Imperial College London

Dynamics of learning and iterated games

Imperial College London

Department of Mathematics

Project 3: The Blotto Game

Author:
Lucjano Muhametaj

October 28, 2022

Abstract

This is the project 3 for the module of Dynamics of Learning and Iterated Games. This project considers regret learning in the context of the Blotto games, which often is associated to two candidates for an election, who each can allocate money in different states.

Contents

Chapter 1

Introduction

This project deals with the Blotto Game in many contexts and in many forms and one of the main purposes is to show how flexible it is and how little changes can give the possibility to this problem to constitute a model which can easily represent real life situations.

1.1 Objectives

Is important to note that being very flexible, this game presents many versions and many of them for great number of players and great number of so called 'fields' present a very large number of 'states' (or 'actions'), maybe infinite, which makes the analysis of this problem a really interesting topic in Game Theory research nowadays.

1.2 Challenges

One of the main points of studying Blotto game is searching for the so called Nash Equilibria, which will be defined later in this work. In some cases, like the continuous version a study of Nash Equilibria we will use slightly more advanced instruments.

1.3 Structure and contributions

In the project we present the Blotto game in the discrete version, we give some definitions of some instruments from game theory, which we use to approach the problem. Then we will deal with a computational approach which is based on a regret minimization learning algorithm.
After that we will approach a different version of the Blotto Game, which in contrast with the first one is not discrete anymore and presents some limitations on the possible way a player can choose to play (which will be represented by a limitation over the budgets of the players).
Finally in the mastery component we analyse the resilience of the Regret based algorithm under the influence of external events, influencing the results over each battlefield.
Before starting is important to note that we will always deal with a game with only two players.

Chapter 2

A quick explanation of the Blotto Game

The colonel Blotto is a game with two players involving:

1. a number of objects S (which correspond to the soldiers in the original game),
2. a number of places N (which correspond to the number of battlefields in the original game)

The game involves a simultaneous choice of the M players. We will take $M = 2$ (called in [TWN13] as Colonel Blotto and Boba Fett).
Those players choose to locate a number of soldiers (or objects) for each battlefield, such that given $S_1, S_2, ..., S_N$ the number of soldiers assigned to each battlefield by one of the players (where S_i could also be 0), then $\sum_{i=1}^{N} S_i = S$. We in particular also will assume $N < S$.

For each battlefield we say that the battle is locally won by the player with more soldier on the battlefield. So for example if one player disposes 3 soldiers in the first battlefield and the second only 1, we can say that the first player has won the local battle in this field. The purpose of the game for a player (at which we will refer as 'winning the war') is not to win 'locally', but to to win more battles (in other words to conquer more battlefields) than the other player.

We will indicate each player's choice with arrays of N elements where the i^{th} element corresponds to S_i. So a players action can be seen as $x \in \mathbb{Z}_+^N$, such that $x_i = S_i$ and $\sum_{i=1}^{N} x_i = S$.
Let's make an example to make it clearer. Imagine to have $S = 5$ and $N = 3$. So in this case each player when plays will produce a vector of three element depending on his choice. Let's suppose Colonel Blotto player plays $(1, 3, 1)$ and Boba Fett on the other hand $(3, 0, 2)$. We can observe that Boba Fett has won the battles on the first and on the third battlefield, while Blotto has won in the second battlefield. So Boba Fett has obtained 2 victories, while Blotto only won locally once so we are going to say that Boba Fett is the final winner.

Let's finally notice that both the game both a single battlefield match can hypothetically end with a draw.

Chapter 3

The Blotto Game with integer allocations and no limitations over the resources

3.1 Premises

Before going on we will make some assumption:

1. the first one is that both players are going to use all their soldiers,
2. the second one is that both players have the same number of soldiers

The first assumption makes sense because we are not imposing limitations over the 'budgets' as we will do later.
For now let's notice that this case is the only one that gives sense to a game with 2 battlefields. In fact with two players with the same number of soldiers and only two battlefields the result is always going to be a draw (because if player A wins on the first battlefield, he will have for sure less soldiers on the second one, so player B is going to win on that battlefield, with a draw as a final result. Otherwise we can have a draw on both battlefields, which is a final draw).

