Proceedings of the 2nd International Conference on Computing Innovation and Applied Physics (CONF-CIAP 2023)
DOI: 10.54254/2753-8818/5/20230516

Game theory--optimum strategy for drawing cards in 21 points

Chenyiteng Han[1,4,5], Ruoxun Peng[2], Hengjie Zhao[3]

[1] Art and Science, University of Toronto, St.George, Toronto, M5S1A1, Canada
[2] International Division, The Second High School Attached to Beijing Normal University, Beijing, 100192, China
[3] RDFZ Chaoyang Branch School, Beijing, 100028, China
[4] Corresponding author

[5] Chenyiteng.han@mail.utoronto.ca

Abstract. Blackjack is a popular casino poker game in which players make drawing decisions based on rules other than the size of their points. We found that despite the randomness of the card draw, blackjack is actually very mathematical and logical. By calculating probabilities and using game theory, we can allow players to make relatively "rational" decisions that maximize their expectations of winning, rather than relying solely on guesswork and vague feelings. Therefore, in this article, we built several models to simulate a blackjack game with different numbers of people. For players in each model, we give them a code that outputs the optimal decision, which helps them figure out the winning percentage (more specifically, the expectation, since once stakes are involved in multiplayer games, the winning percentage doesn't necessarily represent the expectation of winning money) for each decision, thus helping the player make a better decision.

Keywords: game theory, 21 points, probability distribution function, payoff matrix.

1. Introduction

Game theory is the study of mathematical model that evaluates strategic interaction between participants [1]. John von Neumann drew public attention in 1928. He looked game theory as an independent field and introduced the equilibrium strategy in two-players zero-sum game [2]. Basic concepts of game theory: players, actions, payoffs and information (PAPI for short), making up the game strategy evaluable. Observers design a situation based on the game rules. In this situation, players aim to maximize their payoffs, which is known as game strategy. Thus, observers are able to estimate the combination of all players' payoffs, which is called equilibrium. Thus, we are capable of gaining the possible outcomes [3].

21 points, in other words, blackjack, is a poker game. The game was invented in France in 1700, and considered as a legal gambling in Nevada, the United States in 1931.

Today, blackjack is a remarkably popular and straightforward card game that draws crowds of players in both traditional and online casinos. Blackjack is quite easy to understand, so everyone may enjoy playing game. However, mathematics and the odds at play account for the other half of its allure. All

DOI: 10.54254/2753-8818/5/20230516

casino games include a built-in advantage for the house, which means that on average, the house will win more money than the players, allowing the house to overall turn a profit. Every game, from poker to slots, is designed this way, but blackjack actually has the lowest edge of all casino games, making it a great choice for people who are looking for the highest chance of winning money [4].

In Las Vegas, Macao and other cities in the world famous for casinos and gambling, 21 points are one of the four fundamental gambling games: dice, poker and roulette. It has been said that 21 points were underestimated in scientific research. In fact, 21 points provide wide range of unexplored treasures in mathematical and statistical analysis. “Few experienced players recommend standing on a total of 13 under any circumstances, and standing on 12 would be completely out of the question.” Previous article written by Roger and his coworkers has defined D as dealer’s up card, D=2, 3…10 (1,11) and M(D) be the minimum standing number. If player’s total points are greater or equal to M(D), player should stand. And if total points are less than M(D), player should draw [5].

In this article, we aim to find the strategy in playing 21 points. We divide the work into parts; there are introduction, methodologies, results and conclusion.

Based on the basic rules of blackjack -- comparing the size within 21 points -- we simulated several different game scenarios by adding different additional rules: single-player, two-player, multi-player rotation. For single-player games, we calculate the payoff of a player's card draw by calculating the probability distribution function (PDF) of the number of cards in the player's hand, thereby advising the player on what to do. From there, we built code to help players make decisions in multiplayer games -- input a hand of cards and a few other conditions to get the theoretically optimal way to act.

2. Methodologies

2.1. Probability distribution function

In game theory, n-person game is when each player takes one strategy from finite set of pure strategies. And the probability distributions over pure strategies refer to mixed strategies, which payoffs are the strategies players take. Mixed-strategy becomes a form in probabilities that different players take different pure strategies [6].

Probability distribution function is a function representing the probabilities of possible outcomes of an event. To evaluate the event, all possible outcomes and corresponding frequencies of occurrence are needed. In 21 points, the probabilities of total points help to decide whether to draw.

2.2. Programming

Robert had used code to simulate the situation of Iterated Prisoner’s Dilemma. In his experiments, strategies sometimes favor the individual and sometimes the group. And computer helped to make decisions based on previous records, average scores were used to find out the winner of the game [7].

