Colonel Blotto Game

Lukasz Jakubowski

January 2020

1 Abstract

The following report includes short theoretical introduction and methodology in arriving at the result, including the use of Genetic Algorithm (GA). The words *strategy* and *submission* are used interchangeably throughout the report.

2 Search space estimation and theoretical approach

For simplicity, I assumed that a player will always distribute all of their soldiers. To justify this approach, let us consider any possible entry, adding one soldier to any of the castles will not worsen the situation, the player can either keep or increase the score against any opponent. It follows inductively that any number of extra soldiers does not decrease the player's average score. Therefore, it is optimal to use 100 soldiers.

The number of possible submissions can be calculated by spotting self-similarity as the number of castles increases. Let n be the army size and k – number of castles. For $k = 1$ there is 1 option (all soldiers in this castle). For $k = 2$, $n+1$ options (number of soldiers in the first castle strictly determines the number in the second, we can choose from 0 to 100 in the first castle). For $k = 3$, we put m soldiers in the first castle, and then we have $(n–m)$ soldiers to distribute in 2 castles (recursive case). We add options corresponding to m values from 0 to 100. This is given by the sum $\frac{1}{2}(n+1)(n+2)$. In general case, the number of possible submissions is given by the formula:

$$N = \frac{(n+1)(n+2)\dots(n+k-1)}{(k-1)!}$$

For our problem, there are $N = 4.3 \cdot 10^{12}$ possible distributions of soldiers. To decide which entry is the best, we would have to play roughly $N^2 = 10^{25}$ games, which is not trackable, with average CPU performing 10^9 operations per second. Limiting the search space is essential.

From the game theory perspective, this variation of Blotto is a zero-sum, non-symmetric game. The game is not symmetric, as winning 3 castles in a row grants winning the remaining ones (this is later called *combos*). Nevertheless, the game has an important feature of the classical Blotto [**1**] (without combos, all castles worth an equal number of points):

For every (fixed, no mixed strategy allowed) strategy, there is a strategy which wins with it.

Which tells us that there is *no* strategy which will grant us wins or draws only. This is especially important in the case of the contest, only a small proportion of all possible submissions will play against each other, so even if there is a strategy that gets the most points when playing against the whole search space (might be several such strategies), it is likely not to be optimal when playing against the small part of it.

Following this conclusion, I assumed to divide strategies into *good* and *bad* ones. A bad strategy performs poorly, no matter what portion of search space plays with it. An example might be putting 100 soldiers in the first castle, and 0 in the rest of them, this strategy never wins (it is the worst strategy). A good strategy is not a bad strategy. The division has a qualitative meaning, as there is no clear border between the two.

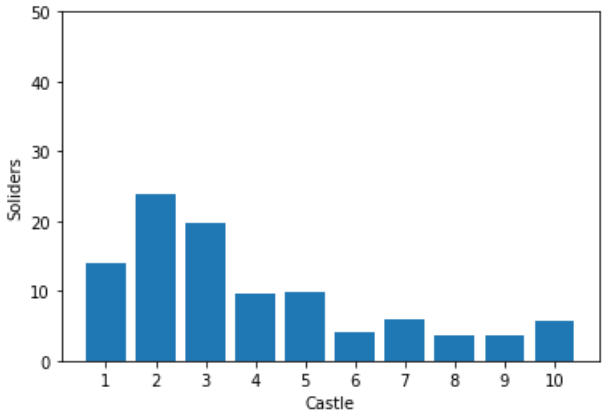
The first aim of my investigation is to assure that my submission is not a bad strategy. With that, it is a psychological question what other players will submit. I decided to use a genetic algorithm as a heuristic which allows for an educated guess.

3 Computational Approach (*Python*)

Initially, my simulations included a generation of (uniformly) random submissions and playing them against each other. A score of player A against player B was calculated as a difference of the points they received, allowing to differentiate poor strategies (otherwise all of them would receive a net score of 0). The strategies were then sorted by the net number of points in all games. The average of the top 10 strategies was plotted on a bar chart.

