Bob Lin
Math381
Homework5
I will simulate a famous card game named Spade 7. This game is another version of a well-known card game called Domino. I just learned the game and wanted to simulate it here. The Spade 7 has interesting rules as following:
Game preparation:
The game required four players and a deck of playing card except for two jokers. The card will be played on a "table". "A" represents number 1, and "J", "Q", and "K" represent number 11, 12, and 13 separately, all cards will be randomly and evenly deal to four players.

The play begins with the player with Spade 7, and he must play that card on the table first. Then switch to next person, who has following two options:

1. He can play any other card nearby to the number on the board with same suits. (ex: if Spade 7 is on the table, he can play Spade 8 and Spade 6)

2. If he cannot play any card, he must drop a card and only he knows the number and suits on the card.

The game ends when all players drop or play all of their card.

Then, we score each player, and the player(s) with lowest score win(s) the game. The score is calculated by basically adding the rank of each card with 1, 11, 12, 13 for "A", "J", "Q", and "K"

During the simulation, I will also introduce a tricky strategy that let all players use, this is denoted by strategy A:

1. When playing a card, each player will play card with lowest rank on it first

2. When dropping a card, each player will drop the card with lowest rank on it first.

To investigate how well this strategy will work, I develop a simple strategy to compare with that above, and this is denoted by strategy B:

1. When playing a card, we use an easy default method, which is each player will play the first card he has in the order of Spade, Heart, Diamond, and Club from A(1) to K(13) (Spade comes before Heart and so on)

2. When dropping a card, each player will drop the first card he has in the order of Spade, Heart, Diamond, and Club from A to K (1 to 13) (same way in

choosing card as we play)

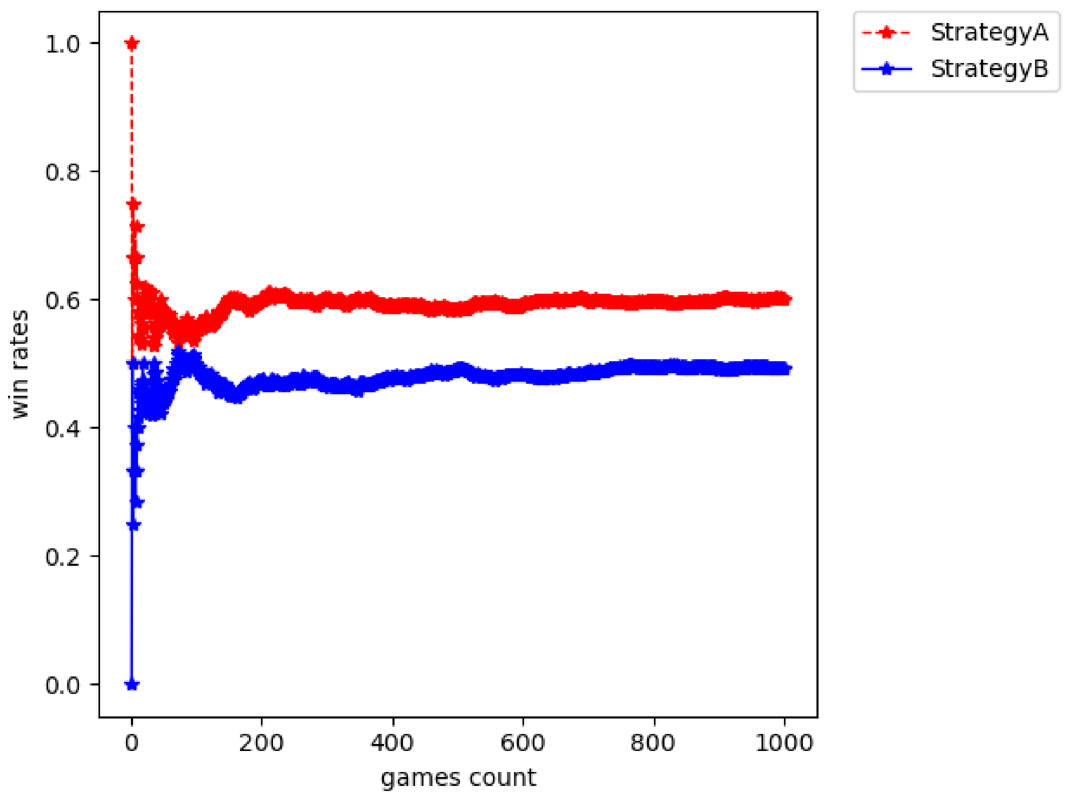
My goal is not to simulate one game to see which strategy wins but to simulate many games and see the win rate of each strategy by letting player 1 and player 2 use first strategy and player 3 and player 4 adapt the second one.