In this article, cards are selected randomly, so every point from 2 to 20 may obtain in the sum of the first two cards that players selected. Even the same value of total points, the sequence and combination of cards may be different. As huge number of calculations are needed, programming may help. Accessible software such as Python would be more straightforward, as there is no need to design a new algorithm [8]. In this article, as we input information such as cards in hand and the number of certain cards in the card pool, the computer gives the probabilities of points in total if drawn.

2.3. Basic rules

Only 52 poker cards are being used in the card pool, which is a full deck of poke cards without jokers. Players hold 2 randomly selected cards initially and decide whether to pick another card from the pool each turn. The decision depends on cards the player already has and the following rules.

2.3.1. Point rules

- ‘A’ counts for 1 point.
- ‘J’, ‘Q’, ’K’ count for 10 points each.

DOI: 10.54254/2753-8818/5/20230516

- cards count for the amount coincident with the number on the card.

2.3.2. Competing rules
- Players need to maximize the total point they hold below or equal to 21.
- Points over 21 means 0.

2.3.3. More rules. 'A' counts for 1 point or 11 points as the player wants.

2.4. 1-player-game
According to basic rules, we assume a 1-player-game. By examining the card in hand, we are able to calculate the PDF of total points the player holds after drawing, which is the foundation for making a drawing decision. Actually, a 1-player-game has nothing to do with 'win' or 'lose'. It is only an assumption helping us to do some basic probability calculations. But for further work, this calculation can be pretty important.

2.4.1. Codes. See Appendix 1.

2.5. 2-player-game with god's perspective
If we add a player to the game with some new rules below, then there is an outcome.

2.5.1. Drawing rules. All cards of each player are NOT shown. Although from God's perspective we can see the cards in the hands of two players, the decisions they make do not depend on the hands of their opponents.
- Both players draw cards together.
- Both players draw from the same deck of cards.
- Each player can only draw 1 card.

2.5.2. Competing rules. After the drawing turn, players show their cards and the one holding more points wins (points over 21 means 0).

2.5.3. Codes. See Appendix 2.

2.6. 2-player-game from player's perspective
Then we do things from one of the players' perspective.

2.6.1. Drawing rules.
- All cards of each player are NOT shown.
- Both players draw cards together.
- Both players draw from the same deck of cards.
- Each player can only draw 1 card.

2.6.2. Competing rules. After the drawing turn, players show their cards and the one holding more points wins (points over 21 means 0).

2.6.3. Codes. See Appendix 3.

2.7. 2+player-game with a dealer
What if we make one of these 3+ players be the dealer? This should be the most complicated problem we are about to solve, and should also be the case that is most close to the real 21-point-game in the casino.

DOI: 10.54254/2753-8818/5/20230516

2.7.1. Drawing rules

- The dealer has 2 initial cards with one shown, while others have 2 initial cards all shown.
- Loafers take turns to draw cards from the same 52-card-pool before the dealer does.
- Each player has only 1 turn to draw cards.
- Each player can only have 5 cards(containing the initial 2).
- All cards drawn by loafers are shown.
- Each player can keep asking for cards until he doesn't want or his total point exceeds 21.
- Dealer shows the hidden initial card before his drawing turn.

We need to think of more than 1 step in this game since the dealer actually has 3 chances to draw a card. That's why we set the restriction that everyone can only draw 3 extra cards. If they can draw more cards, the computation may increase exponentially, and even the result may not change. Actually, in most games no one will ask for 4 or more cards.

2.7.2. Competing rules

- Each loafer bets before all sessions begin.
- Each loafer only competes with the dealer. If he wins the dealer, then he wins the bet from the dealer. Otherwise, he loses the bet to the dealer. i.e. A tie means the dealer wins.

2.7.3. Codes. See Appendix 4.

3. Results

3.1. 1-Player-game

3.1.1. PDF. PDFs of total points the player will get after drawing according to the cards already in hand. It can be calculated as number of certain cards left in the pool divided by total cards left in the pool. For example, if cards in hand are [1,3,1], results see Table 1.

3.1.2. Simulation. As cards are selected randomly, it is necessary to simulate the situation. We input the cards already in player's hand and it returns the total points after drawing.

3.1.3. T-test. Do the T-test for every element in the result of test. It receives true.

3.2. 2-player-game with god's perspective

In this part, we assume that both players follow the strategy to draw when the probability to increase their points is larger than the probability to decrease their points. In fact, this strategy may not be the best strategy but it does make sense to some degrees.

$$\begin{bmatrix} p1q1 & p1q2 \\ p2q1 & p2q2 \end{bmatrix} \tag{1}$$

p1 represents the probability for player 1 to draw and p2 not to draw. Similarly, q1 means the probability for player 2 to draw and q2 not to draw.

