ISYE-6644 Project
Christopher Dae-Hyun Kim
Group 89

# Blackjack Simulation

**Abstract**

The project simulated thirteen optimal strategies derived using analytical methods and reports the generalized Bernoulli distribution, win rates, and returns of said strategies. Average win, tie, and loss rates per strategy were determined to be 39.32%, 9.94%, and 50.74% respectively. Expected returns per strategy were determined to be -0.8999. Average win, tie, and loss rates while using all strategies were determined to be 32.17%, 9.80%, and 58.03% respectively. Expected returns while using all strategies were determined to be -0.3156.

## Background

Blackjack is a popular card game that has been evaluated with numerical and analytical methods (Baldwin et al.), as well simulated (Thorpe). Thorpe had been so successful in modeling Blackjack as a series of dependent events (Thorpe, p. 43-48) that casinos immediately changed the rules of the game to

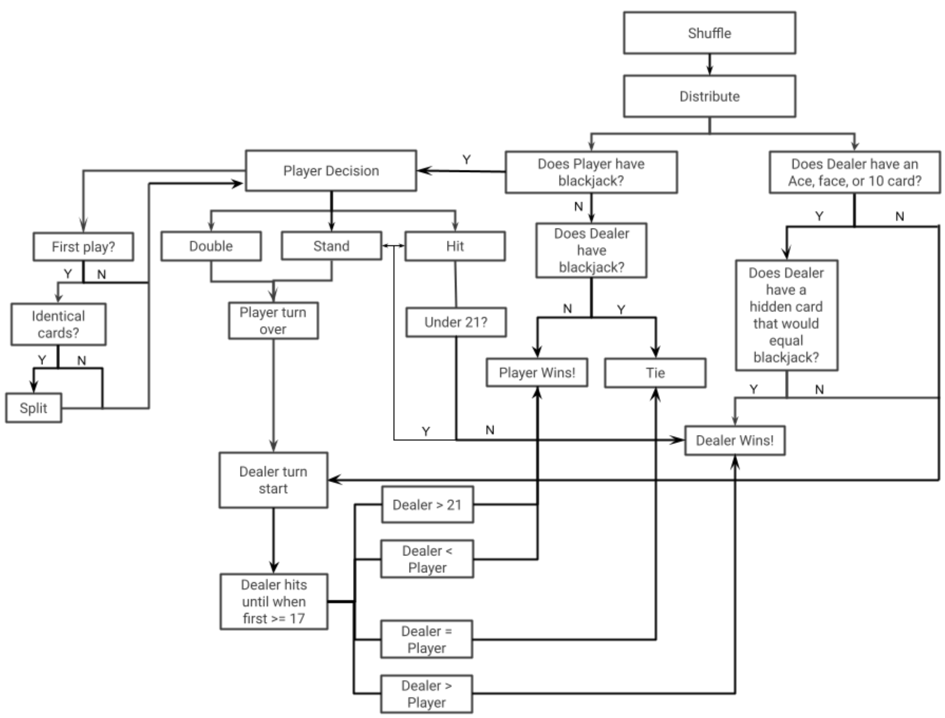


Figure 1. Schematic diagram of player decision with all result pathways.

the detriment of hotel and casino occupancy and visitors; casinos ultimately reverted back to the original rules (Colon). Thorpe's card counting strategy is still viable in modern times albeit frowned upon by the establishment owners. However, casino owners have increased the number of decks used during the game to have a quasi memoryless property.

## Methodology

In order to proceed in the simulation experiment, house rules and jargon must first be established and explained. The experiments will have the following structure:

1. The game will consist of eight decks.

2. Each game will be an independent trial where the cards played will return back to the deck and the deck reshuffled prior to playing another game.
3. There will be two entities in this simulation where one is the Dealer and the other is the Player.
4. The Player and Dealer will receive one card sequentially until each entity has a total of two cards with the Player receiving each card in sequence first.
5. If the first card the Dealer's receives has a value of 10 or 11, the Dealer will peek at the face down card;
    a. If the resultant total is 21, the Dealer automatically wins if the Player does not have Blackjack as well.
6. The Dealer must stand at 17 and continue drawing cards while the total is under 16;
    a. The Dealer must stand at Soft 17.
    b. If the Dealer draws an Ace, the Ace will be interpreted as 11 if the Dealer's total doesn't exceed 21.
    c. Each subsequent Ace will be interpreted as 1.
7. A hard value is the value of the cards in the Player's hand in which there is no Ace card.
8. A soft value when there is an Ace card in the Player's hand that is interpreted as 11.

