7

5

GAME NIGHT EDITION

BLACK JACK

GROUP 20

Ng Wei Hao 40160493

Quek Kai Ling, Jacinta 40160665

Tan Hong Han 40160745

TEAM CONTRIBUTION

Strategy Assignment and Contribution

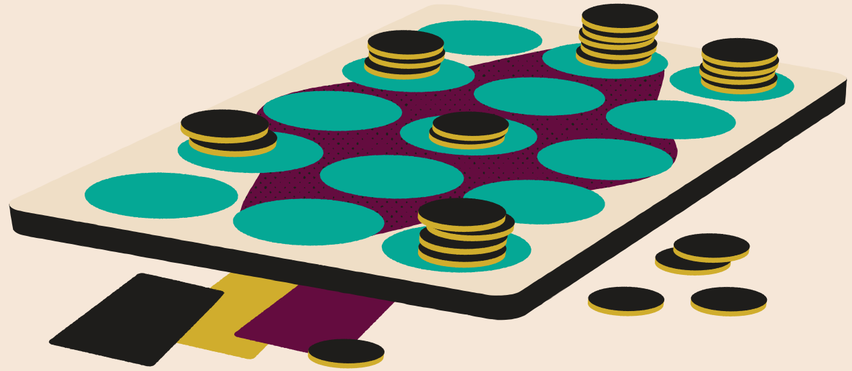
Player Profile	Member
1. Average Rational Player	Hong Han
2. The Overly Conservative	Hong Han
3. The Overly Aggressive	Hong Han
4. The Betting God	Hong Han
5. The Card Counter	Jacinta
6. The Monte Carlo	Ryan

TITLE:
SIMULATION OF BLACK JACK GAMES TO FIND THE MOST OPTIMAL STRATEGY

advisory: this is just a game simulation and does not depict real life.

MOTIVATION

- Explore **different** blackjack **strategies** and **‘best practices’**
- Crucial to find **the optimal strategy** to increase probability of winning



PROCESS DESCRIPTION

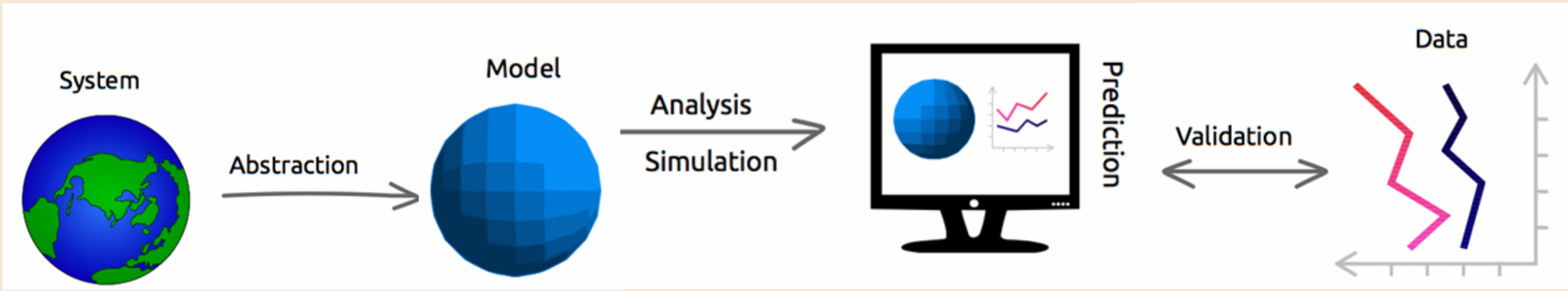
1 DEFINE GAME, HOUSE RULES & ASSUMPTIONS