It is a classic game-theory model, since we have both players' probability to draw and not to draw. Thus, we can set a matrix and then deduct who is having the advantage in a specific game by computing the payoff for both players. In fact, we only need payoff for one of them since it is a 0-sum-game.

Table 1. PDFs if draw.

Total Points	PDFs
0	0
1	0
0	0

DOI: 10.54254/2753-8818/5/20230516

Table 1. (continued).

1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0.08163265306122448
13	0.08163265306122448
14	0.08163265306122448
15	0.32653061224489793
16	0.04081632653061224
17	0.08163265306122448
18	0.061224489795918366
19	0.08163265306122448
20	0.08163265306122448
21	0.08163265306122448

Note: 0 represents points exceeding 21.

$$payoff = draw1 * draw2 * payoff11 + draw1(1 - draw2)payoff12 + (1 - draw1)draw2 * payoff21 + (1 - draw1)(1 - draw2)payoff22 \quad (2)$$

If the payoff for player 1 is positive, then player 1 is having the advantage. Oppositely, player 2 has the advantage when payoff for player1 is negative.

3.3. 2-Player-game from player's perspective

In This part, we assume that the opponent follows the strategy to draw when the probability to increase their points is larger than the probability to decrease their points (same as the strategy in last part). But for the player we are observing, strategy changes. We will go through all the cases of the opponent's two initial hands, separately discussing whether the opponent drew or not, and finally consider the final hand after the opponent drew or not. In each case, we will calculate our probability to win and sum it up by weighting the probability. In the end, we only draw cards if our win rate increases more than our lose rate. For example, if cards in hand are [1,7] and [1,6], results see Tables 2-4.

Table 2. Drawing decisions.

Cards in hand	Decision
1,7	not draw
1,6	draw

Table 3. Probability to win before and after drawing.

Probability to win before drawing	After drawing
0.6603741496598653	0.6167191344622874
0.6050000000000009	0.6058148791431358

DOI: 10.54254/2753-8818/5/20230516

Table 4. Probability to lose before and after drawing.

Probability to lose before drawing	After drawing
0.27819727891156476	0.3241963381097126
0.33806122448979625	0.33615356781010186

3.4. 2+Player-game with a dealer

The strategy is to draw when any of the 3 payoffs after drawing is larger than the initial payoff. Actually we've done something similar in the 3rd part, 2-player-game from player's perspective. When we compute the opponent's payoff, we need to compute the case that he draw 2 (initial cards) or 3 cards (initial cards plus one more card) at one time. When cards in hand for dealer are [1,3,6] and [1,2], results are shown in Tables 5-9.

Table 5. Cards in hand for dealer and loafers.

Cards in hand for dealer	Cards in hand for loafer 1
1, 3, 6	1, 6, 4
1, 2	1, 4

Table 6. Cards in hand for dealer and loafers.

Cards in hand for loafer 2	Cards in hand for loafer 3
10, 7	Loafer 1 and 2 only
10, 7	8, 9

Table 7. Drawing decisions for dealer.

Cards in hand	Drawing Decision
1, 3, 6	not draw
1, 2	draw

Table 8. Payoff for dealer if not draw and draw one card.

Payoff if not draw	Payoff if draw one card
-100	-154.54545454545456
-400	-27.272727272727273

Table 9. Payoff for dealer if draw two and three cards.

Payoff if draw two cards	Payoff if draw three cards
-236.09756097560998	-293.0061640587939
59.93031358885018	-233.2793522267198

The case that the payoff decreases after drawing once but increases even above the initial payoff after drawing twice may occur. So, if we only consider the payoff after drawing once, we may not make the best decision.

4. Conclusion

In the article, with different situations in 21 points, we have investigated and concluded corresponding solutions to win the game. In 2-Player-Game, we have calculated the payoffs and probability with both god and players' sides. In 2+Player-Game, we calculated dealer's payoffs. Those helped us to decide whether to draw card in order to gain the highest points but not exceed 21 to win the game. Thus, we gain the optimum strategies for drawing cards in 21 points.

However, there are limitations. This drawing strategies only succeed when players follow the rules in this article. When playing 21 points in actual casino, rules and conditions are more complicated.

DOI: 10.54254/2753-8818/5/20230516

Psychological strategies and environment also play a role in the game. If players have drawn one card larger than expected, some of them may not affected by the unexpected and draw another card according to the drawing strategies. But what if it happens twice or more, players probably plan to not draw another card in next turn, even if calculations tell them to draw. Emotions and stress make people do wrong decisions.

Acknowledgement
The authors are grateful to Ming Gu for his enlightenment and guide of our research and Kangning Cui for his instruction of paper writing.

