Jane Street Blotto Solution - Armaan Narula

Solution: (0, 0, 46, 1, 1, 25, 12, 15, 0, 0)

1 Introduction

We begin by trying to write the conditions of the game as a set of equations/constraints:

We will denote the allocation of soldiers by each player in the form: $\mathbf{x} = (x_1 x_2 x_3 ... x_{10})$ where

$$\sum_{i=1}^{10} x_i = 100$$

Note that the number off possible combinations for $\mathbf{x}$ is $\binom{109}{9}$, which can be shown using a counting argument.

An important observation that we can make is that there is no allocation that beats all other allocations, proved below:

Theorem: There is no allocation $\mathbf{a} = (a_1 a_2 a_3 ... a_{10})$ that beats all other allocations.

Proof: (**by contradiction**): Assume there is a strategy $\mathbf{a} = (a_1 a_2 a_3 ... a_{10})$ (such that all values sum to 100) that beats all others.

Now consider the strategy $\mathbf{b} = (a_1 - 1, a_2 + 1, a_3 ... a_{10})$

Note: $\mathbf{b}$ is a strategy that satisfies the conditions because if

$$\sum_{i=1}^{10} a_i = 100$$

then

$$(a_1 - 1) + (a_2 + 1) + \sum_{i=3}^{10} a_i \;\; = \;\; \sum_{i=1}^{10} a_i \;\; = \;\; 100$$

If in a game, Player 1 plays $\mathbf{a}$ and Player 2 players $\mathbf{b}$:

- $a_1 > a_1 - 1$ so Player 1 gets 1 point at castle 1
- $a_2 < a_2 + 1$ so Player 2 gets 2 points are castle 2
- At all other castles, the number of soldiers are the same so they both get 0 points at all castles after castle 2

Therefore, Player 1 will get 1 point and Player 2 will get 2 points.

Therefore, we have that strategy $\mathbf{b}$ beats strategy $\mathbf{a}$ and so we have a contradiction.

Therefore since every strategy can be beaten, we want to find some "optimal allocation" that maximises the mean score against the other allocations other players may play. I will do this by creating a sample space of different allocations (on Python) I think people will play, depending on different strategies people may play and how likely they are to play these strategies.

2 Possible Strategies by other players

2.1 Strategy 1: Aiming to win first 3 castles:

With the 3 consecutive wins rule, this may seem like a good strategy at first. If a player is able to win the first 3 castles, they get 1+2+3 = 6 points as well as the other 49 points available.

Therefore, we would expect someone using this strategy to allocate most of their soldiers to the first 3 castles. One important question to think about is how will the soldiers be allocated in this case.

We could have cases such as:

$$(17, 33, 50, 0...0) \quad (33, 33, 34, 0...0,) \quad (30, 30, 40, 0...0)$$

To include this in our sample space, I will assume the number of soldiers follow :

- a normal distribution of N(30,6) at the 1st castle
- a normal distribution of N(33,6) at the 2nd castle
- a normal distribution of N(37,6) at the 3rd castle
- Zeroes everywhere else

(Note: I will standardise the scores and round up and down according to ensure the values sum to 100)

I think a normal distribution will be a good distribution to predict the value that a player may allocate because of its properties, such as the distribution being symmetric and centred around the mean. For example, I have used a normal distribution N(37,6) for the 3rd value of this strategy. I would expect people using this strategy to allocate a mean of around 37 soldiers, and I think 68% of the time being within the range 31 to 43 soldiers (one standard deviation away from the mean) is a good approximation for how players using this strategy may allocate their soldiers.

I predict that people using this strategy will allocate the most resources to the 3rd castle since it is worth the most, and also because other strategies are likely to try and win castle 3, and this is why the mean of the normal distribution for the 3rd value is the highest.

2.2 Strategy 2: Other attempts to win 3 consecutive castles at the start

I am going to split this strategy up into 2 cases:

2.2.1 Aiming to win castles 3, 4 and 5

One such response to strategy 1 is to try and win castles 3, 4 and 5. By trying to win castle 3, we are reducing the chances of losing 3 in a row, and since winning castles 3,4 and 5 gives a total score of 12, this is larger than 6 and gives us the opportunity to win the later ones as well. Therefore we may see allocations such as:

$$(0, 0, 34, 34, 33, 0...0) \quad (0, 0, 50, 25, 25, 0...0)$$

To include this in our sample space, I will assume the number of soldiers follow :

- a normal distribution of N(37, 6) for the 3rd castle
- a normal distribution of N(30, 6) for the 4th castle
- a normal distribution of N(33, 6) for the 5th castle

I think that to avoid losing the first 3 castles, the player will allocate more soldiers to castle 3, followed by castle 5 (since winning castle 5 is worth more points than castle 4)

2.2.2 Aiming to win castles 2, 3 and 4

Similar to the strategy above, this strategy will now try to win castles 2, 3 and 4 as a response to strategy 1. Since this strategy aims to beat strategy 1 and to win castles 2, 3 and 4 (so that it wins all the castles after) I predict they will allocate the more soldiers to castle 3, since the strategies above also rely on winning this castle.