So now we can see that this game is not described in a matrix form, so we want to understand how to convert it in that way. To do this we have to determine the number of possible actions for each player.

3.2 Possible number of action for player

Firstly we observe that for each player we have $\binom{S+N-1}{N-1}$ possible arrays (we remember that we use arrays to indicate an action and more in particular the number of the soldiers for each battlefield). It's easy to obtaining this using a simple combinatorics trick.

3.2.1 Explaination

We observe that we want to find all the ways to redistribute S elements in N spaces.
This is equivalent to finding the number of the anagrams of a 'word' composed by S x and $N-1$ $|$.

To explain this let's see and example: taken a 'word' $xx|x||xxx|$ we interpret the $|$ as dividers between a battlefield and another one (noticing that for N fields we need $N-1$ dividers) and the x can be interpreted as soldiers. So we will have 5 fields and 6 soldiers and the action array is $(2, 1, 0, 3, 0)$.

On the other hand for example $xxxx|xx|||$ is $(4, 2, 0, 0, 0)$.

We notice that this way to count the strategies counts the cases of battlefields with zero soldiers and also counts separately symmetric strategies (or in general strategies with the same number of distribution of soldiers but in different battlefields) because we want to study those situations separately.

3.3 Matrix Interpretation

Now we can see that we can represent this game as a sort of multidimensional matrix. In fact we can exploit what we obtained on the number of the strategies to say that to represent the game it's enough to write the payoff matrix A, with dimension $\binom{S+N-1}{N-1}\text{x}\binom{S+N-1}{N-1}$.

Giving to the first player the row entries and to the other one the column entries we are going to have that each column entry is an action for the second player and each row an action for the first one. Given an intersection of a row and a column (for example row i and column j, with $1 \leq i, j \leq \binom{S+N-1}{N-1}$) we have that player 1 plays action i and player 2 plays action j (we are not referring to any strategy in particular, so i and j depend purely on the order we gave to the actions in the entries). So in $A_{i,j}$ we will have $(1, -1)$ if the first player wins over the second, $(-1, 1)$ if the second wins over the first and then finally $(0, 0)$ if there is a local draw.

Let's see what happens in an easy case, with $N = 3$ and $S = 3$ (Note that we want $S > N$ in the Blotto game, but here I choose them equal to make it easier to write the matrix, which is going to be giant otherwise).

	(0,0,3)	(0,3,0)	(3,0,0)	(0,1,2)	(0,2,1)	(1,0,2)	(2,0,1)	(1,2,0)	(2,1,0)	(1,1,1)
(0,0,3)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(-1,1)	(-1,1)	(-1,1)
(0,3,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(-1,1)	(-1,1)	(0,0)	(0,0)	(-1,1)
(3,0,0)	...	...	...	...	...	...	...	...	...	...
(0,1,2)	...	...	...	...	...	...	...	...	...	...
(0,2,1)	...	...	...	...	...	...	...	...	...	...
(1,0,2)	...	...	...	...	...	...	...	...	...	...
(2,0,1)	...	...	...	...	...	...	...	...	...	...
(1,2,0)	...	...	...	...	...	...	...	...	...	...
(2,1,0)	...	...	...	...	...	...	...	...	...	...
(1,1,1)	...	...	...	...	...	...	...	...	...	...

(3.1)

I wrote only the first two rows but is easy to continue.
In this case we can see for example that if player 1 chooses the action $(0, 3, 0)$ and player 2 chooses the action $(2, 1, 0)$ we are going to have the second player winning the battle on the first field, the first player will win in the second battlefield. On the third battlefield we have a local draw, so at the end with a local victory for each player and a local draw we are going to have a draw for this game.

