much higher and if you continued running you would definitely make a higher amount of chips than the first method. The first one is more consistent at a similar amount as well. I think if you continue playing and don't end on a loss for the second method you should always be in a net positive as well. I believe that a combination of both methods would produce the best results with minimal losses. Also the second method would reset back to a standard bet at a loss which means that some losses would occur.