Chenyiteng Han, Ruoxun Peng and Hengjie Zhao contributed equally to this work and should be considered co-first authors.

Appendix
1. 1-player-game

```
from random import randint
import math

def PDF_for_one(cards_in_hand: list[int]) -> dict[int: float]:
    """Return the probability distribution function(PDF) of the total points of the player after drawing
    when we know his hands.
    The variable cards_in_hand should be cards contained by a deck of poke without jokers."""

    CARD_POOL = {1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4, 10: 16}  # The general card pool.
    sum_in_hand = 0  # The point the player get.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the player.

    PDF = {}  # The PDF of the total points of the player after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 52 - len(cards_in_hand)  # The amount of cards in pool.

    for j in CARD_POOL:
        sum_after_draw = sum_in_hand + j
        if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1):
            PDF[sum_after_draw + 10] += CARD_POOL[j] / total_cards_in_pool  # 'A' counts for 11 or 1
as the player want.
        elif sum_after_draw <= 21:
            PDF[sum_after_draw] += CARD_POOL[j] / total_cards_in_pool
        else:
            PDF[0] += CARD_POOL[j] / total_cards_in_pool
    # Compute the PDF

    return PDF


def draw_card(cards_in_hand: list[int]) -> int:
```

DOI: 10.54254/2753-8818/5/20230516

```
    """Draw a card randomly according to the cards in the player's hand and return the total points after drawing."""

    CARD_POOL = {1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4,
                 10: 16}  # The general card pool.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
    draw = randint(0, 52 - len(cards_in_hand))
    for key in CARD_POOL:
        draw -= CARD_POOL[key]
        if draw < 0:
            CARD_POOL[key] -= 1
            cards_in_hand.append(key)
            break
    if sum(cards_in_hand) + 10 <= 21 and 1 in cards_in_hand:
        return sum(cards_in_hand) + 10
    if sum(cards_in_hand) > 21:
        return 0
    return sum(cards_in_hand)


def test(cards_in_hand: list[int], test_times: int) -> dict[int: int]:
    """Return the simulation for PDF_for_one.
    The variable cards_in_hand refers to the card in the player's hand.
    The variable test_times refers to the number of simulations."""

    result = {}
    i = 0

    for k in range(22):
        result[k] = 0
    while i < test_times:  # The number of simulations can be set at will.
        copy = []
        for e in cards_in_hand:
            copy.append(e)
        d = draw_card(copy)
        for key in result:
            if key == d:
                result[key] += 1
        i += 1
    return result


def t(sample: list[float], u: float) -> bool:
    """Do the T-test for sample and the expected value u.
    Return the result."""

    n = len(sample)  # The sample size
    x_bar = sum(sample) / n  # The average number of the sample.
    v = 0  # The variance.
```

DOI: 10.54254/2753-8818/5/20230516

```
    for x in sample:
        v += (x - u) * (x - u)
    s = math.sqrt(v)  # The standard deviation.
    if s == 0:
        return True
    z = abs(x_bar - u) * math.sqrt(n) / s
    if z <= 2.2622:
        return True
    else:
        return False


def t_test(cards_in_hand: list[int], sample_size: int) -> bool:
    """Do the T-test for every element in the result of test(cards_in_hand, 10000)
    with the result of PDF_for_one(cards_in_hand).
    Return the result of T-test."""

    sample = []
    expected = PDF_for_one(cards_in_hand)
    n = {}

    for i in range(sample_size):
        sample.append(test(cards_in_hand, 10000))
    for point in range(22):
        n[point] = []
        for x in sample:
            n[point].append(x[point] / 10000)
    for p in n:
        if not t(n[p], expected[p]):
            return False
    return True
```

2. 2-player-game with god's perspective

```
def PDF_for_one(cards_in_hand: list[int]) -> dict[int: float]:
    """Return the probability distribution function(PDF) of the total points of the player after drawing
    when we know the cards he holds.
    The variable cards_in_hand should be cards contained by a deck of poke without jokers."""

    CARD_POOL = {1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4, 10: 16}  # The general card pool.
    sum_in_hand = 0  # The point the player get.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the player.

    PDF = {}  # The PDF of the total points of the player after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 52 - len(cards_in_hand)  # The amount of cards in pool.
```

