Mississippi Stud Analysis

Oliver Reinke

April 2023

1 Introduction

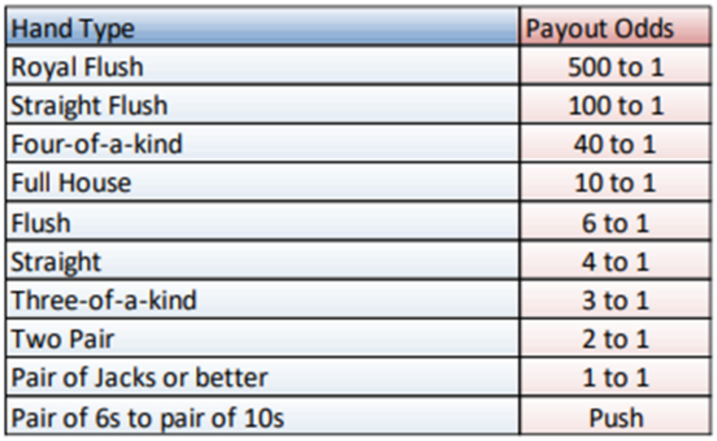
Mississippi Stud is a table game found in most casinos across America. In this project I will find the expected value based off the player's first 2 cards. First, I will go over how the game is played. Next, I will go over the different types of hands and their value. Using these hands, I will analyze the expected value of each of the different types using their payout values and probability of being dealt the cards. Finally, I will code a program that will run through all the possible hands and community cards, comparing the outcomes from the simulation to the outcomes calculated in part 2.

2 Rules Of Mississippi Stud

Mississippi Stud is played with a standard deck of 52 cards. For the purposes of this analysis, there is only one player, as the other players holding do not affect each other's cards or payouts. By the rules of the house, players are not to share with others information about their cards, so this can not be used to eliminate cards from play.

To start play, the player will place an ante, which tends to be a minimum of five to fifteen dollars depending on the house. Once an ante is placed, the player is dealt two cards, which they are allowed to examine. After seeing their cards, the player can choose to fold, meaning that play concludes and the ante is forfeited, or they can place a bet of one to three times the ante. If a bet is placed, a community card will be dealt, which once again the player is allowed to inspect. The player can then again either fold, forfeiting the ante and previous bet, or place a further bet. A fourth card is then dealt for the player to examine, the player folds or bets, and a final fifth card is dealt, concluding play. The player then makes the best possible poker hand, and is paid out a multiple of their total bets and antes according to the chart below.

For the purposes of this analysis, we will only be considering the starting hand and then dealing all 3 community cards without further bets required for cards four and five, as the analysis required expands incredibly. The expected value will represent the multiplier applied to bets and ante, with -1 representing a loss of ante and bets, 0 representing a return of ante and bets, which will be referred to as a push, and any positive value returning antes and bets multiplied by that value plus one. For example, if the expected value of a specific hand is 1, ante of 5 dollars, and bet of 15 dollars, the player will be returned 40 dollars. If the expected value was 0, the player would be returned 20 dollars.



Hand Type	Payout Odds
Royal Flush	500 to 1
Straight Flush	100 to 1
Four-of-a-kind	40 to 1
Full House	10 to 1
Flush	6 to 1
Straight	4 to 1
Three-of-a-kind	3 to 1
Two Pair	2 to 1
Pair of Jacks or better	1 to 1
Pair of 6s to pair of 10s	Push

(Source 2)

3 Starting Hands

The 2 cards dealt to the player at the start of the game will be referred to as the starting hand. Starting hands fall into the 2 main categories: Paired and Unpaired. Paired hands are those where both are of the same number. Unpaired hands are those where the numbers do not match.

Unpaired hands also fall into several subcategories due to the payout table differentiating between the numeric value of paired hands. Numeric value of single cards are high, middle, and low. Aces, Kings, Queens, and Jacks are considered high cards. 10's, 9's, 8's, 7's, and 6's work are considered middle cards. 5's, 4's, 3's, and 2's are considered low cards. For unpaired starting hands, any combination of 2 single card categories can be used, resulting in the categories of: two high cards, one high one middle, one high one low, two middle, one middle one low, 2 low.