2 DEVELOP PLAYER MODELS

3 SIMULATE BLACKJACK ROUNDS

4 ANALYSE & COMPARE PERFORMANCE

5 DRAW CONCLUSIONS & NEXT STEPS



DEFINE GAME & HOUSE RULES

1

Limited decks at home

All games are played with 1 deck

1

2

Simple even a baby can play

No Split
No Double

2

3

Survival 1v1

Each player plays only with the dealer

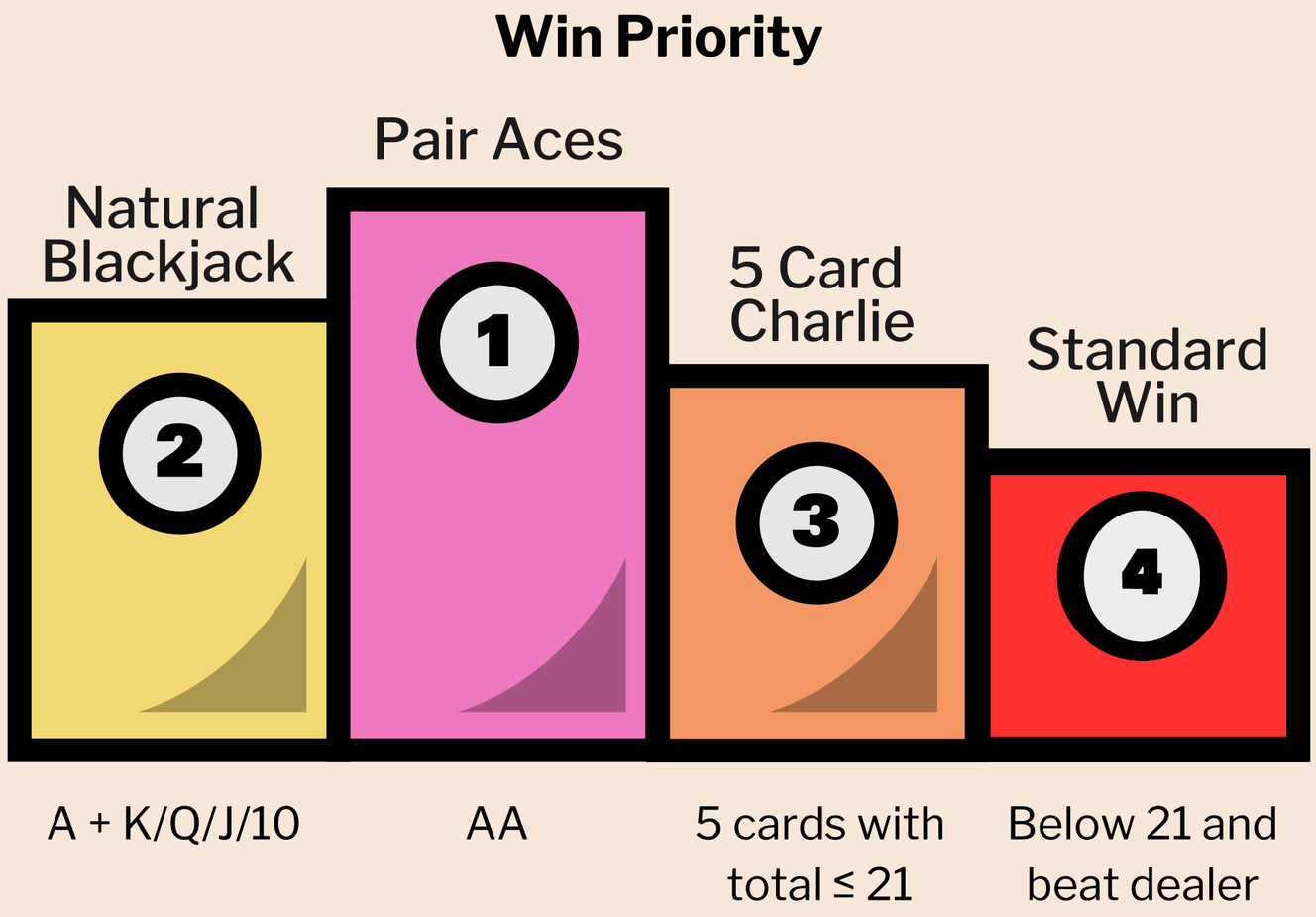
3

DEFINE GAME & HOUSE RULES

4

No Ambiguity

Established Win priority with minimum card value of 16



ASSUMPTIONS

1

For **decision focused strategies**,

Bet size is fixed at $10

Unless informed by strategy (card counting)

1

2

For **bet sizing strategies**,

Players are rational hence assumes the risk profile of the avg rational player.

Unless informed by strategy (card counting)

2

3

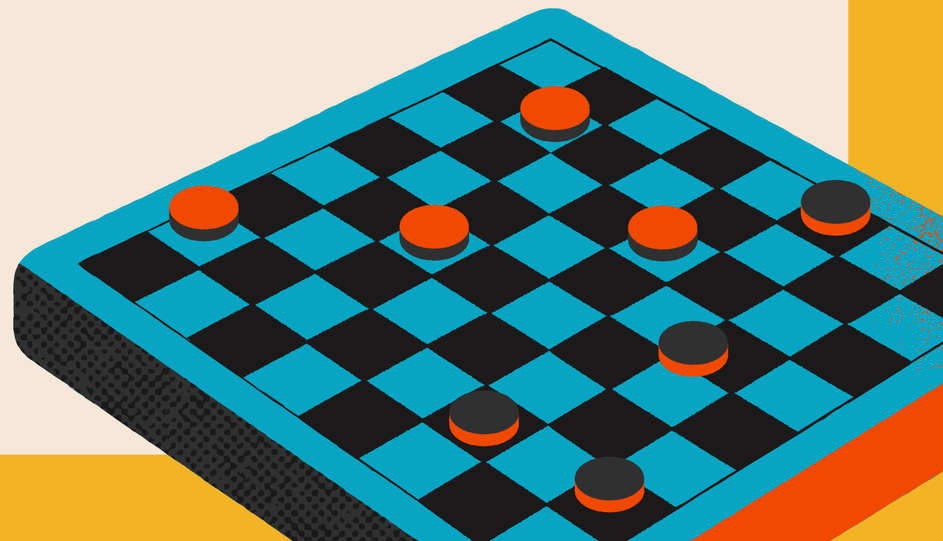
Born into a rich family

All players have unlimited money

3

DEVELOP PLAYER MODELS

6 Player (Strategy) Models



1

The average rational player

1

2

The overly conservative

2

3

The overly Aggressive

3

4

The Betting God ‘赌神’

4

5

The Card Counter

5

6

The Monte Carlo

9

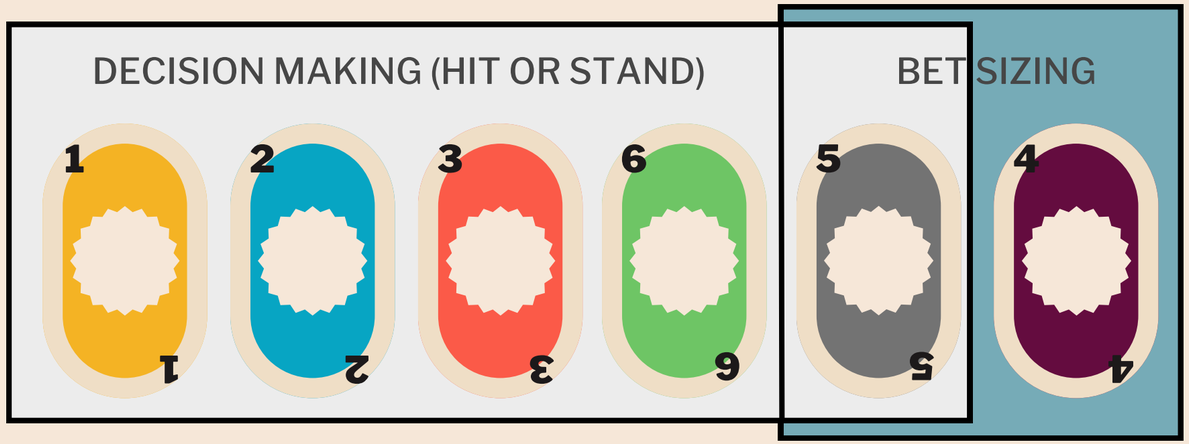
THE PLAYER MODELS

Player Profile	Strategy Type	Decision Making (hit or stand)	Bet Sizing
1. Average Rational Player	Basic Strategy	✅ Yes	❌ No
2. The Overly Conservative	Heuristic	✅ Yes	❌ No
3. The Overly Aggressive	Heuristic	✅ Yes	❌ No
4. The Betting God	Basic Strategy + Martingale Strategy	❌ No	✅ Yes (Doubles after loss)
5. The Card Counter	Basic Strategy + Adjustments from Count	✅ Yes	✅ Yes
6. The Monte Carlo	Monte Carlo (Decision-Making Focus)	✅ Yes	❌ No

