What would be the probability of "breaking the bank" in this 1985 Blackjack game?

Abstract

This study examines the probability and the expected number of hands required to "break the bank" in the 1985 Game and Watch title Blackjack, a game ending when a player’s total money either reaches zero or exceeds $9,999. This analysis is centered on the game’s unique mechanics and restrictions, particularly noting the absence of options such as splitting and insurance, which are commonly found in standard Blackjack games. The game is structured as a 1v1 matchup against the dealer, starting with a $500 balance and allowing a maximum bet of $100, which can be doubled. Notable rules include a single deck that is reshuffled after at least 12 cards are drawn, the dealer standing on soft 17, and special payouts where ties result in no change to the player's wallet except in the case of both player and dealer hitting Blackjack, where the player wins but receives only half the bet.

This Answers seeks to model the game as a random walk, considering each hand as an independent trial with associated probabilities of winning, losing, or tying, based on simplified Blackjack basic strategies. The theoretical framework employed considers the gambler's ruin problem to estimate the likelihood of achieving a wallet amount of $9,999 before bankruptcy,

alongside calculations to estimate the expected number of hands needed under these conditions.

APPROACH

1. **Basic Strategy**: Since splitting is not allowed in this version of Blackjack, we can focus on basic strategy without considering split hands. Basic strategy charts are available for games with similar rules, such as standing on soft 17 and doubling down allowed on any two cards.
2. **Deck Composition**: The probability of drawing specific cards from the deck affects the player's and dealer's chances of winning each hand. With only one deck in play, the composition changes with each hand, especially since the deck is reshuffled after a certain number of cards have been drawn.
3. **Simulation**: Given the complexity of the game and the various factors involved, simulating multiple rounds of play can provide insights into the likelihood of reaching the maximum wallet amount without losing. By running simulations, we can track the player's bankroll over multiple hands and determine the frequency with which the player reaches or exceeds the maximum amount.
4. **Monte Carlo Method**: Using the Monte Carlo method, we can simulate thousands or even millions of rounds of Blackjack based on the game rules and basic strategy. This approach involves randomly generating hands and outcomes according to the probabilities associated with each action (hit, stand, double down) and comparing the results to the player's bankroll.
5. **Analysis**: Once we have the results of the simulations, we can analyze the data to determine the probability of "breaking the bank" and estimate the average number of hands it takes for a player to reach the maximum wallet amount.

First Code: Runned 10000

```
import numpy as np

def simulate_blackjack(starting_funds, win_probability, target,
    funds = starting_funds
    hands_played = 0
    while funds > 0 and funds < target:
        if np.random.rand() <= win_probability:
            funds += bet_size
        else:
            funds -= bet_size
        hands_played += 1

        # Adjust bet size if it would result in negative funds
        if funds < bet_size:
            bet_size = funds

    return funds >= target, hands_played

# Simulation parameters
win_probability = 0.49  # Assuming a win rate close to neutral g
starting_funds = 500
target = 9999
bet_size = 100
trials = 10000

results = [simulate_blackjack(starting_funds, win_probability, t
successes = sum(result[0] for result in results)
average_hands = sum(result[1] for result in results) / trials

print(f"Probability of breaking the bank: {successes / trials:.2
print(f"Average number of hands played: {average_hands:.1f}")
```

```
Results:

Probability of breaking the bank: 0.46%
Average number of hands played: 226.7
```

SECOND CODE (To achieve more accuracy and consistency)

1. **Maintain Bet Limits**: Keep the bet within the allowed maximum of $100, doubling to $200 when favorable.
2. **Improved Decision Logic**: Consider when doubling might be beneficial. However, a precise calculation would require a more sophisticated model of card probabilities, which could be complex given frequent reshuffles.
3. **Increase Trials for Robustness**: Running more trials can improve the statistical robustness of our results.

```
import numpy as np

def simulate_blackjack(starting_funds, win_probability, target)
    funds = starting_funds
    hands_played = 0
    bet_size = 100  # Standard bet size

    while funds > 0 and funds < target:
        if funds < bet_size:
            bet_size = funds  # Adjust bet size if funds are lov

        # Decision to double down (simplified assumption: double
        if np.random.rand() < 0.1 and funds >= bet_size * 2:
            current_bet = bet_size * 2
        else:
            current_bet = bet_size

        if np.random.rand() <= win_probability:
            funds += current_bet
        else:
            funds -= current_bet
        hands_played += 1

    return funds >= target, hands_played

# Simulation parameters
win_probability = 0.49  # A rough estimate for a basic strategy
```

```
starting_funds = 500
target = 9999
trials = 10000

results = [simulate_blackjack(starting_funds, win_probability, t
successes = sum(result[0] for result in results)
average_hands = sum(result[1] for result in results) / trials

success_rate = successes / trials
success_rate, average_hands
```

```
Results:

Probability of breaking the bank:  0.58%
Average number of hands played: 201 hands
```

- **Probability of breaking the bank**: Approximately 0.58%. This is a slight increase from the previous simulation, but it still indicates that achieving the maximum wallet amount of $9,999 starting from $500 is quite rare under these game conditions.
- **Average number of hands played**: About 201 hands. This suggests that, on average, each game lasts for around 201 hands before either going broke or reaching the target amount.