To perform my idea, I wrote following code to simulate the games for 1000 times to see which strategy perform better and plot it:

```
import random
import matplotlib.pyplot as plt

n = 4       # number of players
num_game = 1000;
current_game = 0;
s1 = 0;
s2 = 0;
game_count = []
s1_winrates = []
s2_winrates = []
hist1 = []
hist2 = []
'''
index from 0 to 12 is Spade
index from 13 to 25 is Heart
index from 26 to 38 is Diamond
index from 39 to 51 is Club
card[n] % 4 + 1 gives me the number on the card
int(card[n] / 4) gives me the suits of the card (0 is Spade, 1 is Heart, 2 is Diamond, 3 is Club)
each index has number 1 to 4 represent which player own this card
after playing a card, the corresponding number in that index becomes 0
after drop a card, the corresponding number in that index times -1
'''
card = [0] * 52 # store rules above
deal_card = [0] * 52 # number from 0 to 51 of which card is not deal
turn = 1; # in which player's turn


'''deal'''
def deal():
    global  deal_card, card
    card = [0] * 52 # renew the value if playing more times
    deal_card = [0] * 52 # renew the value if playing more times
    for i in range(0, 52):
        deal_card[i] = i
    for player in range(1, 5):
        for count in range(0, 13):
            length = len(deal_card) - 1
            rand = random.randint(0, length)
            card[deal_card[rand]] = player
            deal_card.pop(rand)


'''play one game'''
def playgame():
    global turn, card, game_count, current_game
    current_game = current_game + 1
    game_count.append(current_game)
    deal()
    turn = card[6] # Spade 7
    # print("Player", turn, "plays", "Spade 7")
    card[6] = 0
    switch_turn()
    for round in range(0, 51):
        if turn == 1 or turn == 2:
            play = strategy1() # play from 1 to 52, is ith card
        else:
            play = strategy2()
        if play > 0:
            card[play - 1] = 0
            # print("Player", turn, "plays", formalize(play - 1))
        elif play < 0:
            card[-1 * play - 1] = -1 * card[-1 * play - 1]
            # print("Player", turn, "drops", formalize(-1 * play - 1))
        switch_turn()
    count_Score()


'''switch player'''
def switch_turn():
    global turn
    if turn == 4:
        turn = 1;
    else:
        turn = turn + 1


'''a complicated strategy'''
def strategy1():
    attempt_drop = 0
    for i in range(0, 52):
        if card[i] == turn:
            attempt_drop = -1 * (i + 1)
            break

    play, drop = search_play(0, 6, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play, drop = search_play(13, 19, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play, drop = search_play(26, 32, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play, drop = search_play(39, 45, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play,drop = search_play(6, 13, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play,drop = search_play(19, 26, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play,drop = search_play(32, 39, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop
    play, drop = search_play(45, 52, True)
    if play != 0: return play
    if drop != 0 and (-1 * attempt_drop - 1) % 13 >= (-1 * drop - 1) % 13: attempt_drop = drop

    if play != 0:
        return play
    else:
        return attempt_drop


'''a simple strategy'''
def strategy2():
    play, drop = search_play(0, 52, False)
    if play != 0:
        return play
    else:
        for i in range(0, 52):
            if card[i] == turn:
                return -1 * (i + 1)

def strategy3():
    playlist = []
    for i in range(0, 52):
        if card[i] == 0:
            if i % 13 != 0:
                if i % 13 < 7:
                    if card[i - 1] == turn:
                        playlist.append(i)
                if 5 < i % 13 < 12:
                    if card[i + 1] == turn:
                        playlist.append(i + 2)
    if playlist:
        return playlist[random.randint(0, len(playlist) - 1)]
    else:
        droplist = []
        for i in range(0, 52):
            if card[i] == turn:
                droplist.append(i + 1)
        return -1 * droplist[random.randint(0, len(playlist) - 1)]