Average Rational Player is the control of the simulation

METHODOLOGY

For **each player profile**:
  For **each round in 300 games**:
  # Decision Making (hit or stand)
    if player **profile >= 1 and <= 3**:
      constant **bet amount at $10**
      simulate each of their strategies
  # Betting Strategy
    if player **profile == 4**:
      implement double bet after loss
      implement basic strategy for decision making
      simulate strategy 4
    if player **profile == 5**:
      implement high-low counting system
      implement scaling betting logic
      simulate strategy 5
    if player **profile == 6**:
      implement **First- Visit Monte Carlo**
      constant **bet amount at $10**
      simulate strategy 6



- Performance Metrics of each player strategy

Winning percentage against House

Average profit per round

PREVIEW OF INDIVIDUAL STRATEGY ANALYSIS

Part 1: 🔣 Pseudo code
Part 2: 🧠 Strategy logic
Part 3: 📈 Performance insights
Part 4: 💬 Reflections & Challenges

PLAYERS PROFILES 1-4

```
IMPORT necessary libraries

DEFINE three blackjack strategies as DataFrames:
    - Basic (standard strategy)
    - Conservative (stands more)
    - Aggressive (hits more)

CLASS Player:
    PROPERTIES: hand, points, money, bet, stats
    METHODS: update hand value

CLASS Deck:
    PROPERTIES: cards
    METHODS: shuffle, deal card

FUNCTION is_blackjack(hand)
    RETURN true if Ace + 10-value card

FUNCTION evaluate_hand_quality(hand)
    RETURN quality score based on hand composition
```

```
FUNCTION play_hand(player, dealer, strategy, deck):
    IF either has blackjack:
        RESOLVE with tiebreakers
    WHILE player wants to hit:
        DRAW card based on strategy
    DEALER plays according to house rules
    COMPARE hands with tiebreakers
    UPDATE player stats and money

FUNCTION run_simulation(strategy):
    INITIALIZE player and deck
    PLAY sample hands for display
    RUN multiple simulations
    CALCULATE statistics
    PLOT results

MAIN:
    SET random seed
    RUN simulations for all strategies
    DISPLAY comparison results
```

1. THE AVERAGE RATIONAL PLAYER

```
FUNCTION play_hand_basic_strategy(player_hand, dealer_upcard):
    WHILE player_hand.total < 21 AND player_hand.cards < 5:
        IF player_hand.total <= 15:
            HIT
        ELSE IF player_hand.total == 16-17:
            IF dealer_upcard in [2,3,4,5,6]:
                STAND
            ELSE:
                HIT
        ELSE IF player_hand.total >= 18:
            STAND

    // Dealer plays according to house rules
    dealer_hand = play_dealer_hand(dealer_hand)

    RETURN compare_hands(player_hand, dealer_hand)
```

	2	3	4	5	6	7	8	9	10	A
3	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H
13	H	H	H	H	H	H	H	H	H	H
14	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17	S	S	S	S	S	H	H	H	H	H
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S
21	S	S	S	S	S	S	S	S	S	S

1. THE AVERAGE RATIONAL PLAYER

Hand #1 (Bet: $10)

Player: 3, 9 (Initial)

Dealer Upcard: 3

→ Hit: 6 (Total: 18)

→ Stand at 18

→ Dealer hits: 4 (Total: 17)

Final Hands:

Player: 3, 9, 6 = 18

Dealer: 3, 10, 4 = 17

Outcome: Win $10

Stack Change: $10

Hand #3 (Bet: $10)

Player: 10, 3 (Initial)

Dealer Upcard: 4

→ Hit: 4 (Total: 17)

→ Stand at 17

→ Dealer hits: 2 (Total: 16)

→ Dealer hits: 8 (Total: 24)

Final Hands:

Player: 10, 3, 4 = 17

Dealer: 4, 10, 2, 8 = 24

Outcome: Win $10 (Dealer Bust)

Stack Change: $10

Hand #7 (Bet: $10)

Player: 4, 6 (Initial)

Dealer Upcard: 6

→ Hit: 5 (Total: 15)

→ Hit: 9 (Total: 24)

→ Dealer hits: 10 (Total: 23)

Final Hands:

Player: 4, 6, 5, 9 = 24

Dealer: 6, 7, 10 = 23

Outcome: Push (Equal quality both bust)

Stack Change: $0

2. THE OVERLY CONSERVATIVE

```
FUNCTION play_hand_conservative_strategy(player_hand, dealer_upcard):
    WHILE player_hand.total < 21 AND player_hand.cards < 5:
        IF player_hand.total < 16:
            HIT
        ELSE:
            STAND

    // Dealer plays according to house rules
    dealer_hand = play_dealer_hand(dealer_hand)

    RETURN compare_hands(player_hand, dealer_hand)
```

	2	3	4	5	6	7	8	9	10	A
3	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H
13	H	H	H	H	H	H	H	H	H	H
14	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H
16	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S
21	S	S	S	S	S	S	S	S	S	S

2. THE OVERLY CONSERVATIVE

Hand #1 (Bet: $10)

Player: 10, 3 (Initial)