This experiment will simulate and evaluate the strategies set out by Shackleford. The strategies are as followed:

1. Always hit hard 11 or less.
2. Stand on hard 12 against a dealer 4-6, otherwise hit.
3. Stand on hard 13-16 against a dealer 2-6, otherwise hit.
4. Always stand on hard 17 or more.
5. Always hit soft 17 or less.
6. Stand on soft 18 except hit against a Dealer 9, 10, or Ace.
7. Always stand on soft 19 or more.
8. Double hard 9 vs. dealer 3-6.
9. Double hard 10 except against a dealer 10 or A.
10. Double hard 11 except against a dealer A.
11. Double soft 13 or 14 vs. dealer 5-6.
12. Double soft 15 or 16 vs. dealer 4-6.
13. Double soft 17 or 18 vs. dealer 3-6.

In terms of scope, the project will be limited to only include the decision realm of hit, stand, and double. After 10,000 trials, the results will be reported. The distribution of strategies played will also be reported, i.e. how many times each strategy is played. Finally, the strategy that maximizes profits will be determined.

**Algorithm 1** Blackjack Simulator

```
1: D ≡ deck of 52 cards where 1, 2, 3, 4, 5, 6, 7, 8, 9, (10, J, Q, K) = 10, A = (1, 11)
2: procedure SIMULATE PLAYER STRATEGY
3:     Create game deck D* = 8D
4:     P0 ≡ Dealer hand
5:     P1 ≡ Player hand
6:     for i = 0, i < 10000, i++ do
7:         Shuffle(D*)
8:         for j = 0, i < 2, j++ do
9:             P1 ← card
10:            P0 ← card
11:        end for
12:        if P0 = {10, A} ∧ P1 = {10, A} then
13:            Continue
14:        else if P0 = {10, A} ∧ P1 ≠ {10, A} then
15:            Continue
16:        else if P0 ≠ {10, A} ∧ P1 = {10, A} then
17:            Continue
18:        end if
19:        while Total(P1) <= 21 do
20:            while Total(P1) ≤ Hard(11) do
21:                P1 ← card
22:            end while
23:            if Total(P1) = Hard(12) ∧ P0[a] ∈ {4, 5, 6} then
24:                Stand
25:            else
26:                P1 ← card
27:            end if
28:            if Total(P1) = Hard(13) ∧ P0[a] ∈ {2, 3, 4, 5, 6} then
29:                Stand
30:            else
31:                P1 ← card
32:            end if
33:            if Total(P1) ≥ Hard(17) then
34:                Stand
35:            end if
36:            if Soft(2) ≥ Total(P1) ≥ Soft(13) then
37:                P1 ← card
38:                Repeat Lines 12 - Lines 28
39:            end if
40:            if Total(P1) = Soft(18) ∧ P0[a] ∈ {9, 10, A} then
41:                P1 ← card
42:            else if Total(P1) = Soft(18) then
43:                Stand
44:            end if
45:            if Total(P1) = Soft(19) then
46:                Stand
47:            end if
48:        end while
49:        if Total(P1) > 21 then
50:            Continue
51:        end if
52:    end for
53: end procedure
```

Figure 2. Pseudocode for player strategies number one through seven. Final code will contain methods to collect information needed to analyze.

**Main Findings - Development**

The simulation was entirely coded using base Python modules and NumPy while utilizing Python version 3.9.7.

```
import numpy as np
import itertools
from collections import defaultdict, Counter
import matplotlib.pyplot as plt
```

Figure 3. All modules and packages used in coding the Monte Carlo analysis of Blackjack

As mentioned in the background and methodology, the reported simulation tries to recreate the quasi memoryless environment set by casinos by using an eight deck shoe. For the sake of programming simplicity, we report using the numerical values for all face cards while leaving the Ace card as a tuple to allow for situational value declarations and rule compliance.

```
def init_deck():
    deck = [2,3,4,5,6,7,8,9,10,10,10,10,(1,11)]
    deck = list(itertools.chain.from_iterable(itertools.repeat(x, 32) for x in deck))
    np.random.shuffle(deck)
    return deck

def distribute(deck):
    player = []
    dealer = []
    for i in range(2):
        player.append(deck.pop(0))
        dealer.append(deck.pop(0))
    return player,dealer

def hits(deck, player_total):
    hit = deck.pop(0)
    player_total[0].append(hit)
    if hit == (1,11):
        if player_total[1] + 11 > 21:
            player_total[1] += 1
            player_total[2] = True
            player_total[3] = False
        else:
            player_total[1] += 11
            player_total[2] = False
            player_total[3] = True
    else:
        player_total[1] += hit
```

Figure 4. A Python implementation of generating, distributing, and hitting from an eight deck shoe.

