Blotto

Lauren Hanlon

Introduction

The goal of this project was to explore strategies related to the game Blotto, and to try and come up with an optimal strategy that will be tested against other players.

To make the game more 21st centry-esque, here are the alternated instructions:

> You and another (equally qualified) candidate are running for office. There are 10 districts, numbered 1, 2, 3, ... 10 and worth 1, 2, 3, ... 10 votes respectively. You have 100 discrete units of resources (e.g. time, campaign workers), which you can allocate between the districts however you wish. Your opponent independently does the same. For each district, whoever applied the most resources to that district wins its votes (in the case of a tie, the indecisive voters abstain and no one receives any votes).

District	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Lauren	10	10	10	10	10	10	10	10	10	10
Opponent	20	0	10	5	10	5	10	20	0	20

> In this match, Lauren wins districts 2, 4, 6, 9, for a total of 21 votes, and her opponent wins districts 1, 8, 10, for a total of 19 votes (no one wins districts 3, 5, 7).

Methodology

My thought process behind coming up with an optimal strategy was to make sure my strategy tested efficiently against a myriad of strategies, including completely random strategies, more optimized strategies as well as popular strategies.

My first set of strategies to test against was a series of random strategies, for example:

```
##  [1]  3  0  0  5  0  0  0  0 61 31
```

```
##  [1]  3  0  0 83  0  0  3 11  0  0
```

```
##  [1]  3 40  0  1  7  4  0 31  3 11
```

As you can tell, this strategy is likely not to be very smart. It often uses up 50+ votes, and doesn't even use them on the larger districts where any normal person would have placed the most amount of resources.

My next series of strategies fixes that error by still choosing random resource numbers, but then sorting the resources so that the highest number of resources goes to the district where they could win the most amount of votes, for example:

```
##  [1]  0  0  0  1  1  3  5 22 27 41
```

```
##  [1]  0  0  0  1  4 13 13 19 19 31
```

```
##  [1]  0  0  0  0  0  0  1  2 12 85
```

As is the case with randomization, sometimes this number going towards district 10 is way too high, and any normal campaign person would not allocate 50+ resources to any one district, so in the next randomization strategy I capped the limit to be 50 for any one district, then still sorted the rersources by district.

```
##  [1]  1  2  2  3  5  7 14 14 16 31
```

```
##  [1]  1  1  2  3  7  7  7 14 25 29
##  [1]  0  2  2  2  3  4  7 16 23 38
```

Strategies

In creating a strategy for myself, I looked at a few different ways to go about this problem. The most obvious choice was an **equal dispersion** technique, allocating `num of votes / num of districts` per district. In this case it would be 10 per district. A second technique included a weighting strategy, where more importance was given to the districts with the **highest rate of return**. In these techniques, I allocated 30+ votes to the tenth district, and then spread out my resources in a tail curve. A third technique included a **bell curve** strategy, where I allocated the majority of my resources to the middle districts, then spread out the rest of the resources along the ends, lowering the amount of resources the further the district was from the middle. The fourth strategy I employed was a **minimalist** strategy, i.e. what were the fewest number of districts I had to win, to make sure that I always won in comparison to the ones I lost. These turned out to be: win districts 10, 9, 8, 7 and you'll always win (since 34>21), or win 8, 7, 6, 5, 4 (30>25) and you'll always win. The strategy here was, how can you minimize the number of districts you need to allocate your resources to. The last strategy I employed was a **camel** strategy, where I used a double bell curve to allocate my rescources.

For each of these strategies, I created a few variations of them to find the most optimal strategy.

Testing my strategies

To test each strategy against my "randomized" opponents, I wanted a way to test the efficiency of each strategy. To do this, I created a function that would generate 1000 random strategies (or however many you so choose), then would test your strategy against each of those random strategies, then create an efficiency score that would give you the percentage of times your strategy won.

For example, let's use the equal dispersion strategy against our randomized weighted strategy (the one that computes random numbers, but then sorts them and allocates the maximum number of resources to the districts with higher returns)

```
myStrategy <- c(10, 10, 10, 10, 10, 10, 10, 10, 10, 10)
efficiencyScore <- calculateEfficiency(myStrategy, strategy=randomStrategy_Smart)
efficiencyScore
```

```
## [1] 0.885
```

To account for a variety of techniques that will be tested against my strategy, I wanted to create a way to give weights to each of the random strategies I created. As such, I created a testing function that calculated:

efficiencyScore = (completely random strategy) x 0.2 + (random strategy with sorting) x 0.3 + (random strategy with sorting and cap) x 0.5

I found these weights to be a fair representation of the types of strategies mine would be tested against, considering the natural human inclination is to allocate the most amount of resources to the districts with the highest returns.