Dealer Upcard: 2

→ Hit: 7 (Total: 20)

→ Stand at 20

→ Dealer hits: 10 (Total: 20)

Final Hands:

Player: 10, 3, 7 = 20

Dealer: 2, 8, 10 = 20

Outcome: Push (Equal quality)

Stack Change: $0

Hand #5 (Bet: $10)

Player: 2, 10 (Initial)

Dealer Upcard: 10

→ Hit: A (Total: 13)

→ Hit: 2 (Total: 15)

→ Hit: 9 (Total: 24)

Final Hands:

Player: 2, 10, A, 2, 9 = 34

Dealer: 10, 10 = 20

Outcome: Lose (Player Bust)

Stack Change: $-10

Hand #8 (Bet: $10)

Player: 10, A (Initial)

Dealer Upcard: 10

Final Hands:

Player: 10, A = 21

Dealer: 10, 6 = 16

Outcome: Player Blackjack! Win $10

Stack Change: $10

3. THE OVERLY AGGRESSIVE

```
FUNCTION play_hand_aggressive_strategy(player_hand, dealer_upcard):
    WHILE player_hand.total < 21 AND player_hand.cards < 5:
        IF player_hand.total <= 19:
            HIT
        ELSE IF player_hand.total == 20:
            IF dealer_upcard in [A]:
                STAND
            ELSE:
                HIT
        ELSE:
            STAND

    // Dealer plays according to house rules
    dealer_hand = play_dealer_hand(dealer_hand)

    RETURN compare_hands(player_hand, dealer_hand)
```

	2	3	4	5	6	7	8	9	10	A
3	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H
13	H	H	H	H	H	H	H	H	H	H
14	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H
16	H	H	H	H	H	H	H	H	H	H
17	H	H	H	H	H	H	H	H	H	H
18	H	H	H	H	H	H	H	H	H	H
19	H	H	H	H	H	H	H	H	H	H
20	H	H	H	H	H	H	H	H	H	S
21	S	S	S	S	S	S	S	S	S	S

3. THE OVERLY AGGRESSIVE

Hand #1 (Bet: $10)

Player: 10, 10 (Initial)

Dealer Upcard: 10

→ Stand at 20

Final Hands:

Player: 10, 10 = 20

Dealer: 10, 7 = 17

Outcome: Win $10

Stack Change: $10

Hand #4 (Bet: $10)

Player: 3, 7 (Initial)

Dealer Upcard: 5

→ Hit: 5 (Total: 15)

→ Hit: 3 (Total: 18)

→ Hit: 10 (Total: 28)

→ Dealer hits: 7 (Total: 17)

Final Hands:

Player: 3, 7, 5, 3, 10 = 28

Dealer: 5, 5, 7 = 17

Outcome: Lose (Player Bust)

Stack Change: $-10

Hand #9 (Bet: $10)

Player: 10, 9 (Initial)

Dealer Upcard: 8

→ Hit: 6 (Total: 25)

→ Dealer hits: 9 (Total: 23)

Final Hands:

Player: 10, 9, 6 = 25

Dealer: 8, 6, 9 = 23

Outcome: Push (Equal quality both bust)

Stack Change: $0

4. THE BETTING GOD ‘赌神’

```
FUNCTION play_hand_betting_god(player_hand, dealer_upcard):
    // Uses Basic Strategy for gameplay
    outcome = play_hand_basic_strategy(player_hand, dealer_upcard)

    // Special betting rules
    IF outcome == WIN:
        current_bet = initial_bet
        consecutive_losses = 0
    ELSE IF outcome == LOSE:
        consecutive_losses += 1
        current_bet = 2 * initial_bet
    ELSE: // Push
        consecutive_losses = 0

    RETURN outcome
```

	2	3	4	5	6	7	8	9	10	A
3	H	H	H	H	H	H	H	H	H	H
4	H	H	H	H	H	H	H	H	H	H
5	H	H	H	H	H	H	H	H	H	H
6	H	H	H	H	H	H	H	H	H	H
7	H	H	H	H	H	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H
9	H	H	H	H	H	H	H	H	H	H
10	H	H	H	H	H	H	H	H	H	H
11	H	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H
13	H	H	H	H	H	H	H	H	H	H
14	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17	S	S	S	S	S	H	H	H	H	H
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S
21	S	S	S	S	S	S	S	S	S	S

4. THE BETTING GOD ‘赌神’

Hand #1 (Bet: $10)

Player: 6, 10 (Initial)

Dealer Upcard: 5

→ Stand at 16

→ Dealer hits: 7 (Total: 15)

→ Dealer hits: 3 (Total: 18)

Final Hands:

Player: 6, 10 = 16

Dealer: 5, 3, 7, 3 = 18

Outcome: Lose $10

Stack Change: $-10

Hand #2 (Bet: $20)

Player: 4, A (Initial)

Dealer Upcard: 9

→ Hit: 10 (Total: 15)

→ Hit: 10 (Total: 25)

→ Dealer hits: 9 (Total: 22)

Final Hands:

Player: 4, A, 10, 10 = 35

Dealer: 9, 4, 9 = 22

Outcome: Push $0 (Equal quality both bust)

Stack Change: $0

Hand #3 (Bet: $20)

Player: 8, A (Initial)

Dealer Upcard: 10

→ Stand at 19

Final Hands:

Player: 8, A = 19

Dealer: 10, 10 = 20

Outcome: Lose $20

Stack Change: $-20

4. THE BETTING GOD ‘赌神’

Hand #4 (Bet: $40)

Player: 6, 5 (Initial)

Dealer Upcard: 10

→ Hit: 8 (Total: 19)

→ Stand at 19

Final Hands:

Player: 6, 5, 8 = 19

Dealer: 10, 9 = 19

Outcome: Push $0 (Equal quality)

Stack Change: $0

Hand #5 (Bet: $40)

Player: 3, 5 (Initial)

Dealer Upcard: A

→ Hit: 10 (Total: 18)