In order to comply with Rule 6, we needed to code checkpoints for the dealer and methods for the dealer while in play. These checkpoint methods were called after each decision by the dealer while the methods for dealer play are executed after the player is finished with their turn.

```
def dealerCheck(dealer):
    if any(isinstance(x, tuple) for x in dealer):
```

```
        if sum([isinstance(x,tuple) for x in dealer])==2:
            return [dealer, 12]
        else:
            total_max = np.sum([max(x) if isinstance (x, tuple) else x for x in\
                                   dealer])
            return [dealer, total_max]
    else:
        return [dealer, sum(dealer)]

def dealer_hits(deck, dealer_total):
    hit = deck.pop(0)
    if hit == (1,11):
        if dealer_total[1] + 11 > 21:
            dealer_total[1] += 1
        else:
            dealer_total[1] += 11
    else:
        dealer_total[1] += hit
    dealer_total[0].append(hit)

def dealer_turn(dealer, deck):
    dealer_total = dealerCheck(dealer)
    if dealer_total[1] == 17:
        return dealer_total

    while dealer_total[1] <= 16:
        dealer_hits(deck, dealer_total)

    return dealer_total
```

Figure 5. A Python implementation of Dealer rule compliance and decisions.

Similar checkpoints were needed for the player in order for the simulations to make rational decisions. An example of a rational decision would be that if a player has the following hand $\{(1, 11), 2, 10\}$, the simulation would continue instead of ending since a rational player wouldn't say that the value of their hand is 23; the rational player would always state that their hand valuation is a Hard 13. As implied in the example, we also needed to evaluate if the player's hand is considered soft or hard and implement a system that would allow for state changes. We also return a nested list of the Player's hand, the total, and booleans indicating the state of the hand.

```
def playerCheck(player):
    hard = False
    soft = False
    if any(isinstance(x, tuple) for x in player):
        if sum([isinstance(x,tuple) for x in player])==2:
            soft = True
            return [player, 2, hard, soft]
        else:
            total_max = np.sum([max(x) if isinstance (x, tuple) else x for x in
player])
            soft = True
            return [player, total_max, hard, soft]
```

```
    else:
        hard = True
        return [player, sum(player), hard, soft]

def softCheck(player_total):
    if (1,11) in player_total[0] and player_total[1]>21:
        player_total[1] = np.sum([min(x) if isinstance(x,tuple) else x for x in
player_total[0]])
        player_total[2] = True
        player_total[3] = False
```

Figure 6. A Python implementation of Player rational decision making and card state changes.

Final method needed was to check if either the Player or Dealer immediately drew Blackjack in the beginning of the round.

```
def bjCheck(player,dealer):
    playerWin = False
    dealerWin = False
    tie = False
    BJsets = [{10,(1,11)}]
    if (set(player) in BJsets) & (set(dealer) in BJsets):
        tie = True
    elif (set(player) in BJsets) & (set(dealer) not in BJsets):
        playerWin = True
    elif (set(player) not in BJsets) & (set(dealer) in BJsets):
        dealerWin = True
    return playerWin, dealerWin, tie
```

Figure 7. A Python implementation of checking for Blackjack.