Therefore, to include this in our sample space, I will assume the number of soldiers follow :

- a normal distribution of N(31, 6) for the 2nd castle
- a normal distribution of N(37, 6) for the 3rd castle
- a normal distribution of N(32, 6) for the 4th castle

2.3 Strategy 3: Trying to win the later castles:

In this strategy, we may expect players to try to win the later castles since they are worth more points. For this strategy to be successful, we would need to avoid losing 3 in a row at the start. I am going to split this up into 2 cases, depending on where we place soldiers to avoid losing 3 in a row.

2.3.1

The first case I will look at is trying to place a a large number of soldiers at castles 3 and 6, as this will help to avoid losing 3 in a row, and then place the remaining castles in positions 7 and 8 with the aim of winning castles 6, 7 and 8 in a row since many players will have already allocated all of their soldiers at previous castles.

Examples of possible allocations:

$$(0, 0, 38, 0, 0, 38, 12, 12, 0, 0) \quad (0, 0, 33, 0, 0, 36, 17, 14, 0, 0)$$

To include this in our sample space, I will assume the number of soldiers follow :

- a normal distribution of N(36, 6) for the 3rd castle
- a normal distribution of N(33, 6) for the 6th castle
- a normal distribution of N(15, 3) for the 7th castle
- a normal distribution of N(16, 3) for the 8th castle

2.3.2

Similar to the strategy above, but players will now place a large number of soldiers in positions 2 and 5. This strategy may be more effective if people playing strategy 1 decide to place slightly less soldiers at castle 2 since it is worth less points than strategy 3.

Examples of possible allocations:

$$(0, 38, 0, 0, 37, 0, 12, 13, 0, 0) \quad (0, 36, 0, 0, 38, 0, 12, 14, 0, 0)$$

I will include this in my sample by assuming the number of soldiers follow:

- a normal distribution of N(33, 6) for the 2nd castle
- a normal distribution of N(36, 6) for the 5th castle
- a normal distribution of N(15, 3) for the 6th castle
- a normal distribution of N(16, 3) for the 7th castle

2.3.3

It is also worth considering that some people may expect the least soldiers to be placed at castle 1 in most strategies, and so placing more soldiers here can help increase the chances of losing 3 in a row. I will include this strategy in my sample space by assuming the number of soldiers follows:

- a normal distribution of N(33, 6) for the 1st castle
- a normal distribution of N(36, 6) for the 4th castle
- a normal distribution of N(15, 3) for the 5th castle
- a normal distribution of N(16, 3) for the 6th castle

2.4 Strategy 4:

We have only considered a couple of possible strategies Jane Streeters may use. It is very likely that there are strategies we haven't thought of, and it is important to try to somehow incorporate some of the different strategies people may pick. For example, none of the strategies I have looked at allocate soldiers to the last 3 castles; this is because I think that it is better to try and also win the earlier castles, since 10+9+8 = 27 (less than half the points available), and you can will these castles by winning 3 consecutive at lower castles.

Therefore in my sample of strategies, I will also include a strategy of random allocations. I will do this by drawing each number from a discrete uniform distribution and standardising to ensure all the numbers sum to 100.

3 Creating sample space of strategies

We have now discussed several strategies, we now want to create our sample space of entries we think other players may submit.

I will construct the sample space based on the strategies I think will be most popular. Since we are testing against "several hundred" other strategies, I will create a sample space of 1000 entries as follows:

- 200 entries of strategy 1
- 200 entries of strategy 2.1
- 80 entries of strategy 2.2
- 200 entries of strategy 3.1
- 120 entries of strategy 3.2
- 50 entries of strategy 3.3
- 150 entries of strategy 4 (random allocations)

I think that the most popular strategies will be strategy 1, 2.1 and 3.1 and the least popular strategy will be strategy 3.3. Results from running my program: The strategies that perform the best are 5 trials are:

$$(18, 37, 45, 0, 0, 0, 0, 0, 0, 0,) \quad (18, 39, 43, 0, 0, 0, 0, 0, 0, 0) \quad (17, 35, 47, 0, 0, 0, 0, 0, 0, 0)$$

$$(20, 35, 45, 0, 0, 0, 0, 0, 0, 0) \quad (19, 37, 44, 0, 0, 0, 0, 0, 0, 0)$$

There is a clear pattern in the results: Strategy 1 is performing the best, with allocations of small values(around 18-20) for the first value, and a very large value (close to 45) for the third value.

My opinion is that this is a high risk strategy; of the 1000 strategies in my sample space, less than a quarter of allocations have soldiers at the first castle, and this may or may not be representative of the true sample space.

Looking at the above strategies that perform the best, if people have used a similar reasoning to me to get their solution, then they will submit entries such as the above. Therefore, I will submit an entry that beats the above entries (on most occasions) and performs well against the other entries in the sample space. After trying a few combinations, I arrive at the solution:

(0, 0, 46, 1, 1, 25, 12, 15, 0, 0)

which gives a mean score of 41.4 (out of a maximum of 55) after 5 samples, which turns out to be greater than the mean score of any of the entries in the sample space in the 5 samples.