→ Stand at 18

→ Dealer hits: 10 (Total: 13)

→ Dealer hits: 10 (Total: 23)

Final Hands:

Player: 3, 5, 10 = 18

Dealer: A, 2, 10, 10 = 33

Outcome: Win $40 (Dealer Bust)

Hand #6 (Bet: $10)

Player: 9, 10 (Initial)

Dealer Upcard: 7

→ Stand at 19

→ Dealer hits: 3 (Total: 16)

→ Dealer hits: 10 (Total: 26)

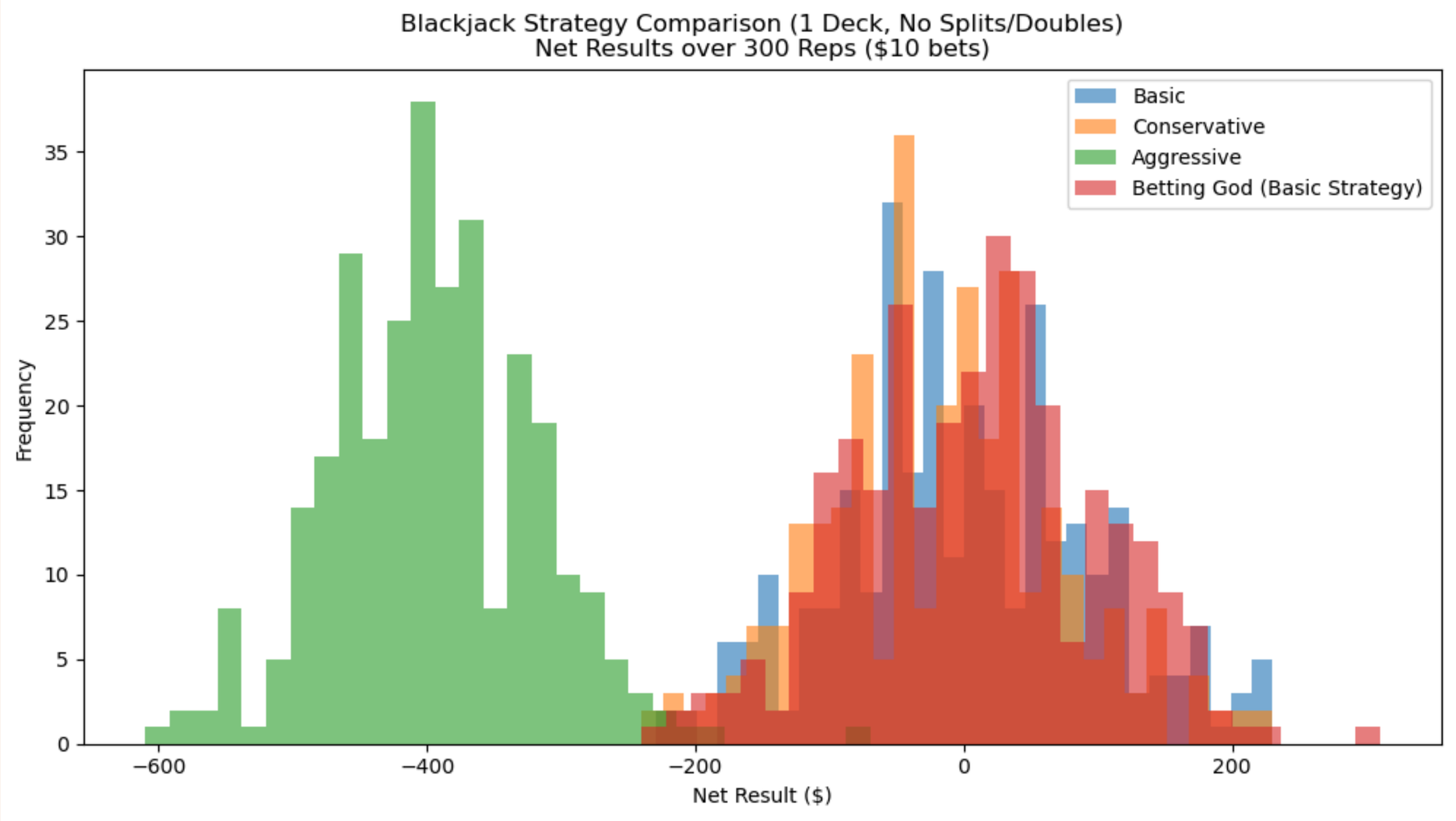
Final Hands:

Player: 9, 10 = 19

Dealer: 7, 6, 3, 10 = 26

Outcome: Win $10 (Dealer Bust)

SUMMARY OF FIRST 4 STRATEGIES



5. THE CARD COUNTER

HOW DOES CARD COUNTING WORK

1

Assign Hi-Lo values to cards

Low Cards (2-6):
+1
High Cards (10, J, Q, K,A):
-1
Neutral Cards (7-9):
0

1

2

Derivation of True Count

Calculation of the true count based on the Hi-Lo running count

$$\text{True count} = \frac{RunningCount}{No.\,of decks}$$

2

3

Make decisions

Players can use the true count to decide how much to bet

3

5. THE CARD COUNTER

Assigning Hi-lo value to cards

```
# Assign Hi-lo values to cards
def get_card_value(card):
    if card in [2, 3, 4, 5, 6]:
        return 1  # Low cards
    elif card in [10, 'J', 'Q', 'K', 'A']:
        return -1  # High cards
    else:
        return 0  # Neutral cards (7, 8, 9)
```

Definition of true count

```
def true_count(self):
    remaining_decks = max(len(self.cards) / 52, 0.5)  # Never less than 0.5 decks
    return self.running_count / remaining_decks
```