As seen in the pseudocode in Figure 2, logic behind Player strategies is quite straightforward. It is a series of iterative, conditional statements. Figure 8 is the full Python implementation while the entire code to run the simulation can be found in the Appendix.

```
def player_turn(player, dealer, deck):
    double = False
    strategies = []

    player_total = playerCheck(player)

    # Strategies from 1 to 13.
    while player_total[1] < 21:
        # Strategy 1, 8, 9, 10 grouped together
        while (player_total[1] <= 11) and (player_total[2] == True):
            # Strategy 8
            if ((player_total[1] == 9) and (player_total[2] == True)) and (dealer[0]
in {3,4,5,6}):
                hits(deck, player_total)
                double = True
                strategies.append(8)
                return player_total, double, strategies
            # Strategy 9
```

```
            elif ((player_total[1] == 10) and (player_total[2] == True)) and
(dealer[0] not in {10, (1,11)}):
                hits(deck, player_total)
                double = True
                strategies.append(9)
                return player_total, double, strategies
            # Strategy 10
            elif ((player_total[1] == 11) and (player_total[2] == True)) and
(dealer[0] !=(1,11)):
                hits(deck, player_total)
                double = True
                strategies.append(10)
                return player_total, double, strategies
            # Strategy 1
            else:
                hits(deck,player_total)
                softCheck(player_total)
                strategies.append(1)

        # Strategy 2
        if ((player_total[1] == 12) and (player_total[2] == True)):
            if dealer[0] in {4,5,6}:
                strategies.append(2)
                return player_total, double, strategies
            else:
                strategies.append(2)
                hits(deck, player_total)
                softCheck(player_total)

        # Strategy 3
        if ((player_total[1]>= 13 and player_total[1]<=16) and (player_total[2] ==
True)):
            if dealer[0] in {2,3,4,5,6}:
                strategies.append(3)
                return player_total, double, strategies
            elif dealer[0] not in {2,3,4,5,6}:
                strategies.append(3)
                hits(deck, player_total)
                softCheck(player_total)
        # Strategy 4
        if (player_total[1] >= 17) and (player_total[2] == True):
            strategies.append(4)
            return player_total, double, strategies

        # Strategy 5, 11, 12, 13(soft 17)
        if (player_total[1]<=17) and (player_total[3] == True):
            # Strategy 11
            if (player_total[1] in {13,14}) and (dealer[0] in {5,6}):
                hits(deck, player_total)
                double = True
                strategies.append(11)
                return player_total, double, strategies
            # Strategy 12
            elif (player_total[1] in {15,16}) and (dealer[0] in {4,5,6}):
                hits(deck, player_total)
```

```
            double = True
            strategies.append(12)
            return player_total, double, strategies
        #Strategy 13 (Soft 17)
        elif (player_total[1]==17) and (dealer[0] in {3,4,5,6}):
            hits(deck, player_total)
            double = True
            softCheck(player_total)
            strategies.append(13)
            return player_total, double, strategies
        # Strategy 5
        else:
            strategies.append(5)
            hits(deck, player_total)
            softCheck(player_total)


    # Strategy 6, 13(soft 18)
    if ((player_total[1] == 18) and (player_total[3] == True)):
        # Strategy 13
        if dealer[0] in {3,4,5,6}:
            hits(deck, player_total)
            double = True
            softCheck(player_total)
            strategies.append(13)
            return player_total, double, strategies
        # Strategy 6
        if dealer[0] in {9,10,(1,11)}:
            hits(deck, player_total)
            softCheck(player_total)
            strategies.append(6)
            return player_total, double, strategies
        else:
            strategies.append(6)
            return player_total, double, strategies

    #Strategy 7
    if (player_total[1] >= 19) and (player_total[3] == True):
        strategies.append(7)
        return player_total, double, strategies

else:
    return player_total, double, strategies
```

Figure 8. Python implementation of Player's strategies.

**Main Findings - Statistical Analysis**

After 10,000 samples, we report the generalized Bernoulli distribution per strategy in Figure 9. Since Blackjack is a sequence of events with probabilistic transitions and rewards, we counted each event within the sequence as an occurrence in the derivation of the probability mass function where the

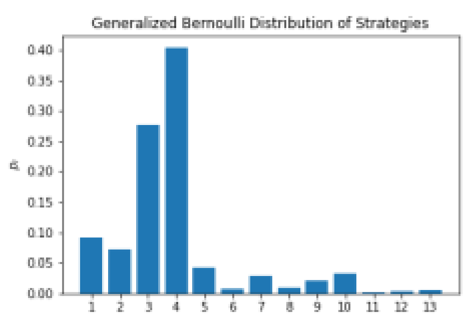


Figure 9. Distribution of strategies, a categorical variable.

probability mass function is defined as: $f(x = i \mid p) = p_i$ where $p = (p_1 \ldots p_k)$, and $\sum_{i=1}^{k} p_i = 1$.

Not surprisingly, strategies 1-4 have the highest probability to occur as a hard hand is more likely to occur than a soft hand as soft hands are contingent on the existence of an Ace card in the Player's hand. From an intuitive level, approximately 40% of all hands played with all strategies in mind will have the Player attempting to stand at a Hard 17.