def search_play(range1, range2, if_drop):
    play = 0;
    drop = 0;
    for i in range(range1, range2):
        if card[i] == 0:
            if i % 13 != 0:
                if i % 13 < 7:
                    if card[i - 1] == turn:
                        play = i
                        break
                if 5 < i % 13 < 12:
                    if card[i + 1] == turn:
                        play = i + 2
                        break
        if if_drop:
            if card[i] == turn and i % 13 <= (-1 * drop - 1) % 13:
                drop = -1 * (i + 1)
    if play == 0:
        if card[19] == turn:
            play = 20
        elif card[32] == turn:
            play = 33
        elif card[45] == turn:
            play = 46
    return play, drop


'''formalize the print statement'''
def formalize(play):
    num = play % 13
    suits = int(play / 13)
    if suits == 0:
        output = "Spade "
    elif suits == 1:
        output = "Heart "
    elif suits == 2:
        output = "Diamond "
    else:
        output = "Club "
    if 0 < num < 10:
        output = output + str(num + 1)
    elif num == 0:
        output = output + "A"
    elif num == 10:
        output = output + "J"
    elif num == 11:
        output = output + "Q"
    elif num == 12:
        output = output + "K"
    return output


def count_Score():
    global s1_winrates, s2_winrates, s1, s2
    player = [0, 0, 0, 0]
    for i in range(0, 52):
        if card[i] == -1:
            player[0] = player[0] + i % 13 + 1
        elif card[i] == -2:
            player[1] = player[1] + i % 13 + 1
        elif card[i] == -3:
            player[2] = player[2] + i % 13 + 1
        elif card[i] == -4:
            player[3] = player[3] + i % 13 + 1
```

```
#print("player 1 scores:", str(player[0]))
#print("player 2 scores:", str(player[1]))
#print("player 3 scores:", str(player[2]))
#print("player 4 scores:", str(player[3]))
minScore = min(player)
player1 = -1
player2 = -1
player3 = -1
player4 = -1
if player.count(minScore) == 1:
    player1 = player.index(minScore) + 1
    '''print("The winner is: player", str(player1))'''
elif player.count(minScore) == 2:
    player1 = player.index(min(player)) + 1
    player2 = player[player1:].index(min(player)) + 1 + player1
    '''print("The winner is: player", str(player1), "and", str(player2) )'''
elif player.count(minScore) == 3:
    list = [1,2,3,4]
    for i in range(0,4):
        if player[i] != minScore:
            list.pop(i)
            break
    player1 = list[0]
    player2 = list[1]
    player3 = list[2]
    output = "The winner is: player "
    output = output + str(player1) + ", "
    output = output + str(player2) + ",and "
    output = output + str(player3)
    '''print(output)'''
else:
    player1 = 1
    player2 = 2
    player3 = 3
    player4 = 4
    '''print("The winner is: player", "1,", "2,", "3,and 4")'''
if 0 < player1 < 3:
    s1 = s1 + 1
if 2 < player1 < 5:
    s2 = s2 + 1
if 0 < player2 < 3:
    s1 = s1 + 1
if 2 < player2 < 5:
    s2 = s2 + 1
    if player3 > 0:
        s2 = s2 + 1
    if player4 > 0:
        s2 = s2 + 1
    s1_winrates.append(s1 / current_game)
    s2_winrates.append(s2 / current_game)


for i in range(0, 1):
    print(i)
    s1_winrates = []
    s2_winrates = []
    s1 = 0
    s2 = 0
    current_game = 0
    game_count = []
    for j in range(0, num_game):
        '''print("Game:", str(i))'''
        playgame()
    hist1.append(s1_winrates[num_game - 1])
    hist2.append(s2_winrates[num_game - 1])

'''
plt.hist(hist1, bins=40, facecolor="blue", edgecolor="black", alpha=0.7)
plt.xlabel("win times")
# 显示纵轴标签
plt.ylabel("frequency")
# 显示图标题
plt.title("Strategy 1")
plt.show()

plt.hist(hist2, bins=40, facecolor="red", edgecolor="black", alpha=0.7)
plt.xlabel("win times")
# 显示纵轴标签
plt.ylabel("frequency")
# 显示图标题
plt.title("Strategy 2")
plt.show()
'''

plt.plot(game_count, s1_winrates, color="r", linestyle="--", marker="*", linewidth=1.0, label='Strategy1')
plt.plot(game_count, s2_winrates, color="b", linestyle="-", marker="*", linewidth=1.0, label='Strategy2')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.xlabel('games count')
plt.ylabel('win times')
plt.show()
```