5. THE CARD COUNTER

Adjusting of bet based on true count

```
# Variable Betting Based on True Count
'''
Implementation of dynamic betting based on the true count and player stack
1. Higher bets and bets adjustments are placed when the true count indicates favourable odds
'''
def decide_variable_betting(true_count, player_stack, base_bet, min_bet = 0,max_bet = 20):
    max_bet = 1000
    # Adjust bet based on the true count
    if true_count > 2:
        bet = base_bet * 2  # Higher betting for favourable conditions
    elif true_count < -2:
        bet = base_bet / 2  # Minimise loss in unfavourable conditions
    else:
        bet = base_bet  # Default betting for neutral conditions

    # Ensure the bet is within valid boundaries and does not exceed player stack
    bet = round(bet)
    bet = max(min_bet, min(bet, max_bet, player_stack))
    return bet
```

Analysis of Results

Code logic

```
Game 273
  Player's initial hand: [10, 'A'] (Points: 21)
  Dealer's initial hand: [10, 5]
  True count: -10.399999999999999
  Bet placed: $5, Player's stack: $9985.0
  Player has blackjack! Wins $7.50
```

```
 Game 262
   Player's initial hand: ['A', 10] (Points: 21)
   Dealer's initial hand: ['A', 10]
   True count: -9.454545454545455
   Bet placed: $5, Player's stack: $10060.0
   Both player and dealer have blackjack - Push
```

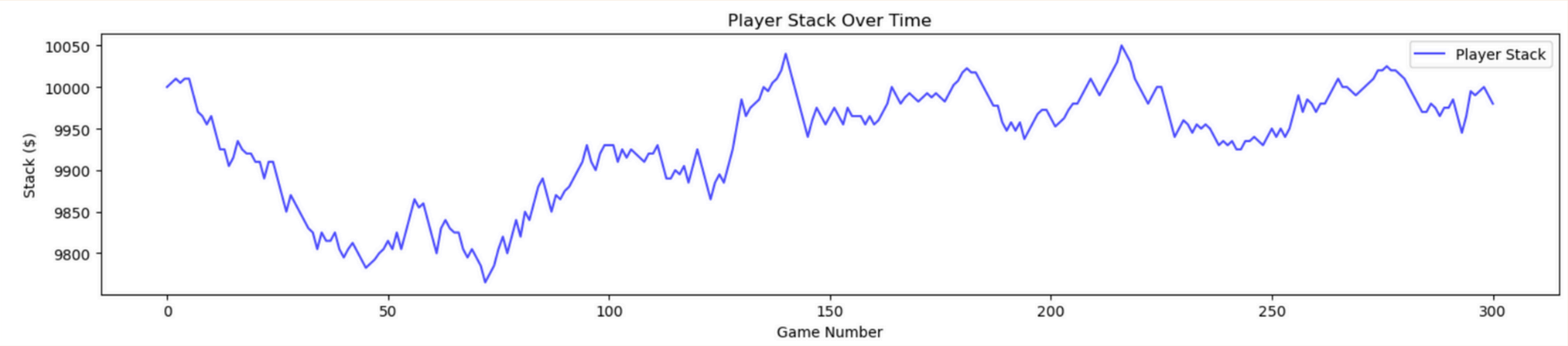
```
Game 276
  Player's initial hand: [5, 2] (Points: 7)
  Dealer's initial hand: [10, 10]
  True count: 0.0
  Bet placed: $10, Player's stack: $10055.0
Player draws a card: 10. New hand: [5, 2, 10] (Points: 17)
Player's final hand: [5, 2, 10] (Points: 17). Decision: S
  Player's chosen action: S
  Dealer's final hand: [10, 10] (Points: 20)
  Result: Dealer wins. Stack: $10045.0
  Cumulative Wins: 127, Losses: 121
```

5. THE CARD COUNTER

```python
def analyze_results(results):
    total_games = len(results["bets"])
    win_rate = (results["wins"] / total_games) * 100 if total_games > 0 else 0
    average_bet = sum(results["bets"]) / total_games if total_games > 0 else 0
    final_stack = results["player_stack"][-1]
    earnings = final_stack - 10000  # Assuming starting stack of $10,000
```

```
Game 300
  Player's initial hand: [2, 2] (Points: 4)
  Dealer's initial hand: [3, 9]
  True count: 1.7333333333333334
  Bet placed: $10, Player's stack: $9990.0
Player draws a card: 10. New hand: [2, 2, 10] (Points: 14)
Player draws a card: 10. New hand: [2, 2, 10, 10] (Points: 24)
Player busts with 24 points
  Player's chosen action: B
  Result: Player busts. Stack: $9980.0
Win Rate: 46.00%
Average Bet: $11.60
Final Stack: $9980.00
Earnings: $-20.00
```

5. THE CARD COUNTER



Over 300 games, the player wins about 46.00% of the games he plays and loses $0.0667 per game on average

5. THE CARD COUNTER

Advantages	Disadvantages
Variable betting helps to maximise profits	Variance and requires significant practice
Can be combined with other strategies (basic strategy + card counting)	

6. MONTE CARLO

6. THE MONTE CARLO

Method	Description	Advantages	Disadvantages
First Visit Monte Carlo	Only the first time a state-action pair appears per episode	✅ Avoids over-representing frequent states	❌ Can miss useful updates from repeated states
Every Visit Monte Carlo	Every time a state-action pair appears per episode	✅ Focuses on best actions	❌ Can overvalue frequently visited states
On Policy Monte Carlo	Learns the value of the policy being followed using samples from that same policy.	✅ Consistent with behavior	❌ Learns only about the current policy
Off Policy Monte Carlo	Learns value of a target policy using episodes generated by a different behavior policy (via importance sampling).	✅ Learns about any policy	❌ High variance due to importance sampling

6. THE MONTE CARLO

Pseudo code for the First-Visit Monte Carlo method

Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s
        Append G to $Returns(s)$
        $V(s) \leftarrow \text{average}(Returns(s))$

First-Visit Monte Carlo method in Blackjack Terms

Initialize:
    Policy ← hit below 16 and stand otherwise.
    Record cards drawn as state-value
    Create a list to record match history

Start simulating full games (episodes):
    State 1: player 6 vs dealer 17
    State 2: player 14 vs dealer 17
    State 3: player 18 vs dealer 17 (game ends)
    End of game: player wins and gets +1 reward

we won the game when **first state** is **(player 6 vs dealer 17)**.