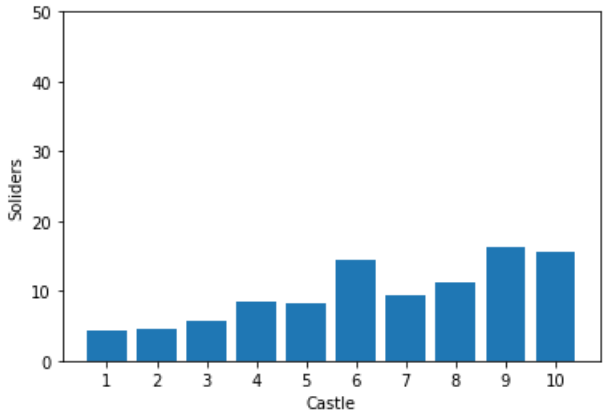
I have implemented different versions of the game ((I) original, (II) with all castles worth 1 point and (III) without combos). As (I) is a mixture of (II) and (III), I tried to extract useful information from analysing the simplified games and apply it as a mixture. Playing 10 000 entries against each other gave a relatively stable outcome, presented on the charts:

Game (II) bar chart



As we can see, (II) pushes the distribution toward the first 3 castles, whereas (III) - distributes soldiers in castles concerning the number of points they are worth. Unfortunately, the two simplified games have opposite impacts on the original game (I), so I abandoned considering them further.

Game (III) bar chart



Playing 10000 against each other in (I) did not give consistent results, yet allowed for some conclusions:

1. Putting all/great majority soldiers in 1-2 castles is a bad strategy. (no use of combos).

2. Putting many (around 25) soldiers in the 3rd castle increases the chances of a win when playing against randomly generated trials.

3. Castles from 5 on are less important (around 5 soldiers each).

Surprisingly, a uniform distribution of soldiers (10 in each castle) was usually in the top 3, somehow contradictory to the conclusions.

Additionally, my implementation allowed for the generation of nonuniform random submissions e.g. [0.33, 0.33, 0.34, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] would distribute soldiers uniformly only in the first 3 castles. This gave me a broad outlook how particular submissions perform in different parts of search space. In environment distributing most soldiers in the first 3 castles, (33, 33, 34, 0, 0, 0, 0, 0, 0, 0) usually received the highest number of points.

4 Genetic Algorithm (GA)

Inability to infer any interesting strategies encouraged me to formulate the problem in terms of simplified genetic algorithm (GA). In terms of GA, a submission is a chromosome, a number of soldiers in a particular castle - a gene. Chromosomes reproduce by taking the average of soldiers in a castle, choosing randomly whether non-integer results should be rounded in up-down-up-... or down-up-down-... manner. This method of reproduction is different from conventionally used in GAs but assures that children have a total of 100 soldiers. Mutations were not introduced in the algorithm, but 10% of next-generation chromosomes was generated at random (ie. not by reproduction). Top 10% of performers were passed onto the next generation and top 30% of parent generation was allowed to mate, to produce 80% of the next generation. There was a maximum of 10 generations. The first generation was generated randomly. Algorithm evaluation proceeded in two ways:

1. Fitness of individuals is evaluated by the performance against a group of randomly generated opponents (usually 10 000).

2. Two GA were competing against each other, new generation A is computed based on generation B, then new generation B is computed based on new generation A.

Method (1) proved to be a useful tool in generating highly specialised generation, that would almost always win against randomly generated opponents (including biased ones).
Method (2) solidified the claim that every strategy has a winning strategy, with the parameters set as above, neither of the algorithms lost in the long term e.g. as soon as player A became specialised in winning against the dominating gene pattern in player B, random part of the generation managed to recover it from the losing state.