DOI: 10.54254/2753-8818/5/20230516

```
    for j in CARD_POOL:
        sum_after_draw = sum_in_hand + j
        if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1):
            PDF[sum_after_draw + 10] += CARD_POOL[j] / total_cards_in_pool
        elif sum_after_draw <= 21:
            PDF[sum_after_draw] += CARD_POOL[j] / total_cards_in_pool
        else:
            PDF[0] += CARD_POOL[j] / total_cards_in_pool
    # Compute the PDF

    return PDF


def probability_to_draw(cards_in_hand: list[int]) -> float:
    """Return the probability to draw a card according to the player's hands.
    The probability to draw is equal to the probability to increase his points
    divided by the total probability to change his points."""

    prob1 = 0  # The probability to increase total points.
    prob2 = 0  # The probability to decrease total points.
    sum_in_hand = 0  # The player's points.
    PDF = PDF_for_one(cards_in_hand)  # The PDF of his points after draw.

    for c in cards_in_hand:
        sum_in_hand += c
    if sum_in_hand <= 11 and 1 in cards_in_hand:
        sum_in_hand += 10

    for point in PDF_for_one(cards_in_hand):
        if point > sum_in_hand:  # The case his points increases.
            prob1 += PDF[point]
        if point < sum_in_hand:  # The case his points decreases.
            prob2 += PDF[point]

    return prob1 / (prob1 + prob2)


def payoff_for_player1(player1: list[int], player2: list[int]) -> float:
    """Return the total payoff of player1 according to hands of 2 players.
    2 players both obey the strategy stated in probability_to_draw."""

    draw1 = probability_to_draw(player1)  # The probability to draw for player 1.
    draw2 = probability_to_draw(player2)  # The probability to draw for player 2.
    PDF1 = PDF_for_one(player1)  # The PDF for player1 after drawing.
    PDF2 = PDF_for_one(player2)  # The PDF for player2 after drawing.
    payoff11 = 0  # The payoff of player1 if they both draw.
    payoff12 = 0  # The payoff of player1 if only player1 choose to draw.
    payoff21 = 0  # The payoff of player1 if only player2 choose to draw.
    payoff22 = 0  # The payoff of player1 if neither of them draw.
```

DOI: 10.54254/2753-8818/5/20230516

```
    for i in PDF1:
        for j in PDF2:
            if i > j:
                payoff11 += PDF1[i] * PDF2[j]
            elif i < j:
                payoff11 -= PDF1[i] * PDF2[j]

    for i in PDF1:
        if i > sum(player2):
            payoff12 += PDF1[i]
        if i < sum(player2):
            payoff12 -= PDF1[i]

    for j in PDF2:
        if sum(player1) > j:
            payoff21 += PDF2[j]
        if sum(player1) < j:
            payoff21 -= PDF2[j]

    if sum(player1) > sum(player2):
        payoff22 = 1
    elif sum(player1) < sum(player2):
        payoff22 = -1

    payoff = draw1 * draw2 * payoff11 + draw1 * (1 - draw2) * payoff12 + \
             (1 - draw1) * draw2 * payoff21 + (1 - draw1) * (1 - draw2) * payoff22
    return payoff
```

3. 2-player-game from player's perspective

```
def PDF_for_one(cards_in_hand: list[int], cards_opponent: list[int], CARD_POOL: dict[int: int]):
    """Return the probability distribution function(PDF) of the total points of the player after drawing
    when we know his hands and his opponent's hands.
    The variable cards_in_hand should be cards contained by a deck of poke without jokers.
    The variable CARD_POOL should contain the player's hands but not his opponent's."""

    sum_in_hand = 0  # The point the player get.

    for i in cards_in_hand:
        sum_in_hand += i

    for k in cards_opponent:
        CARD_POOL[k] -= 1
    # Removes from the pool the cards that have been drawn by the opponent.

    PDF = {}  # The PDF of the total points of the player after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 52 - len(cards_in_hand) - len(cards_opponent)  # The amount of cards in pool.

    for j in CARD_POOL:
```

DOI: 10.54254/2753-8818/5/20230516

```
        sum_after_draw = sum_in_hand + j
        if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1):
            PDF[sum_after_draw + 10] += CARD_POOL[j] / total_cards_in_pool  # 'A' counts for 11 or 1
as the player want.
        elif sum_after_draw <= 21:
            PDF[sum_after_draw] += CARD_POOL[j] / total_cards_in_pool
        else:
            PDF[0] += CARD_POOL[j] / total_cards_in_pool
    # Compute the PDF

    for k in cards_opponent:
        CARD_POOL[k] += 1

    return PDF


def opponent_draw(cards_opponent: list[int], CARD_POOL: dict[int: int]) -> dict[int: list[int, float]]:
    """Return the probability distribution function(PDF) of the total points of the opponent after drawing
once
    when we know the cards he holds.
    The variable cards_opponent should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable CARD_POOL should be a list of integers referring to the cards left in the pool,
    containing the opponent's hand."""

    sum_in_hand = 0  # The point the dealer get.

    for i in cards_opponent:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the opponent.

    PDF = {}  # The PDF of the total points of the opponent after drawing.

    total_cards_in_pool = 48  # The amount of cards in pool.

    for j in CARD_POOL:
        sum_after_draw = sum_in_hand + j
        if sum_after_draw <= 11 and (1 in cards_opponent or j == 1):
            PDF[j] = [sum_after_draw + 10, CARD_POOL[j] / total_cards_in_pool]
        elif sum_after_draw <= 21:
            PDF[j] = [sum_after_draw, CARD_POOL[j] / total_cards_in_pool]
        else:
            PDF[j] = [0, CARD_POOL[j] / total_cards_in_pool]
    # Compute the PDF

    for i in cards_opponent:
        CARD_POOL[i] += 1

    return PDF
```