Win, tie, and loss rates are calculated by dividing the total associated results of a sequence that a strategy appears in by the number of unique occurrences of a strategy for all sequences. For example, if a Player's final hand was [3, 5, 4, 7] and the Player beat the Dealer's hand of [7,10], then the win rate associated for Strategy 1 and 2 would be 100%. If the Player's next hand was [2,2,3,5,4] against the Dealer's hand of [6,10], the cumulative win rate associated for Strategy 1 and 2 would both be 50%. The win rate for the sequence of Strategy 1 and Strategy 2 would be 50% as well.


Figure 10. Charts for win, tie, and loss rates per strategy.

| | Probability | Win rates | Loss rates | Tie rates | Returns |
|---|---|---|---|---|---|
| (4,) | 0.269643 | 0.250820 | 0.602459 | 0.146721 | -856 |
| (3, 4) | 0.160460 | 0.629477 | 0.277548 | 0.092975 | 511 |
| (3, 3, 4) | 0.022654 | 0.619512 | 0.312195 | 0.068293 | 63 |
| (1, 5, 3, 4) | 0.000442 | 0.250000 | 0.750000 | 0.000000 | -2 |
| (9,) | 0.028401 | 0.214008 | 0.712062 | 0.073930 | -256 |
| (2, 3, 4) | 0.014919 | 0.637037 | 0.237037 | 0.125926 | 54 |
| (2, 4) | 0.053155 | 0.488565 | 0.415800 | 0.095634 | 35 |
| (5, 5, 3, 4) | 0.001216 | 0.727273 | 0.000000 | 0.272727 | 8 |
| (11,) | 0.003868 | 0.342857 | 0.600000 | 0.057143 | -18 |
| (6, 3, 4) | 0.014808 | 0.671642 | 0.246269 | 0.082090 | 57 |
| (1, 3, 3, 4) | 0.003315 | 0.633333 | 0.300000 | 0.066667 | 10 |
| (8,) | 0.013261 | 0.133333 | 0.808333 | 0.058333 | -162 |
| (1, 3, 4) | 0.029174 | 0.598485 | 0.287879 | 0.113636 | 82 |
| (7,) | 0.025970 | 0.348936 | 0.493617 | 0.157447 | -34 |
| (10,) | 0.044756 | 0.241975 | 0.666667 | 0.091358 | -344 |
| (2, 3, 3, 4) | 0.001768 | 0.875000 | 0.062500 | 0.062500 | 13 |
| (12,) | 0.009636 | 0.450980 | 0.372549 | 0.176471 | 8 |
| (5, 2, 4) | 0.003426 | 0.387097 | 0.483871 | 0.129032 | -3 |
| (1, 4) | 0.043225 | 0.178571 | 0.637363 | 0.184066 | -167 |
| (5, 7) | 0.010167 | 0.456522 | 0.423913 | 0.119565 | 3 |
| (1, 2, 4) | 0.007183 | 0.538462 | 0.369231 | 0.092308 | 11 |
| (1, 7) | 0.005304 | 0.395833 | 0.500000 | 0.104167 | -5 |
| (1, 2, 3, 4) | 0.002210 | 0.750000 | 0.200000 | 0.050000 | 11 |
| (5, 6) | 0.002763 | 0.360000 | 0.480000 | 0.160000 | -3 |
| (1, 1, 3, 4) | 0.002763 | 0.640000 | 0.280000 | 0.080000 | 9 |
| (1, 1, 4) | 0.005304 | 0.395833 | 0.416667 | 0.187500 | -1 |
| (1, 10) | 0.008620 | 0.269231 | 0.692308 | 0.038462 | -66 |
| (5, 5, 5, 13) | 0.000111 | 1.000000 | 0.000000 | 0.000000 | 2 |
| (3, 3, 3, 4) | 0.001437 | 0.769231 | 0.153846 | 0.076923 | 8 |
| (5, 3, 3, 4) | 0.002984 | 0.703704 | 0.222222 | 0.074074 | 13 |