Before calculating the expected value of each hand, we will calculate the probability of each category of hand. The first card dealt will be taken as given. This leaves 3 cards of 51 remaining cards having the same numeric value as card one. So, the probability of being dealt a paired hand is $\frac{3}{51}$. All other hands will be unpaired, resulting in a probability of $\frac{48}{51}$.
The subcategories of Unpaired hands will be dealt at the probabilities in the table below:

Starting Hand	Calculation	Probability
2 High cards	$\frac{16}{52} \times \frac{12}{51}$	0.07239819
1 High, 1 Mid	$2 \times \frac{16}{52} \times \frac{20}{51}$	0.2413273
1 High, 1 Low	$2 \times \frac{16}{52} \times \frac{16}{51}$	0.19306184
2 Middle cards	$\frac{20}{52} \times \frac{16}{51}$	0.12066365
1 Mid, 1 Low	$2 \times \frac{20}{52} \times \frac{16}{51}$	0.2413273
2 Low cards	$\frac{16}{52} \times \frac{12}{51}$	0.07239819

Adding up all of the final probabilities results in p(Unpaired) = .94117647, which is equivalent to $\frac{48}{51}$, verifying that this probability is correct. There are 2 further subcategories of unpaired hands: Suited and Connected. Suited cards are those with matching suits and connected cards are those where a straight is possible, or where the numeric value of the higher card minus the value of the lower card results in a difference of 3 or less. The expected value of these 2 types of hands will be calculated separately, as they only affect optimal game strategy for the starting hand 65 suited, as calculated by Michael Shackleford on the website Wizard of Odds (Source 1). Along with this, the final hands of straights and flushes cannot be present when any pair is made, as all 5 cards are needed, making them mutually exclusive with other final hands (One pair, Two pair, Three of a kind, Full house, and Four of a kind all require at least one pair).
The probabilities for type of paired hands are much more simple, as the second card will be the same numeric value as the first by the definition of a paired hand. So, to get the probability of a category of High, Middle, and Low card pairs, we will multiply the probability of getting a paired hand, $\frac{3}{51}$, and the ratio of number of cards in category to the full deck.

Starting Hand	Calculation	Probability
High pair	$\frac{3}{51} \times \frac{4}{13}$	.018099548
Middle pair	$\frac{3}{51} \times \frac{5}{13}$	.022624434
Low pair	$\frac{3}{51} \times \frac{4}{13}$	.018099548

These 3 probabilities add up to .05882353, which is equivalent to $\frac{3}{51}$, verifying the probabilities.

4 Expected Value

Now that we have the probabilities of each starting hand category, we can calculate the expected value of each hand. For every starting hand, there are 50 choose 3 outcomes, as three cards are dealt from the remaining 50 cards and order is not relevant for this analysis (Besides whether the cards are part of the starting two or three community cards). 50 choose 3 will be the denominator of every probability, with the numerator being a combination representing the number of community cards combinations resulting in that particular outcome.

To begin, the probabilities of final hands containing pairs must be calculated. Every final hand higher than one pair will have the different probabilities for paired and unpaired starting hands, but not for the subcategories within those two, as numeric values of pairs are treated differently for only one pair hands.
Paired Hands:

End Hand	Calculation	Probability	Payout Multiplier
Two pair	$\frac{12\binom{4}{2}\binom{44}{1}}{\binom{50}{3}}$	0.1616327	2
Three of a kind	$\frac{\binom{2}{1}\times(\binom{48}{2}-12\binom{4}{2})}{\binom{50}{3}}$	0.1077551	3
Full house	$\frac{12\binom{4}{3}+\binom{2}{1}12\binom{4}{2}}{\binom{50}{3}}$	0.0097959	10
Four of a kind	$\frac{\binom{2}{2}\binom{48}{1}}{\binom{50}{3}}$	0.0024490	40

Unpaired Hands:

End Hand	Calculation	Probability	Payout Multiplier
Two pair	$\frac{\binom{3}{1}\times\binom{3}{1}\times\binom{44}{1}+\binom{6}{1}\times 11\times\binom{4}{2}}{\binom{50}{3}}$	0.0404082	2
Three of a kind	$\frac{2\times\binom{3}{2}\times\binom{44}{1}+11\times\binom{4}{3}}{\binom{50}{3}}$	0.0157143	3
Full house	$\frac{2\times\binom{3}{2}\times\binom{3}{1}}{\binom{50}{3}}$	0.0001020	10
Four of a kind	$\frac{2\times\binom{3}{3}}{\binom{50}{3}}$	0.0009184	40

With those final hands covered, One pair holdings must be accounted for. The main distinction is between Middle and High card pairs, as low card pairs hold no value for payouts (when in one pair holdings).