THIRD CODE (ANALYSIS)

```
import numpy as np

def simulate_blackjack(starting_funds, win_probability, target,
    funds = starting_funds
    hands_played = 0
    while 0 < funds < target:
        bet_size = 100  # Standard bet size
        # Ensure sufficient funds for doubling down
        if np.random.rand() < double_down_probability and funds
            bet_size = 200
        # Simulate the hand
        if np.random.rand() <= win_probability:
            funds += bet_size
        else:
            funds -= bet_size

        hands_played += 1

    return funds >= target, hands_played

# Simulation parameters
win_probability = 0.49
double_down_probability = 0.1
starting_funds = 500
target = 9999
trials = 10000

results = [simulate_blackjack(starting_funds, win_probability, t
successes = sum(result[0] for result in results)
average_hands = sum(result[1] for result in results) / trials

# Output the results
success_rate = successes / trials
print(f"Success rate: {success_rate}, Average hands played: {ave
```

```
Results:

Success rate: 0.0075
Average hands played: 197.4562
```

FOURTH (FINAL)

```
import numpy as np

def simulate_blackjack():
    goal = 9999
    start_funds = 500
    max_bet = 200  # Considering doubling

    # Simulate multiple scenarios
    num_simulations = 10000
    num_successes = 0
    average_hands = 0

    for _ in range(num_simulations):
        wallet = start_funds
        hands_count = 0
        deck = create_deck()
        while wallet > 0 and wallet < goal:
            if len(deck) < 20:  # Reshuffle if the deck has less
                deck = create_deck()

            bet = min(max_bet, wallet)
            player_cards, dealer_cards, deck = deal_initial_card

            # Player decision: basic strategy (simplified)
            player_total = calculate_total(player_cards)
            dealer_upcard = dealer_cards[0]
            while player_total < 17 or (player_total < 18 and de
                if len(deck) < 1:  # Check to avoid running out
                    deck = create_deck()
                player_cards.append(deck.pop())
                player_total = calculate_total(player_cards)
                if player_total > 21:
                    wallet -= bet  # Player busts
                    break
            if player_total <= 21:
                # Dealer plays
                while calculate_total(dealer_cards) < 17 or (cal
                    if len(deck) < 1:  # Check to avoid running
                        deck = create_deck()
                    dealer_cards.append(deck.pop())

                dealer_total = calculate_total(dealer_cards)
                if dealer_total > 21 or player_total > dealer_to
                    wallet += bet
                elif player_total < dealer_total:
                    wallet -= bet

            hands_count += 1

        if wallet >= goal:
            num_successes += 1
        average_hands += hands_count

    average_hands /= num_simulations
    probability_of_success = num_successes / num_simulations

    return probability_of_success, average_hands

def create_deck():
    """Creates a shuffled deck."""
    deck = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J',
    np.random.shuffle(deck)
    return deck

def deal_initial_cards(deck):
    """Deals two cards to the player and two to the dealer from
    return [deck.pop(), deck.pop()], [deck.pop(), deck.pop()], d

def calculate_total(cards):
    """Calculates the total of the given hand of cards."""
    total = 0
    ace_count = 0
    for card in cards:
        if card in 'JQK':
            total += 10
        elif card == 'A':
            ace_count += 1
        else:
            total += int(card)

    # Add aces last because their value depends on the current t
    for _ in range(ace_count):
        if total + 11 > 21:
            total += 1
        else:
            total += 11

    return total

# Running the simulation
probability, avg_hands = simulate_blackjack()
print(f"Probability of reaching $9,999: {probability * 100:.2f}%
print(f"Average hands played per simulation: {avg_hands:.0f}")
```

```
Results:

Probability of reaching $9,999: 0.00%
Average hands played per simulation: 26
```

Analysis

First Code

- **Simulation Parameters**: Win probability set at 0.49, with a simple win/lose scenario for each bet.
- **Results**: Probability of reaching $9,999 is 0.46%, and the average number of hands played is 226.7.

Interpretation: This simulation suggests that reaching the $9,999 mark is a rare event, with less than half a percent chance. The number of hands required to reach this scenario indicates a prolonged game session, considering the high variance and the slim margin of winning each hand.