| | Probability | Win rates | Loss rates | Tie rates | Returns |
|---|---|---|---|---|---|
| (5, 5, 7) | 0.001768 | 0.500000 | 0.312500 | 0.187500 | 3 |
| (5, 5) | 0.000663 | 0.833333 | 0.166667 | 0.000000 | 4 |
| (6,) | 0.006741 | 0.163934 | 0.688525 | 0.147541 | -32 |
| (1, 1, 2, 4) | 0.000332 | 0.333333 | 0.333333 | 0.333333 | 0 |
| (1, 12) | 0.000332 | 1.000000 | 0.000000 | 0.000000 | 6 |
| (1, 9) | 0.005857 | 0.226415 | 0.716981 | 0.056604 | -52 |
| (1, 1, 7) | 0.000995 | 0.333333 | 0.444444 | 0.222222 | -1 |
| (1, 5, 2, 4) | 0.000111 | 1.000000 | 0.000000 | 0.000000 | 1 |
| (5, 12) | 0.000111 | 1.000000 | 0.000000 | 0.000000 | 2 |
| (1, 1, 3, 3, 4) | 0.000332 | 0.333333 | 0.333333 | 0.333333 | 0 |
| (5, 5, 6) | 0.001105 | 0.900000 | 0.100000 | 0.000000 | 8 |
| (6, 6, 10) | 0.000221 | 1.000000 | 0.000000 | 0.000000 | 4 |
| (5, 5, 3, 4) | 0.000221 | 1.000000 | 0.000000 | 0.000000 | 2 |
| (1, 8) | 0.001768 | 0.250000 | 0.687500 | 0.062500 | -14 |
| (5, 4) | 0.005304 | 0.020833 | 0.854167 | 0.125000 | -40 |
| (13,) | 0.008962 | 0.095238 | 0.809524 | 0.095238 | -90 |
| (1, 13) | 0.001105 | 0.100000 | 0.900000 | 0.000000 | -95 |
| (1, 1, 2, 3, 4) | 0.000111 | 1.000000 | 0.000000 | 0.000000 | 1 |
| (5, 5, 4) | 0.000553 | 0.400000 | 0.600000 | 0.000000 | -1 |
| (5, 2, 3, 4) | 0.000553 | 0.400000 | 0.200000 | 0.400000 | 1 |
| (5, 5, 13) | 0.000442 | 0.250000 | 0.750000 | 0.000000 | -4 |
| (5, 5, 5, 5, 7) | 0.000553 | 0.400000 | 0.200000 | 0.400000 | 1 |
| (3,) | 0.122223 | 0.000000 | 1.000000 | 0.000000 | -1106 |
| (2,) | 0.022102 | 0.000000 | 1.000000 | 0.000000 | -200 |
| (5, 3) | 0.004862 | 0.000000 | 1.000000 | 0.000000 | -44 |
| (1, 3) | 0.013151 | 0.000000 | 1.000000 | 0.000000 | -119 |
| (1, 2) | 0.002542 | 0.000000 | 1.000000 | 0.000000 | -23 |
| (1, 1, 3) | 0.000774 | 0.000000 | 1.000000 | 0.000000 | -7 |
| (1, 6) | 0.001216 | 0.000000 | 1.000000 | 0.000000 | -11 |
| (5, 6, 4) | 0.000884 | 0.000000 | 0.875000 | 0.125000 | -7 |
| (5, 10) | 0.001105 | 0.000000 | 0.900000 | 0.100000 | -10 |

| | Probability | Win rates | Loss rates | Tie rates | Returns |
|---|---|---|---|---|---|
| (2, 3) | 0.004531 | 0.000000 | 1.000000 | 0.000000 | -41 |
| (1, 5, 4) | 0.000442 | 0.000000 | 0.750000 | 0.250000 | -3 |
| (5, 5, 3) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (1, 5, 3) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (1, 5, 6) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (1, 1, 10) | 0.000332 | 0.000000 | 1.000000 | 0.000000 | -6 |
| (1, 2, 3) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (5, 2, 3) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (5, 5, 2, 3, 4) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (2, 5, 5, 6) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (5, 3, 2, 3, 4) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (5, 5, 5, 7) | 0.000442 | 0.000000 | 0.500000 | 0.500000 | -2 |
| (2, 3, 3, 3, 4) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (1, 5, 7) | 0.000442 | 0.000000 | 0.750000 | 0.250000 | -3 |
| (5, 11) | 0.000221 | 0.000000 | 1.000000 | 0.000000 | -4 |
| (2, 5, 5, 5) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (5, 5, 11) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (1, 3, 2, 3, 4) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (1, 1, 3, 3, 3, 4) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (5, 5, 2, 3) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (5, 2) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -1 |
| (1, 1, 13) | 0.000111 | 0.000000 | 1.000000 | 0.000000 | -2 |
| (3, 3, 3, 3, 4) | 0.000111 | 0.000000 | 0.000000 | 1.000000 | 0 |
| (5, 3, 3, 3, 4) | 0.000111 | 0.000000 | 0.000000 | 1.000000 | 0 |

Figure 11. Table of sequence strategies associated probability, win rates, loss rates, tie rates, and returns.