6. THE MONTE CARLO

Training (500,000 episodes)

Game Mechanics

Your reward values (1.5, 1.3, 1.2, etc.) encode win priority and guide the agent during training.

```python
# Dealer bust
if self.dealer_sum > 21:
    # Checks for Win priority conditions
    if player_label == "Pair of Aces":
        return 1.5, self.game_state, True
    elif player_label == "Blackjack (Natural)":
        return 1.3, self.game_state, True
    elif player_label == "5-Card Charlie":
        return 1.2, self.game_state, True
    elif player_label == "Regular":
        return 1, self.game_state, True
    else:
        return -1, self.game_state, True  # bust
```

Outcome Incentives

Agent will prioritise winning > tie > losing.

```python
# model learning based off r
if reward == 0:
    tie += 1
elif reward == 1.5:
    win += 1
    blackjack_wins += 1
elif reward == 1:
    win += 1
elif reward == -1:
    loss += 1
```

Testing (300 episodes)

Performance Testing

$10 bet logic for Expected Return in testing uses.

```python
results_dict = {}
results_dict['Win Rate'] = win / n
results_dict['Tie Rate'] = tie / n
results_dict['Loss Rate'] = loss / n
results_dict['Expected Return'] = (
    results_dict['Win Rate'] * 10
    + results_dict['Loss Rate'] * -10
)

return results_dict
```

6. THE MONTE CARLO

Testing data

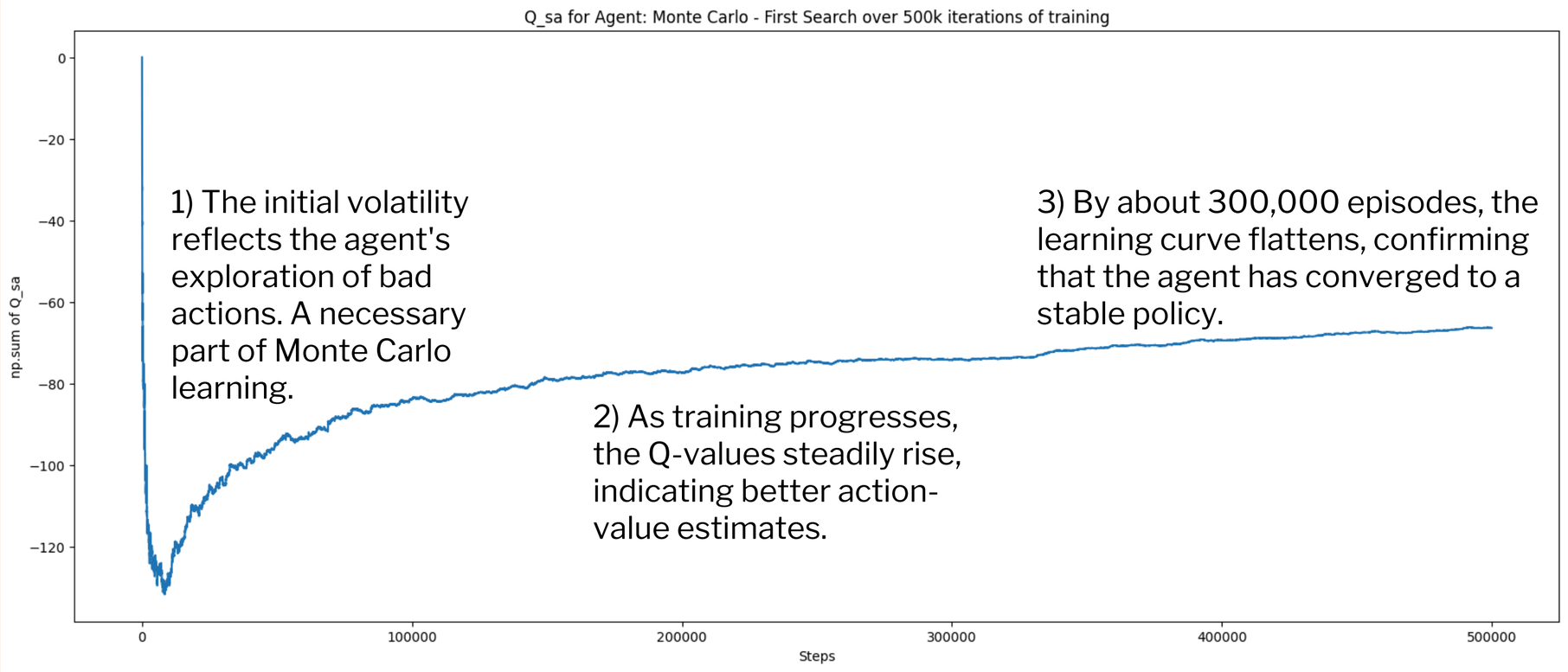
	Win Rate	Tie Rate	Loss Rate	Expected Return
Monte Carlo - First Search	**43.67%**	7.00%	46.67%	**-0.300000**
Monte Carlo - Every Search	38.67%	10.67%	44.00%	-0.533333
Monte Carlo - On Policy	38.67%	8.33%	47.00%	-0.833333
Monte Carlo - Off Policy	37.67%	7.33%	48.00%	-1.033333

Betting $10 over 300 games, MCFS agent wins 43.67% of the games he plays and loses $0.3 per game on average.

Best model may change run-to-run. And
Win rate ≠ best return, and vice versa.