Results

Here are the results of my strategies:

```
##               strategy efficiency
## 1      equalDispersion     0.7657
## 2         weightedEnd1     0.0333
```

```
## 3         weightedEnd2     0.0960
## 4         weightedEnd3     0.3650
## 5         weightedEnd4     0.6207
## 6         weightedEnd5     0.6319
## 7         weightedEnd6     0.8404
## 8         weightedEnd7     0.7939
## 9           bellCurve1     0.8453
## 10          bellCurve2     0.8156
## 11  minimalistStrategy     0.9101
## 12 minimalistStrategy1     0.9704
## 13 minimalistStrategy2     0.4006
## 14 minimalistStrategy3     0.4814
## 15 minimalistStrategy4     0.5771
## 16       camelStrategy     0.5117
## 17      camelStrategy1     0.6833
```

I ran this a few times to make sure the efficiency scores stayed accurate when different random strategies were created, and I found that the error term was consistently <0.02, which was sufficient for this part of the exercise, and running the tests with more random sampling just created unnecessary time.

From these results, we can clearly see that our `weightedEnd1`, `weightedEnd2`, `weightedEnd3`, `minimalistStrategy2` and `minimalistStrategy3` received the lowest efficiency ratings. For context the strategies are shown below

District	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
weightedEnd1	0	0	0	0	0	0	0	0	0	100
weightedEnd2	0	0	0	0	0	0	0	0	50	50
weightedEnd3	0	0	0	0	0	0	0	20	30	50
minimalistStrategy2	0	0	0	0	0	0	25	25	25	25
minimalistStrategy3	0	0	0	0	0	0	20	25	25	30

Obviously, one would never have wanted to use those weighted end strategies in real life, because it banks on your opponent either allocating all their resources in the same way, give or take a district or two, which just isn't likely. I included these more for testing validation and variety than as actual potentials for an optimal strategy. Same rationality with the minimalist strategies. They rely on your opponent choosing to allocate 0 resources to the lower districts, and then winning marginally in the larger districts.

We now want to shift our focus to the higher performing strategies. Strategies that consisently ranked >0.8 in efficiency were `weightedEnd6`, `weightedEnd7`, `bellCurve1`, `bellCurve2`, `minimalistStrategy`, `minimalistStrategy1`. The strategies were as follows:

District	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
weightedEnd6	0	0	0	0	0	10	15	20	25	30
weightedEnd7	1	1	1	1	1	5	15	20	25	30
bellCurve1	1	4	5	20	20	20	20	5	4	1
bellCurve2	1	4	5	15	25	25	15	5	4	1
minimalistStrategy	0	0	0	20	20	20	20	0	0	0
minimalistStrategy1	1	1	1	17	18	20	20	20	1	1

To find the optimal strategy, I turn away from testing against randomized strategies and I simply used a tournament style techinque to determine the optimal strategy out of these top-performers. A 1 indicates that the first strategy won, and a 0 indicatest that the second strategy won.

```
## [1] 0
```

```
## [1] 0
```

```
## [1] 0
```

I first tested the weighted strategies, bell curve strategies and minimalist strategies against one another, then I tested those winning strategies against one another.

```
winner(weightedEnd7, bellCurve2) #weighted
```

```
## [1] 1
```

```
winner(weightedEnd7, minimalistStrategy1) #minimalist
```

```
## [1] 0
```

```
winner(bellCurve2, minimalistStrategy1) #bell curve
```

```
## [1] 1
```

```
winner(bellCurve2, weightedEnd7) #weighted
```

```
## [1] 0
```

```
winner(minimalistStrategy1, weightedEnd7) #minimalist
```

```
## [1] 1
```

```
winner(minimalistStrategy1, bellCurve2) #bellcurve
```

```
## [1] 0
```

As we can see from these results, each strategy wins against each other and there is no true conclusive winner, so in my opinion all of these strategies are intelligent and strategic. However, there must be a winner, so I compared all three strategies against each other, and decided that my final allocation of resources will be:

District	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Lauren	1	1	1	17	18	20	20	20	1	1

Conclusion

In conclusion, I ended up using a minimalist strategy, since it tested well against my randomized strategies, and was competive against the other strategies I came up with. The minimalist strategy was my initial thought going into this project, since if you can concentrate your resources into a small number of districts, then you're able to focus primarily on those ones, win them, then the other districts won't matter *as much* since you'll have reached your minimum number of districts you need to win, in this case the districts I wanted to win were 4, 5, 6, 7, 8 for a total of 30 votes. This intuitively makes sense that I would allocate my resources into these districts, since they maximize my returns *and* the majority of competitors will put the majority of their resources into the higher districts (or so I would think they would - I could be wrong:) The only thing that concerns me with this strategy is district 8 - my opponent might win this district, in which case I would need to win districts 1-7 in order to win (or tie in a district). Nevertheless, I'm confident in my strategy and am curious as to how others decided to play this game!

Last Remarks: if decimals were allowed

If decimals were allowed, I could redo this analysis, using `runif(1, 1, 100)` instead of `sample(1:100, 1)` in creating random strategies. After giving this some thought, I decided that my strategy would not change - this being because my initial thought was to increase each district by 0.1, so as to account for making sure I win it in case there was a tie. But this strategy 1) ruins the beauty of my minimalist strategy, in allocating resources to just the right amount of districts, and 2) if we think along the lines of our opponent increasing each of their resources by 0.1 by the same logic, we can continue this thinking so as to increment it by 0.2, to make sure we don't end up at a tie again. Each district then increase by a factor of 0.1 every time we have this thought, and limits at the next highest integer. In conclusion, I have faith that my strategy is just fine without increasing/decreasing any district by a decimal number of resources.