DOI: 10.54254/2753-8818/5/20230516

```
def decision(cards_in_hand: list[int]):
    """Return the decision that whether to draw a card according to the player's hands,
    followed by the probability to win and lose before and after drawing.
    The strategy of the player is to draw
    when either the winning probability increases of the losing probability decreases.
    The strategy of the opponent is to draw when the probability to increase his points
    is larger than the probability to decrease his points."""

    global prob_to_draw
    CARD_POOL = {1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4, 10: 16}  # The general card pool.
    sum_in_hand = 0  # The points the player gets.
    p1_win = 0  # The probability to win before drawing.
    p1_lose = 0  # The probability to lose before drawing.
    p2_win = 0  # The probability to win after drawing.
    p2_lose = 0  # The probability to lose after drawing.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    if sum_in_hand <= 11 and 1 in cards_in_hand:
        sum_in_hand += 10
    # Removes from the pool the cards that have been drawn by the player.

    for j in CARD_POOL:
        for k in CARD_POOL:
            sum_opponent = j + k  # The total points of the opponent with 2 initial cards.
            if sum_opponent <= 11 and (1 in cards_in_hand or j == 1 or k == 1):
                sum_opponent += 10

            if j == k:
                prob = CARD_POOL[j] * (CARD_POOL[k] - 1) / 2450
            else:
                prob = CARD_POOL[j] * CARD_POOL[k] / 2450

            PDF_draw = opponent_draw([j, k], CARD_POOL)
            prob_to_increase = 0
            prob_to_decrease = 0
            for case in PDF_draw:
                point = PDF_draw[case][0]
                if point > sum_opponent:
                    prob_to_increase += PDF_draw[case][1]
                elif point > sum_opponent:
                    prob_to_decrease += PDF_draw[case][1]

            if prob_to_increase > prob_to_decrease:  # The case that the opponent choose to draw.
                for l in PDF_draw:
                    if sum_in_hand > PDF_draw[l][0]:
                        p1_win += prob * PDF_draw[l][1]
                    elif sum_in_hand < PDF_draw[l][0]:
                        p1_lose += prob * PDF_draw[l][1]
```

DOI: 10.54254/2753-8818/5/20230516

```
                PDF_self = PDF_for_one(cards_in_hand, [j, k, l], CARD_POOL)
                for m in PDF_self:
                    if m > PDF_draw[l][0]:
                        p2_win += prob * PDF_self[m] * PDF_draw[l][1]
                    elif m < PDF_draw[l][0]:
                        p2_lose += prob * PDF_self[m] * PDF_draw[l][1]
            else:
                if sum_in_hand > sum_opponent:
                    p1_win += prob
                elif sum_in_hand < sum_opponent:
                    p1_lose += prob
                pdf_self = PDF_for_one(cards_in_hand, [j, k], CARD_POOL)
                for m in pdf_self:
                    if m > sum_opponent:
                        p2_win += prob * pdf_self[m]
                    elif m < sum_opponent:
                        p2_lose += prob * pdf_self[m]

    if (p2_win - p1_win) > (p2_lose - p1_lose):  # Draw when winning rate increases or losing rate decreases.
        return 'draw', p1_win, p2_win, p1_lose, p2_lose
    else:
        return 'notdraw', p1_win, p2_win, p1_lose, p2_lose
```

4. 2+-player-game with a dealer

```
def PDF1_for_dealer(cards_in_hand: list[int], CARD_POOL: dict[int: int]) -> dict[int: float]:
    """Return the probability distribution function(PDF) of the total points of the player after drawing once
    when we know the cards he holds.
    The variable cards_in_hand should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable CARD_POOL should be a list of integers referring to the cards left in the pool."""

    sum_in_hand = 0  # The point the dealer get.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the dealer.

    PDF = {}  # The PDF of the total points of the dealer after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 0
    for c in CARD_POOL:
        total_cards_in_pool += CARD_POOL[c]  # The amount of cards in pool.

    for j in CARD_POOL:
        sum_after_draw = sum_in_hand + j
        if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1):
```