Second Code

- **Adjustments**: Bet doubling considered more strategically, although still simplified.
- **Results**: Probability increased slightly to 0.58%, with fewer average hands played (201).

Interpretation: The introduction of more strategic betting (doubling down) seems to have a modest effect on improving the chances of winning. However, the results still highlight the difficulty of achieving the $9,999 goal under the game conditions.

Third Code

- **Incorporated Double Down Probability**: Factored in the likelihood of doubling the bet based on a favorable scenario.
- **Results**: Success rate significantly drops to 0.0075, with average hands decreasing slightly to 197.4562.

Interpretation: This version shows a drastically lower success rate, which may be due to the riskier strategy of doubling down not paying off as expected. This highlights the high-risk nature of aggressive betting strategies in this game setting.

Fourth Code

- **Detailed Game Mechanics**: Uses a more detailed simulation involving deck management and basic Blackjack strategies.
- **Results**: Probability of success fell to 0.00%, with an average of only 26 hands played.

Interpretation: The detailed simulation, which likely provides the most accurate reflection of the game's dynamics, shows almost no chance of breaking the bank, with players likely going bust quickly. This result underscores the challenge posed by the game's structure and rules.

Overall Insights and Recommendations

- **Game Complexity and Strategy**: Your analyses demonstrate that "breaking the bank" in this particular version of Blackjack is exceptionally challenging. This is largely due to the restrictive game rules, such as the lack of splitting and the limitations on doubling down.
- **Risk Management**: The simulations show that more conservative betting strategies prolong the game but do not necessarily increase the probability of reaching the target significantly. Aggressive strategies (e.g., frequent doubling down) may lead to quicker losses.
- **Recommendation for Further Analysis**: To enhance these simulations, consider adjusting the win probability dynamically based on the actual deck composition and a more accurate implementation of the Basic Strategy. This requires a complex model but could provide more precise insights.
- **Use in Video Project**: These results could be very informative for your video project, highlighting how game design and rule variations significantly affect player strategy and outcomes in gambling scenarios. You can discuss the implications of game rules on strategy formulation and risk assessment in gambling games.

Differences in Each Simulation Analysis

First Code

Key Features:

- **Simple Betting Model**: Fixed bet size with no consideration for bet adjustment based on game conditions.
- **Assumed Win Probability**: A constant probability of 0.49, close to a fair game, was used for all decisions.

Impact on Results:

- This approach resulted in a moderate number of hands played and a slightly higher success rate than might be realistic, given the simplicity of the betting strategy and no adaptation to the game state.

Second Code

Key Features:

- **Introduction of Bet Doubling**: Bet doubling was considered under favorable conditions, though still a random chance (10%).
- **More Dynamic Betting Strategy**: Adjustments in bet size were based on the funds available, promoting a more realistic betting strategy.

Impact on Results:

- The introduction of a more dynamic betting strategy allowed for slightly better capitalization on favorable situations, thus slightly increasing the success rate and reducing the average number of hands played compared to the first code.

Third Code

Key Features:

- **Explicit Double Down Probability**: Introduced a probability for choosing to double down, attempting to simulate more strategic decision-making.
- **Risks Associated with Aggressive Strategies**: The increase in bet size when doubling down was more risky, which could lead to faster losses or gains.

Impact on Results:

- The increased risk from more frequent and probabilistic doubling down led to a higher rate of losing the initial stake before reaching the goal, thus a much lower success rate and a similar number of hands played compared to the second code.

Fourth Code

Key Features:

- **Detailed Game Mechanics**: Incorporated real Blackjack strategies such as basic strategy decision-making based on the dealer’s upcard and the player's hand total.
- **Deck Management**: Simulated more realistic game conditions with deck reshuffling and card depletion, affecting decision-making and bet strategy.

Impact on Results:

- This simulation showed the harshest reality, with nearly no successful scenarios. The detailed inclusion of realistic game mechanics and the challenges posed by the game rules led to quicker losses and a significantly lower number of hands played. This highlights the impact of stringent game constraints and more complex strategies on game outcomes.

Conclusion of Analysis Differences

- **From Simple to Complex**: As the simulations evolved from simple random outcomes to detailed strategy and deck management, the results became increasingly pessimistic regarding the probability of breaking the bank, indicating that more detailed simulations tend to predict outcomes more aligned with real-world challenges in Blackjack.
- **Strategy and Risk**: Implementing more sophisticated strategies, such as considering when to double down based on more than just random chance, can significantly impact the results, usually resulting in quicker game ends due to the higher risk of losing bets.
- **Game Mechanics and Player Decisions**: The fourth code's integration of detailed player decisions and deck reshuffling shows the crucial role these factors play in gambling games, dramatically affecting the expected outcomes.