Here, I plot the comparison of the win rates of the two strategies:



Here, I see that both methods are converge, and the Strategy A is near 0.6 win rate, while Strategy B only has about 0.4 win rate.

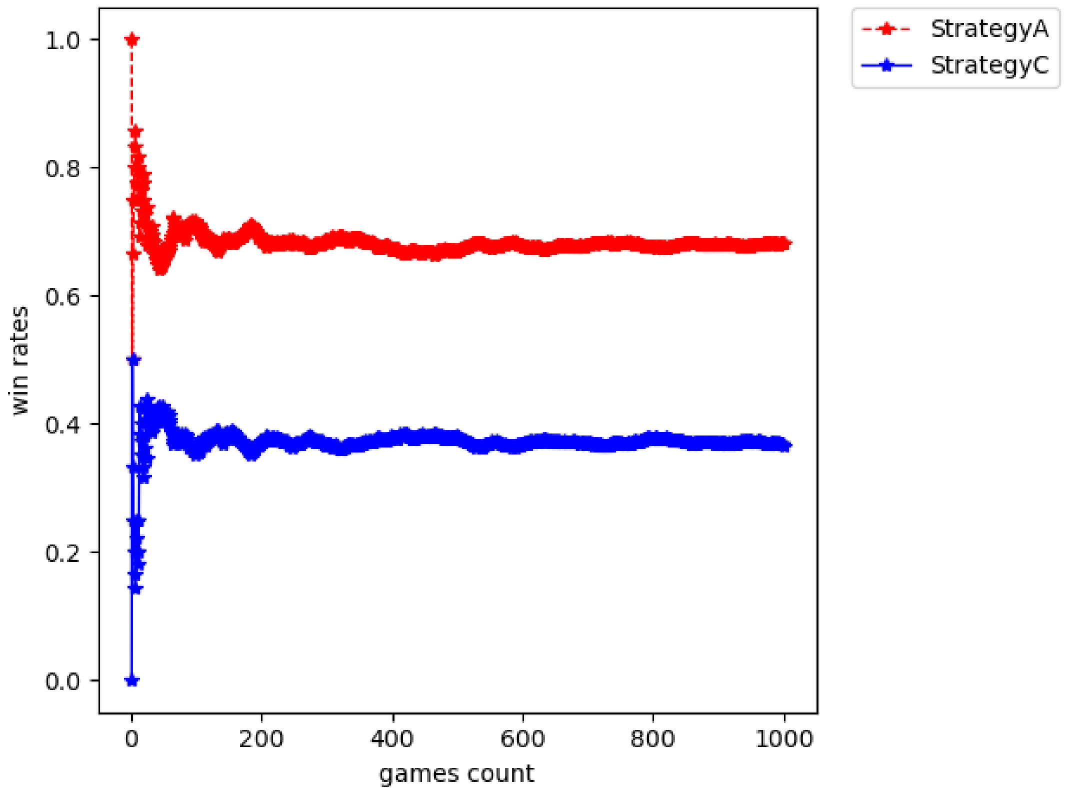
In this result, Strategy A perform well always. Then, I want to introduce a random Strategy C:

1. When playing a card, each player will play a card by choosing randomly from cards he has

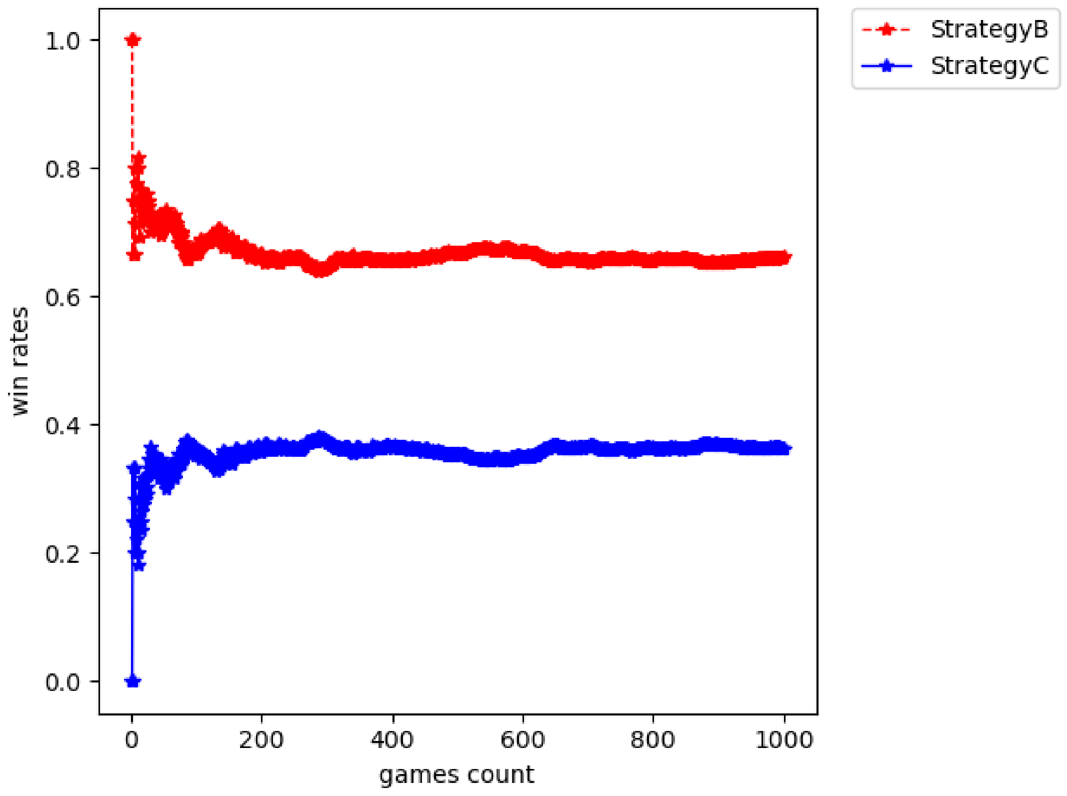
2. When dropping a card, each player will drop a card by choosing randomly from cards he has

Strategy C is different from Strategy B because Strategy C choosing card randomly instead of choosing in order. To see how well this Strategy work, I will use python to plot the comparison of Strategy A and Strategy C, Strategy B and Strategy C:

(player 1 and player 2) A vs C (player 3 and player 4):



(player 1 and player 2) B vs C (player 3 and player 4):

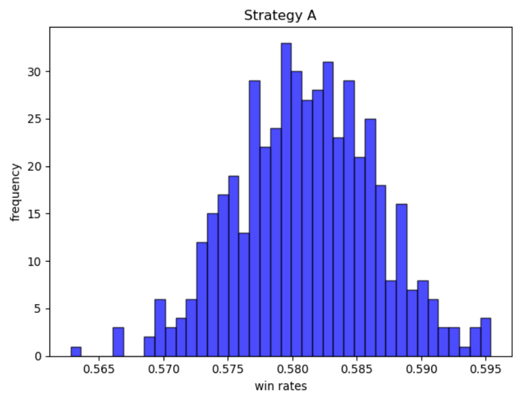


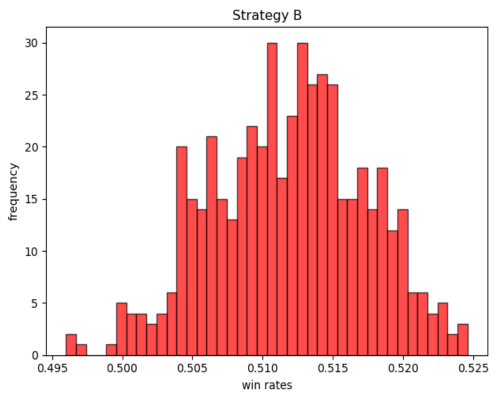
From the plots above, we see that all methods converge, and the win rate between two Strategy is below (the number is the win rate for col):