With the exception of Strategy 12, none of the analytically optimal strategies are associated with a positive win rate as seen in Figure 10. Moreover, average win, tie, and loss rates per strategy were determined to be 39.32%, 9.94%, and 50.74% respectively while average win, tie, and loss rates using sequential strategies were 32.17%,  9.80%, and 58.03% respectively. However, expected return for single strategies was-0.8999 whereas expected return for all sequential strategies was -0.3156.

**Conclusions**

The first point to address is the relatively low win rates for the two cases. Although the house edge is typically a couple of percentages greater, the cause for the rather large spread as seen from the reported simulation is due to the fact that the strategy of splitting has not been incorporated into this simulation. However, it is hard to believe that one single strategy would ensure a fair game.

An interesting extension of this simulation would be to implement reinforcement learning and learn optimal policies. However, judging from the rather pessimistic single strategy win rates, optimal policies may still end up with a negative return.

The key take away from this is that the house must have a competitive advantage in order to create a profitable and sustainable business model. In order for the player to have any advantage would be to move the framework of Blackjack as a series of independent events to a series of dependent events, a feature that Thorpe had already exploited and Las Vegas had already patched.

# Appendix

## [1] Blackjack Simulation

```
blackjack_dict = defaultdict(int)
strategy_count_dict = defaultdict(int)
strategy_single_count_dict = defaultdict(int)
strategy_wins_dict = defaultdict(int)
strategy_loss_dict = defaultdict(int)
strategy_ties_dict = defaultdict(int)
strategy_returns_dict = defaultdict(int)
strategy_single_wins_dict = defaultdict(int)
strategy_single_loss_dict = defaultdict(int)
strategy_single_ties_dict = defaultdict(int)
strategy_single_returns_dict = defaultdict(int)
np.random.seed(0)
for i in range(10000):
    deck = init_deck()
    player, dealer = distribute(deck)
    (pWin, dWin, tie) = bjCheck(player, dealer)
    if True in (pWin, dWin, tie):
        idx = [i for i,x in enumerate((pWin, dWin, tie)) if x][0]
        blackjack_dict[idx] += 1
    else:
        player_go = player_turn(player, dealer, deck)
        dealer_go = dealer_turn(dealer, deck)
        strat = player_go[2]
        p_total = player_go[0][1]
        d_total = dealer_go[1]
        if d_total > p_total:
            if player_go[1] == True:
                strategy_returns_dict[tuple(strat)] -= 2
            else:
                strategy_returns_dict[tuple(strat)] -= 1
            strategy_loss_dict[tuple(strat)] += 1
        elif d_total < p_total:
            if player_go[1] == True:
                strategy_returns_dict[tuple(strat)] += 2
            else:
                strategy_returns_dict[tuple(strat)] += 1
            strategy_wins_dict[tuple(strat)] += 1
        else:
            strategy_ties_dict[tuple(strat)] += 1
        strategy_count_dict[tuple(strat)]+=1

        for j in strat:
            strategy_single_count_dict[j] += 1
            if d_total > p_total:
                strategy_single_loss_dict[j] += 1
            elif d_total < p_total:
                strategy_single_wins_dict[j] += 1
            else:
                strategy_single_ties_dict[j] += 1
        if player_go[1] == True:
            Strategy_returns_dict
```

References

Baldwin, R. R., Cantey, W. E., Maisel, H., & McDermott, J. P. (1956). The Optimum Strategy in Blackjack. *Journal of the American Statistical Association*, *51*(275), 429–439. https://doi.org/10.1080/01621459.1956.10501334

Colon, N. G. (2017, September 25). *Ed Thorp: The Man Who Changed Casino Gaming*. Gambling Insider. https://www.gamblinginsider.com/in-depth/4202/ed-thorp-the-man-who-changed-casino-gaming

Shackleford, M. (n.d.). *4-Deck to 8-Deck Blackjack Strategy*. Wizard of Odds. https://wizardofodds.com/games/blackjack/strategy/4-decks/

Thorp, E. O. (1966). *Beat the Dealer*. Vintage Books.