4 Code for Python program

```
import numpy as np

def randomIntegers():
    x = np.random.randint(100,size=10)
    y=[0,0,0,0,0,0,0,0,0,0]
    randomSum = sum(x)
    for j in range(0,10):
        y[j] = x[j] * 100 / randomSum

    return y

def strategy1():
    x = [np.random.normal(30,6),np.random.normal(33,6),np.random.normal(37,6)]
    y = [0,0,0,0,0,0,0,0,0,0]
    randomSum = sum(x)
    for j in range(0,3):
        y[j] = x[j] * 100 / randomSum

    return y

def strategy2_1():
    x = [0,0,np.random.normal(37,6),np.random.normal(30,6),np.random.normal(33,6)]
    y = [0,0,0,0,0,0,0,0,0,0]
    randomSum = sum(x)
    for j in range(2,5):
        y[j] = x[j] * 100 / randomSum

    return y


def strategy2_2():
    x = [0,np.random.normal(31,6),np.random.normal(37,6),np.random.normal(32,6),0]
    y = [0,0,0,0,0,0,0,0,0,0]
    randomSum = sum(x)
    for j in range(2,5):
        y[j] = x[j] * 100 / randomSum

    return y


def strategy3_1():
    x = [0, 0, np.random.normal(36,7), 0, 0, np.random.normal(33,7),
    np.random.normal(15,3), np.random.normal(16,3), 0, 0]

    y = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    randomSum = sum(x)
    for j in range(0,10):
        y[j] = x[j] * 100 / randomSum

    return y



def strategy3_2():
    x = [0, np.random.normal(33, 6), 0, 0, np.random.normal(36, 6),
    np.random.normal(15, 3), np.random.normal(16, 3), 0, 0, 0]

    y = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    randomSum = sum(x)
    for j in range(0, 10):
        y[j] = x[j] * 100 / randomSum

    return y

def strategy3_3():
    x = [np.random.normal(33, 7), 0, 0, np.random.normal(37, 7), np.random.normal(15, 3),
    np.random.normal(15, 3), 0, 0, 0, 0]

    y = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    randomSum = sum(x)
    for j in range(0, 10):
        y[j] = x[j] * 100 / randomSum

    return y
```

```
JaneStreetSolutions = []


for a in range(0,200):
    JaneStreetSolutions.append(strategy1())

for b in range(0, 200):
    JaneStreetSolutions.append(strategy2_1())

for c in range(0, 80):
    JaneStreetSolutions.append(strategy2_2())

for d in range(0, 200):
    JaneStreetSolutions.append(strategy3_1())

for e in range(0, 120):
    JaneStreetSolutions.append(strategy3_2())

for f in range(0,50):
    JaneStreetSolutions.append(strategy3_3())

for g in range(0,150):
    JaneStreetSolutions.append((randomIntegers()))



Totalscore = []



for i in range(0, len(JaneStreetSolutions)):
    score = 0
    for j in range(0,len(JaneStreetSolutions)):
        k = 0
        player1consecutive = 0
        player2consecutive = 0
        #score = 0
        while (player1consecutive < 4 and player2consecutive < 3 and k < 10):
            if JaneStreetSolutions[i][k] > JaneStreetSolutions[j][k]:
                score += k + 1
                player1consecutive += 1
                player2consecutive = 0

            elif JaneStreetSolutions[i][k] == JaneStreetSolutions[j][k]:
                player1consecutive = 0
                player2consecutive = 0

            else:
                player1consecutive = 0
                player2consecutive += 1
            if player1consecutive == 3:
                for l in range(k+1,10):
                    score += l+1
                player1consecutive = 4

            k += 1
    Totalscore.append(score/len(JaneStreetSolutions))




maximum = Totalscore.index(max(Totalscore))
print("Best strategy in sample space", JaneStreetSolutions[maximum])
print("Maximum score of ", Totalscore[maximum])




mySolution = [0,0,46,1,1,25,12,15,0,0]
print("My solution", mySolution)
myscore = 0
for i in range(0, len(JaneStreetSolutions)):
    k = 0
    player1consecutive = 0
    player2consecutive = 0
    while (player1consecutive < 4 and player2consecutive < 3 and k < 10):
            if mySolution[k] > JaneStreetSolutions[i][k]:
```

```
                myscore += k + 1
                player1consecutive += 1
                player2consecutive = 0

            elif mySolution[k] == JaneStreetSolutions[i][k]:
                player1consecutive = 0
                player2consecutive = 0

            else:
                player1consecutive = 0
                player2consecutive += 1
            if player1consecutive == 3:
                for l in range(k+1,10):
                    myscore += l+1
                player1consecutive = 4

            k += 1

print("My score: ",  myscore/len(JaneStreetSolutions))
```

Sample output:

```
Best strategy in sample space [17.79283602078832, 37.19522596533336, 45.01193801387833, 0, 0, 0, 0, 0, 0, 0]
Maximum score of  39.684
My solution [0, 0, 46, 1, 1, 25, 12, 15, 0, 0]
My score:  41.927
```