5 Summary

The (1) GA enabled me to create specialised generations, designed to win with a particular randomly generated group of opponents. I started with producing a generation well designed for defeating uniformly random opponents. Then I created a bias based on the top 10 individuals from the population. Similarly, I created a generation that could beat the one generated from the previous bias. I repeated the process until the produced generation did not escalate the number of soldiers in the 3rd castle, at this point, results started to be inconsistent (the majority of soldiers started to be distributed in castles 1 and 2, but with no obvious pattern). I then took the mixture of all previous biases in equal amounts, and created a strategy which beats them all.
In addition, I decided not to put any soldiers in castle 2 and 7 (there was 0% that a solider in an initial population will be distributed in these castles). This increased the score, as the algorithm redistributed soldiers from castle 2 to castle 1 (in order to break the combo in the first 3 castles). Castle 7 proved to be not important. The solution underestimated the importance of castle 6, which is essential to strike a combo on castles 4,5 and 6, so I slightly increased the number of soldiers in it (swapped the numbers in the 5th and 6th castle).

This resulted in a strategy equal to:

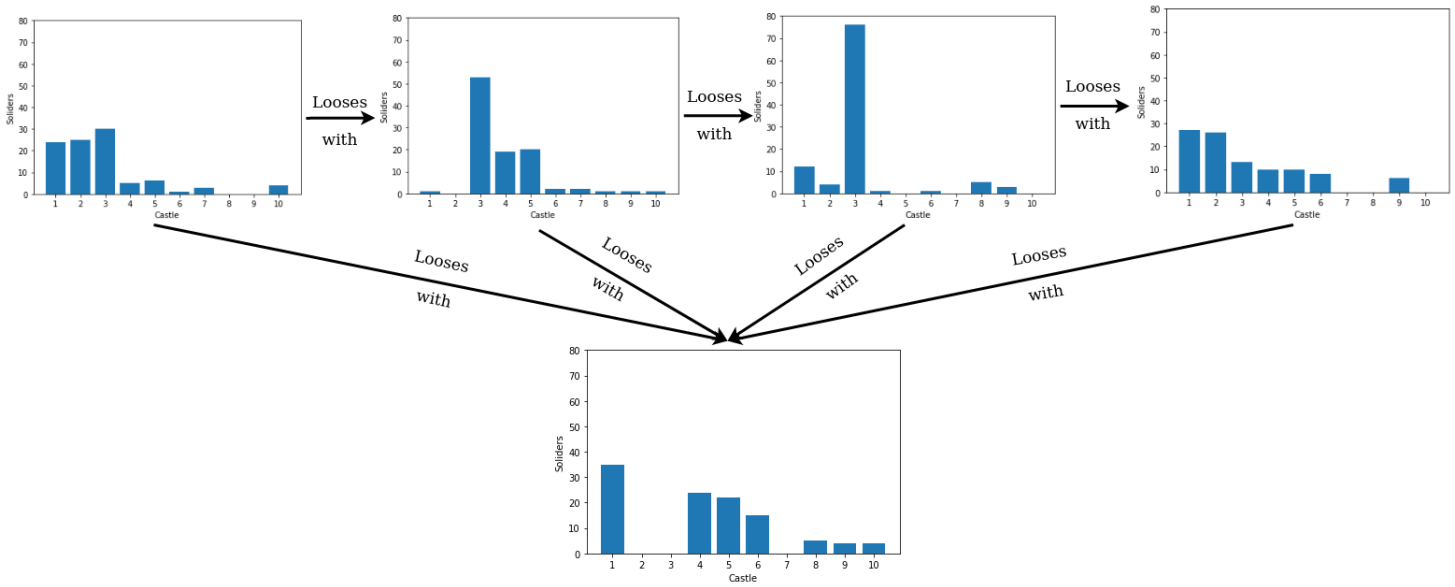
[35, 0, 0, 24, 15, 22, 0, 5, 4, 4]

The diagram on the following page explains how the solution was created.
Python code can be accessed through the following link, the code was not shared with any other contestants:

Python Code

6 Bibliography

[1] Golman R, Page SE. General Blotto: games of allocative strategic mismatch. Public Choice. 2009 Mar 1;138(3-4):279-99.


Looses with
Looses with
Looses with
Looses with
Looses with
Looses with
Looses with
Looses with
Soliders
Castle