Paired Hands:

End Hand	Probability	Payout Multiplier
High pair	$\frac{4}{13}$	1
Middle pair	$\frac{5}{13}$	0

Unpaired hands:

Starting Hand	Probability High Pair	Probability Mid Pair
2 High cards	$\frac{\binom{6}{1}\times\binom{44}{2}}{\binom{50}{3}}$	0
1 High, 1 Mid	$\frac{\binom{3}{1}\times\binom{44}{2}}{\binom{50}{3}}$	$\frac{\binom{3}{1}\times\binom{44}{2}}{\binom{50}{3}}$
1 High, 1 Low	$\frac{\binom{3}{1}\times\binom{44}{2}}{\binom{50}{3}}$	0
2 Middle cards	0	$\frac{\binom{6}{1}\times\binom{44}{2}}{\binom{50}{3}}$
1 Mid, 1 Low	0	$\frac{\binom{3}{1}\times\binom{44}{2}}{\binom{50}{3}}$

For any specific hands expected value, it is simply the probability times the payout multiplier listed above (Overall expected value for game is calculated in next section, Simulation).

Finally, we will cover Straights, Flushes, Straight flushes, and the Royal flush. These hands are all mutually exclusive with the hands above, and therefore will be calculated separately. We will also be adding 1 to the multipliers, the reasoning for which will be explained in the Simulation section of this paper. Straights can begin with the Ace through the 5 (aces can be used to complete the 5-4-3-2-A straight, but cannot be used as a lower card in any lower straights. It must be the first or last card), making there 10 different numeric values of straight. Flushes must be 5 cards of the same suit, 13 choose 5, and there are 4 suits total. Straight flushes must meet both of these conditions, and Royal flushes must meet both conditions and start with an Ace. Written out, this results in the following table and probabilities:

Hand	Calculation	Probability	Payout multiplier
Straight	$\frac{10\times\binom{4}{1}\times\binom{4}{1}\times\binom{4}{1}\times\binom{4}{1}\times\binom{4}{1}-40}{\binom{52}{5}}$	0.00392465	5
Flush	$\frac{4\times\binom{13}{5}}{\binom{52}{5}}$	0.00198079	7
Straight Flush	$\frac{4\times9\times\binom{1}{1}\times\binom{1}{1}\times\binom{1}{1}\times\binom{1}{1}\times\binom{1}{1}}{\binom{52}{5}}$	0.00001386	101
Royal Flush	$\frac{4}{\binom{52}{5}}$	.00000154	501

To find the total contribution to the overall game's expected value, we take the probability of each, multiply it by their payout, and add them together for the chart above, resulting in: 0.0356648

5 Simulation

To simulate a game and the subsequent multiplier, the code appended to the end of this paper can be used. This program will simulate a deck of 52 cards, force every different numerical starting hand, and then simulate a number of dealings of community cards, in this case one million, adding the reached multiplier to the total Expected value and count of cards dealt. At the end, the average expected value will be printed to the screen along with number of runouts dealt. However, this program does not account for flushes and straights, and will give a multiplier of -1. Since these hands will not occur at the same time as any pair holdings, we will simply add the expected value found for straights, flushes, straight flushes, and the royal flush found in the last section. These holdings were given an extra point to their multiplier to counteract the -1 multiplier applied to these holdings in the program.
After 169 million rounds of play and adding the expected value discussed above, the program resulted in a final expected value of -0.175724.

6 Conclusion

Over the course of this paper, the probabilities of each final holding was calculated based off of the player's starting hand and the overall expected value was found via simulation, resulting in an expected value of -0.175724.
In reality, Mississippi Stud has an expected value of about -0.05 (Source 1). The main source of this discrepancy is that this analysis did not account for player strategy, but only outcome of hands. Players can impact this by simply folding out hands that consist of low cards, placing higher bets on holdings with positive expected value, and decreasing bet size on marginal holdings. However, this will also be somewhat counteracted by holdings that the player bets on until the river, only to not make a paid hand, as most hands will. So while the expected value found in this paper is close to the real value, player choice brings the game much closer to an expected value of 0, making Mississippi Stud one of the best bets for the player in a casino.