	A	B	C
A	-	0.6	0.62
B	0.4	-	0.62
C	0.38	0.38	-

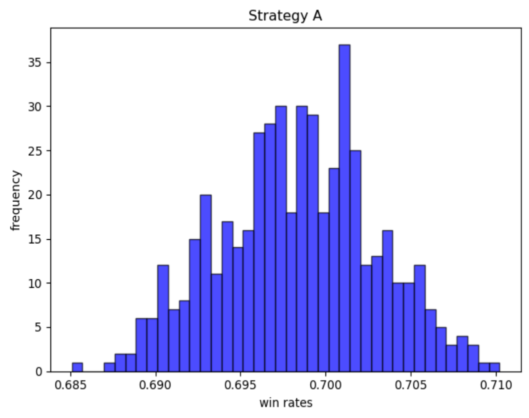
So, we can conclude that player with Strategy A > player with Strategy B > player with Strategy C. To further approve it, I will use Central limit Theorem. First, I will calculate 500 sample win rate in running 10000 games:

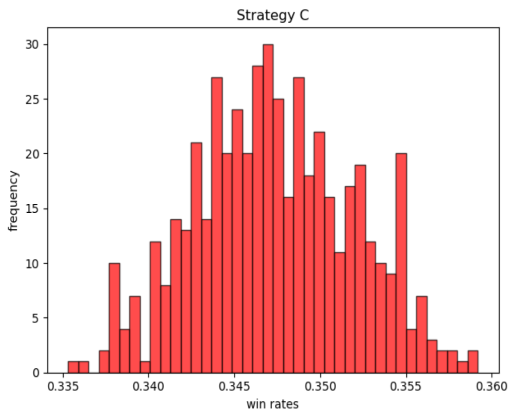
(player 1 and player 2) A vs B (player 3 and player 4):



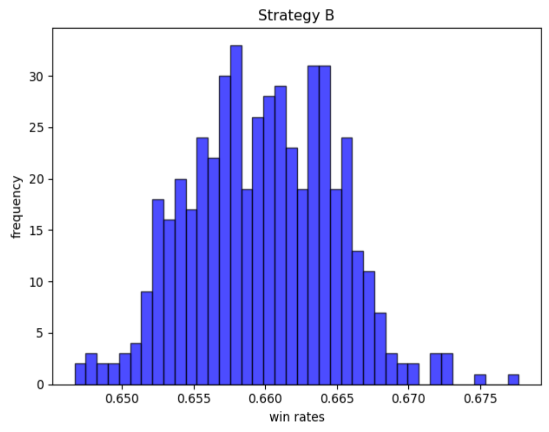


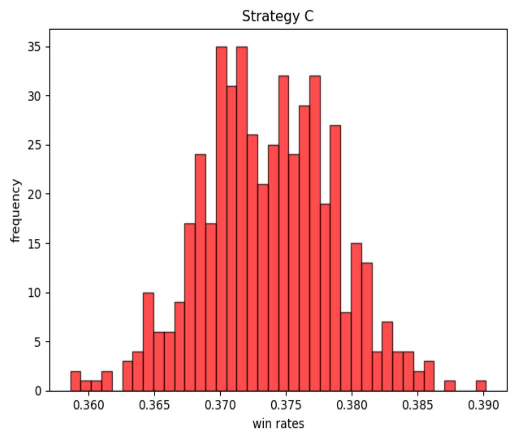
(player 1 and player 2) A vs C (player 3 and player 4):





(player 1 and player 2) B vs C (player 3 and player 4):





We see that all of three cases are normal distribution, so I can make conclusion according to Central limit theorem. Because $1 - 2^{1000} \approx 100\ \%$, I am 100 percent confident that:

1. In A vs B, A lies in range from 0.565 to 0.595, and B is in range from 0.495 to 0.525
2. In A vs C, A lies in range from 0.685 to 0.710, and C is in range from 0.335 to 0.360
3. In B vs C, B lies in range from 0.647 to 0.678, and C is in range from 0.360 to 0.390

Based on the plots and data, I can make conclusions toward three cases. In the first case, I know that player 1 and player 2 with strategy A has greater win rate, because their range is higher than that of player 3 and player 4 with Strategy B overall without any overlap. Also, in the second and third case, I am confident that player 1 and player 2 with strategy A has a greater win rate than player 3 and player 4 with Strategy C, and player 1 and player 2 with strategy B has a greater win rate than player 3 and player 4 with Strategy C. So, I can conclude that I am sure that playing and dropping cards based on cards' rank is better than playing and dropping cards in a fixed order, and I am sure that both of methods are better than playing and dropping cards randomly!