6. THE MONTE CARLO

Convergence Plot of Monte Carlo First Search

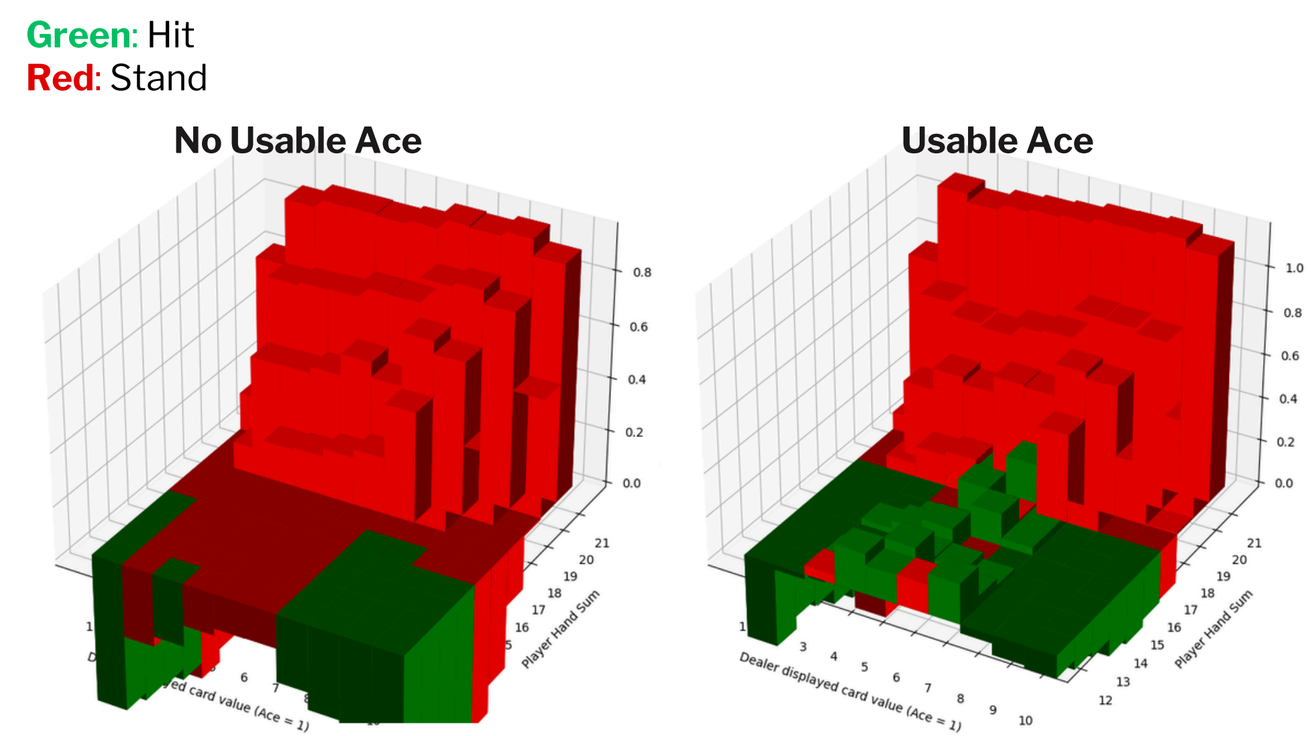


Hence we conclude that the MC First Search agent has **learned effectively** without needing additional heuristics or function approximation

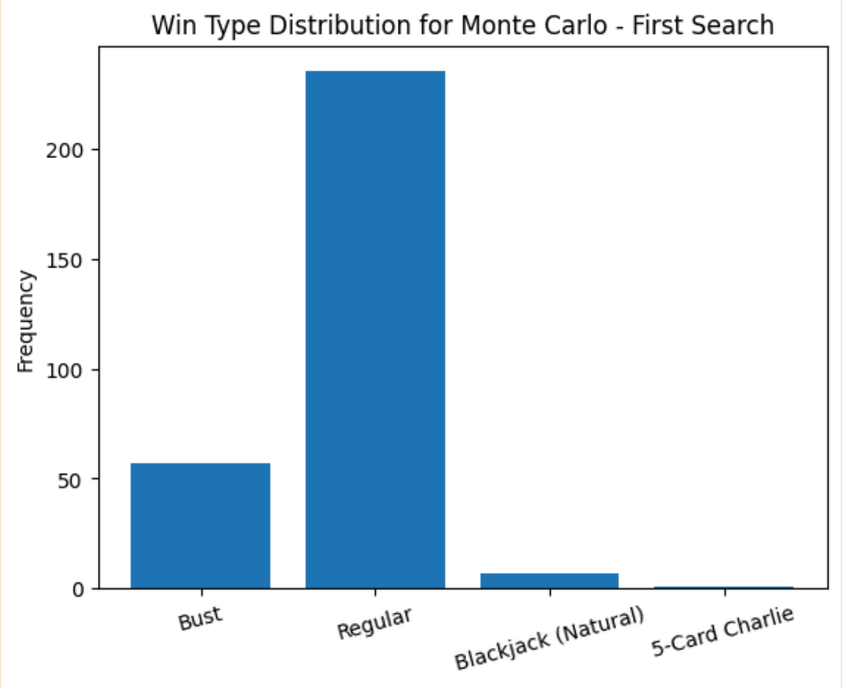
6. THE MONTE CARLO

Learned state-action values

Green: Hit
Red: Stand



Win type distribution reveals how it wins



ANALYSE & COMPARE PERFORMANCE

- Performance Metrics of each player strategy

Player Profile	Win Rate	Expected Return
1.Average Rational Player	42.10%	+ 0.002
2. The Overly Conservative	42.00%	-0.158
3. The Overly Aggressive	18.60%	-3.967
4. The Betting God	42.40%	+0.075
5. The Card Counter	46.00%	-0.0667
6. The Monte Carlo	43.67%	-0.300

DRAW CONCLUSIONS & NEXT STEPS

1. Flaws in Abstraction

- 1v1 vs multiple players
- Savvy play with Double and Splits
- Games with no betting

2. Limited by interpretability and application

- Real players often don’t stick to one policy
- Card Counting requires practice and focus over multiple rounds that is difficult to do casually
- Monte Carlo isn't intuitive to replicate during a quick round with family


Q&A
THANK YOU FOR LISTENING!