DOI: 10.54254/2753-8818/5/20230516

```
            PDF[sum_after_draw + 10] += CARD_POOL[j] / total_cards_in_pool
        elif sum_after_draw <= 21:
            PDF[sum_after_draw] += CARD_POOL[j] / total_cards_in_pool
        else:
            PDF[0] += CARD_POOL[j] / total_cards_in_pool
    # Compute the PDF

    return PDF


def PDF2_for_dealer(cards_in_hand: list[int], CARD_POOL: dict[int: int]) -> dict[int: float]:
    """Return the probability distribution function(PDF) of the total points of the player after drawing 2
cards
    when we know the cards he holds.
    The variable cards_in_hand should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable CARD_POOL should be a list of integers referring to the cards left in the pool."""

    sum_in_hand = 0  # The point the dealer get.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the dealer.

    PDF = {}  # The PDF of the total points of the dealer after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 0
    for c in CARD_POOL:
        total_cards_in_pool += CARD_POOL[c]  # The amount of cards in pool.

    for j in CARD_POOL:
        for k in CARD_POOL:
            sum_after_draw = sum_in_hand + j + k
            if j == k:
                prob = CARD_POOL[j] * (CARD_POOL[k] - 1) / (total_cards_in_pool *
(total_cards_in_pool - 1))
            else:
                prob = CARD_POOL[j] * CARD_POOL[k] / (total_cards_in_pool * (total_cards_in_pool -
1))

            if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1 or k == 1):
                PDF[sum_after_draw + 10] += prob
            elif sum_after_draw <= 21:
                PDF[sum_after_draw] += prob
            else:
                PDF[0] += prob
    # Compute the PDF
```

DOI: 10.54254/2753-8818/5/20230516

```
    return PDF


def PDF3_for_dealer(cards_in_hand: list[int], CARD_POOL: dict[int: int]) -> dict[int: float]:
    """Return the probability distribution function(PDF) of the total points of the player after drawing 3 cards
    when we know the cards he holds.
    The variable cards_in_hand should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable CARD_POOL should be a list of integers referring to the cards left in the pool."""

    sum_in_hand = 0  # The point the dealer get.

    for i in cards_in_hand:
        CARD_POOL[i] -= 1
        sum_in_hand += i
    # Removes from the pool the cards that have been drawn by the dealer.

    PDF = {}  # The PDF of the total points of the dealer after drawing.
    for k in range(22):
        PDF[k] = 0

    total_cards_in_pool = 0
    for c in CARD_POOL:
        total_cards_in_pool += CARD_POOL[c]  # The amount of cards in pool.

    for j in CARD_POOL:
        for k in CARD_POOL:
            for l in CARD_POOL:
                sum_after_draw = sum_in_hand + j + k + l
                if j == k and k == l:
                    prob = CARD_POOL[j] * (CARD_POOL[k] - 1) * (CARD_POOL[l] - 2) / (
                            total_cards_in_pool * (total_cards_in_pool - 1) * (total_cards_in_pool - 2))
                elif j == k:
                    prob = CARD_POOL[j] * (CARD_POOL[k] - 1) * CARD_POOL[l] / (
                            total_cards_in_pool * (total_cards_in_pool - 1) * (total_cards_in_pool - 2))
                elif j == l:
                    prob = CARD_POOL[j] * (CARD_POOL[l] - 1) * CARD_POOL[k] / (
                            total_cards_in_pool * (total_cards_in_pool - 1) * (total_cards_in_pool - 2))
                elif k == l:
                    prob = CARD_POOL[k] * (CARD_POOL[l] - 1) * CARD_POOL[j] / (
                            total_cards_in_pool * (total_cards_in_pool - 1) * (total_cards_in_pool - 2))
                else:
                    prob = CARD_POOL[j] * CARD_POOL[k] * CARD_POOL[l] / (
                            total_cards_in_pool * (total_cards_in_pool - 1) * (total_cards_in_pool - 2))

                if sum_after_draw <= 11 and (1 in cards_in_hand or j == 1 or k == 1 or l == 1):
                    PDF[sum_after_draw + 10] += prob
                elif sum_after_draw <= 21:
                    PDF[sum_after_draw] += prob
                else:
```