To further explore the win rate with using different strategy, I will let player 1 uses Strategy A, B, and C separately, and other players use different strategy each time. To test it, I will collect 10 average win rate from 20000 games as below. Player 1 is denoted P1, and other are denoted Po:

1. Player 1 uses Strategy A, and other use Strategy B:

P1:

[0.2853, 0.2841, 0.28965, 0.2872, 0.2892, 0.2903, 0.2896, 0.2908, 0.2902, 0.2859]

Po:

[0.79615, 0.794, 0.79105, 0.79695, 0.7915, 0.79365, 0.7917, 0.794, 0.7908, 0.79415]

2. Player 1 uses Strategy A, and other use Strategy C:

P1:

[0.38335, 0.37775, 0.37885, 0.3738, 0.3796, 0.38385, 0.3742, 0.3782, 0.3765, 0.3797]

Po:

[0.64645, 0.6557, 0.64985, 0.6571, 0.65165, 0.6476, 0.6545, 0.6521, 0.65625, 0.64975]

3. Player 1 uses Strategy B, and other use Strategy A:

P1:

[0.2398, 0.2417, 0.2446, 0.2472, 0.2415, 0.245, 0.2467, 0.2478, 0.24365, 0.24315]

Po:

[0.8646, 0.86645, 0.8599, 0.86095, 0.86335, 0.8616, 0.85735, 0.86325, 0.8625, 0.86255]

4. Player 1 uses Strategy B, and other use Strategy C:

P1:

[0.3446, 0.34155, 0.3436, 0.3394, 0.3467, 0.3436, 0.34425, 0.33995, 0.34505, 0.347]

Po:

[0.68505, 0.68775, 0.6859, 0.69135, 0.6816, 0.6841, 0.68545, 0.68875, 0.6858, 0.68205]

5. Player 1 uses Strategy C, and other use Strategy A:

P1:

[0.12935, 0.1306, 0.13085, 0.1326, 0.1338, 0.1329, 0.1327, 0.1296, 0.1348, 0.132]

Po:

[0.94755, 0.9459, 0.94585, 0.94465, 0.94945, 0.946, 0.9453, 0.9471, 0.942, 0.94395]

6. Player 1 uses Strategy C, and other use Strategy B:

P1:

[0.13525, 0.13975, 0.14205, 0.1393, 0.14015, 0.13355, 0.1324, 0.1398, 0.1338, 0.13895]

Po:

[0.91015, 0.90975, 0.90505, 0.9083, 0.9091, 0.91325, 0.91375, 0.90845, 0.9162, 0.90885]

By using Central limit Theorem, because $1 - 2^{10} \approx 99.8\,\%$, I am 99.8 percent confident that the win rates lies in :

	Minimum	Maximum
Player 1 with Strategy A	0.2841	0.2908
Case 1, other players with B	0.7908	0.79695
Player 1 with Strategy A	0.37335	0.38385
Case 1, other players with C	0.64654	0.6571
Player 1 with Strategy B	0.2398	0.2478
Case 2, other players with A	0.85735	0.86645
Player 1 with Strategy B	0.3394	0.347
Case 2, other players with C	0.6816	0.69135
Player 1 with Strategy C	0.12935	0.1348
Case 2, other players with A	0.94942	0.94945
Player 1 with Strategy C	0.1324	0.14205
Case 2, other players with B	0.90505	0.9162

To sum up, Player 1 has a greater wins rates when using Strategy A against players with other strategy than when using other Strategy against players with strategy A, and I also found that Player 1 has a greater wins rates using when Strategy B than using Strategy C against other players with other strategy.