7 Sources

Soruce 1: Michael Shackleford, https://wizardofodds.com/games/mississippi-stud/
Source 2: Massachuessetts gaming board, https://massgaming.com/wp-content/uploads/Rules-Mississippi-Stud-10-08-2020.pdf
Source 3: Vegas Aces, https://vegasaces.com/lobby/news/online-poker/history-of-mississippi-stud-when-was-it-created-is-it-the-same-as-texas-holdem/

8 code

A C file with this code will be attached for convenience.

```
// Oliver Reinke, 4/28/23

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double getExpectedValue(int* runout)
{
        int anyPair = 0;
        int i;
        int p1 = 0;
        int p1Val = 0;
        int k3 = 0;

        for (i = 0; i < 4; i++)
        {
                if (runout[i] == runout[i + 1])
                {
                        anyPair = 1;
                        break;
                }
        }

        if (!anyPair)
        {
                return -1.0;
        }

        for (i = 0; i < 4; i++)
        {
                if (runout[i] == runout [i + 1]) // pair
                {
```

```
                                if (i < 3 && runout[i] == runout[i + 2]) // trips
                                {
                                        if (i < 2 && runout[i] == runout[i + 3])
                                        {
                                                return 40; // quads
                                        }
                                        else if (p1)
                                        {
                                                return (10);
                                        }
                                        else
                                        {
                                                i += 2;
                                                k3 = 1;
                                        }
                                }
                                if (p1)
                                {
                                        return 2; // two pair, no trips
                                }
                                else
                                {
                                        p1 = 1;
                                        p1Val = runout[i];
                                }
                        }
                }

                if (!p1)
                {
                        printf ("!!!ERROR, HAND CHECK ENTERED W/O A PAIR!!!\n");
                }
                else if (k3)
                {
                        return 3;
                }
                else if (p1Val < 5)
                {
                        return -1;
                }
                else if (p1Val < 10)
                {
                        return 0;
                }
                else if (p1Val >= 10)
                {
```

```
                return 1;
        }
        else
        {
                printf("ERROR IN EVAL FUNCTION!\n");
                return -2000000000;
        }

}

void getUpdatedDeck(int *hand, int *curDeck)
{
        int i = 0;

        while (curDeck[i] != hand[0])
        {
                i++;
        }
        curDeck[i] = -1;
        i = 0;

        while (curDeck[i] != hand[1])
        {
                i++;
        }
        curDeck[i] = -1;
}

void sortRunout(int *runout)
{
        int i;
        int j;
        int temp;
        int n = 5;

        for (i = 0; i < n; i++)
        {
                for (j = i + 1; j < n; j++)
                {
                        if (runout[i] < runout[j])
                        {
                                temp = runout[j];
                                runout[j] = runout[i];
                                runout[i] = temp;
                        }
                }
        }
}

void getRunout(int *hand, int *runout, int *curDeck)
{
        int randNums[3] = {-1, -1, -1};
        int temp;
        int num = -1;
        int i;

        runout[0] = hand[0];
        runout[1] = hand[1];
        runout[2] = -1;
        runout[3] = -1;
        runout[4] = -1;

        for (i = 0; i < 3; i++)
        {
                while (runout[i + 2] == -1)
                {
                        temp = rand() % 52;
                        if (curDeck[temp] != -1 && temp != randNums[0] && temp !=
                        {
                                randNums[i] = temp;
                                runout[i + 2] = curDeck[randNums[i]];
                        }

                }
        }

}

void resetDeck (int *baseDeck, int *curDeck)
{
        int i;

        for (i = 0; i < 52; i++)
        {
                curDeck[i] = baseDeck[i];
        }

        return;
}

void printRunout(int *runout)
{
        int i;

        for (i = 0; i < 5; i++)
        {
                printf("%d ", runout[i]);
        }
        printf("\n");
}

int main(void)
{
        int baseDeck[52] = {1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
                         7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11,
                                                                          12, 12,
        int curDeck[52] = {1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5,
                         7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11,
                                                                          12, 12,
        int hand[2] = {1,1};
        int runout[5];
        double ev = 0;
        int count = 0;
        int i;

        srand(time(NULL));

        while (hand[0] <= 13)
        {
                getUpdatedDeck(hand, curDeck);
                while (hand[1] <=13)
                {
                        for (i = 0; i < 1000000; i++)
                        {
                                getRunout(hand, runout, curDeck);
                                sortRunout(runout);
                                // printRunout(runout);
                                ev += getExpectedValue(runout);
                                count++;

                        }
                        hand[1]++;
                }
                resetDeck(baseDeck, curDeck);
                hand[0]++;
                hand[1] = 1;
        }
        // printf("Cur Hand: %d, %d\n", hand[0], hand[1]);
        printf("Count: %d\n", count);
        printf("EV: %f\n", ev/count + 0.0356648);
        return 0;
}
```