DOI: 10.54254/2753-8818/5/20230516

```
                PDF[0] += prob
    # Compute the PDF

    return PDF


def notdraw_for_dealer(cards_in_hand: list[int], loafers: dict) -> float:
    """return the payoff if the dealer chooses not to draw.
    The variable cards_in_hand should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable loafers should be a dictionary with keys referring the serial numbers or name of loafers.
    The value of loafers should be a list of a integer and a list.
    The integer refers to the loafers' bets, while the list refers to cards in his hand."""

    notdraw = 0  # The payoff if the dealer chooses not to draw.
    dealer = 0  # The total point of the dealer.

    for c in cards_in_hand:
        dealer += c
    if dealer <= 11 and 1 in cards_in_hand:
        dealer += 10

    for i in loafers:
        point = 0  # The total points of loafer i.
        for c in loafers[i][1]:
            point += c
        if point <= 11 and 1 in loafers[i][1]:
            point += 10
        elif point > 21:
            point = 0

        if dealer < point:
            notdraw -= loafers[i][0]
        else:
            notdraw += loafers[i][0]

    return notdraw


def draw_up_to_3(cards_in_hand, loafers):
    """Return the payoff of the dealer if he chooses to draw according to the cards of loafers shown on
the table.
    The variable cards_in_hand should be a list of integers.
    The corresponding cards should be contained by a deck of poke without jokers.
    The variable loafers should be a dictionary with keys referring the serial numbers or name of loafers.
    The value of loafers should be a list of a integer and a list.
    The integer refers to the loafers' bets, while the list refers to cards in his hand."""

    CARD_POOL = {1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4, 10: 16}  # The general card pool.
    for loafer in loafers:
        for card in loafers[loafer][1]:
```

DOI: 10.54254/2753-8818/5/20230516

```
            CARD_POOL[card] -= 1
    PDF1 = PDF1_for_dealer(cards_in_hand, CARD_POOL)  # The PDF of the dealer's total point after drawing once.
    PDF2 = PDF2_for_dealer(cards_in_hand, CARD_POOL)  # The PDF of the dealer's total point after drawing twice.
    PDF3 = PDF3_for_dealer(cards_in_hand, CARD_POOL)  # The PDF of the dealer's total point after drawing 3 times.
    draw1 = 0  # The payoff of the dealer if he chooses to draw.
    draw2 = 0  # The payoff of the dealer if he chooses to draw.
    draw3 = 0  # The payoff of the dealer if he chooses to draw.

    for i in loafers:
        point = 0  # The total points of loafer i.
        for c in loafers[i][1]:
            point += c
        if point <= 11 and 1 in loafers[i][1]:
            point += 10
        elif point > 21:
            point = 0

        j = 0
        lose_rate1 = 0  # The probability dealer lose to the loafer i after draw once.
        lose_rate2 = 0  # The probability dealer lose to the loafer i after draw twice.
        lose_rate3 = 0  # The probability dealer lose to the loafer i after draw 3 cards.
        while j < point:
            lose_rate1 += PDF1[j]
            lose_rate2 += PDF2[j]
            lose_rate3 += PDF3[j]
            j += 1
        draw1 += (1 - 2 * lose_rate1) * loafers[i][0]
        draw2 += (1 - 2 * lose_rate2) * loafers[i][0]
        draw3 += (1 - 2 * lose_rate3) * loafers[i][0]

    draw0 = notdraw_for_dealer(cards_in_hand, loafers)

    return draw0, draw1, draw2, draw3


def dealer_decision(cards_in_hand, loafers):
    """Return the dealer's decision maximizing the payoff."""

    draw = draw_up_to_3(cards_in_hand, loafers)
    max_payoff = max(draw)
    if max_payoff > draw[0]:
        return 'draw'
    else:
        return 'not draw'
```

References

[1] R. B. Myerson, Game theory: analysis of conflict. Harvard university press, 1997.

DOI: 10.54254/2753-8818/5/20230516

[2] J. Von Neumann, “On the theory of games of strategy,” Contributions to the Theory of Games, vol. 4, pp. 13–42, 1959.
[3] E. Rasmusen, Games and information. Basil Blackwell Oxford, 1989, vol. 13.
[4] A. Zimran, A. Klis, A. Fuster, and C. Rivelli, “The game of blackjack and analysis of counting cards,” 2009.
[5] R. R. Baldwin, W. E. Cantey, H. Maisel, and J. P. McDermott, “The optimum strategy in blackjack,” Journal of the American Statistical Association, vol. 51, no. 275, pp. 429–439, 1956.
[6] J. F. NAs, “Equilibrium points in n-person games,” Proceedings of the National Academy of Sciences of the United States of America, vol. 36, no. 1, pp. 48–49, 1950.
[7] N. E. Glynatsi and V. A. Knight, “Game theory and python: An educational tutorial to game theory and repeated games using python,” Journal of Open Source Education, vol. 4, no. 39, p. 78, 2021.
[8] V. Knight and J. Campbell, “Nashpy: A python library for the computation of nash equilibria,” The Journal of Open Source Software, vol. 3, no. 30, p. 904, 2018.