Let's observe that we are giving rewards and loss basing our analysis only on the final result for the game. However we can do a slightly different analysis studying each battlefield separately and computing rewards or losses for each one of them. At the end summing rewards and losses for each battlefield the player with the highest $|rewards - losses|$ wins.

Approaching the game this way does not make a lot of difference for us, but working in this way makes it easier to deal with a possible case where different battlefields have different weight (which however is not our case).

Chapter 4

Nash Equilibria for the Blotto Game with integer allocations

In this chapter we start to discuss about Nash Equilibrium for this game and in particular we discuss the case when the allocation to each battle field is an integer. To proceed with the analysis of this case it's first necessary to give some definitions and to talk about pure and mixed strategies and about pure and mixed strategy Nash Equilibrium. I will refer to the lecture notes and to 'An Introduction to Counterfactual Regret Minimization' from Todd W. Neller and Marc Lanctot [TWN13].

4.1 Definitions

The Blotto Game is a zero-sum game. This means that taken the utility vectors (that we used to put together the matrix) the values of each utility vector sum is 0.

Definition 1. *We say that a player uses a pure strategy if he chooses a single action with probability 1. On the other hand we say that he plays with mixed strategy if the player chooses from at least two different actions, with both of them with probability* > 0.

So in a discrete game a mixed strategy is a probability vector, a vector $y \in \mathbb{R}^m$ such that the sum of the components of the vector is 1. A pure strategy is a base vector e_j, with all components equal to zero except from the component j. Let's note that m is the number of possible actions.
As we saw, the Blotto problem an 'action' is the choice of a possible distribution of soldiers that can be uniquely determined by a vector of N elements.
So basically an action is a choice of a vector of length N and a strategy is a vector choosing for sure one or is mixed, randomly more than one action.
To make it even more clear, if we have a strategy $y = (y_1, y_2, ...)$, we are choosing the first action with probability y_1, the second with probability y_2...

Definition 2. *A strategy is a best response strategy for a player if given all the other strategies, maximizes the expected utility for the player.*

Definition 3. *When every player plays with a best response strategy to each of the other player's strategies then the combination of the strategies is called a Nash equilibrium.*

There exist both pure and mixed strategies forms of Nash equilibrium. In particular a mixed strategy Nash means that at least one of the players is using a 'stochastic approach' (I mean that this idea involves some randomness). It is important to note that in this case we deal with the so called expected utility (which is the expected value of the utility/payoff)

We can see a Nash Equilibrium as a couple of probability vectors where each entry is associated to one of the possible actions.

Notice that there is not a natural order for the actions since we can't order not uni-dimensional vectors, but we can choose an order at priori.

4.2 Case $N = 2$

As written before for $N = 2$ if we have two players with the same number of soldiers and only two battlefields the result is always going to be a draw.

This is because if player A wins on the first battlefield, he will have for sure less soldiers on the second one, so player B is going to win on that battlefield, with a draw as a final result. Otherwise we can have a draw on both battlefields, which is a final draw. So in this case all possible combinations of strategies of the two players are Nash Equilibria.

4.3 Case $N > 2$, pure strategy

Still remaining in the situation when the two players have the same number of soldiers and supposing $N > 2$ many situation can occur. Firstly it's easy to notice the following result:

Lemma 1. *In the Blotto game with 2 players and same number of soldiers S, the two players can always draw. In particular for each action chosen by a player there exists an action for the other player that gives a draw as a final result of the game.*

Proof. The proof is trivial. In fact it's easy to observe that a symmetric behaviour (so a choice of the same action vector from both the players) is always possible and that it gives as a result a local draw on each battlefield and as a consequence a final draw. □

From this lemma we can obtain the following corollary:

Corollary 1. *Every couple (x, y) of vectors indicating a Nash Equilibrium in pure strategy has as utility (0,0) (which means the result is a draw).*

Proof. In pure strategy we can write $x = e_i$ and $y = e_j$. So if (e_i, e_j) is a Nash equilibrium both players will play with probability 1 the actions i and j respectively obtaining the utility $u_{i,j}$.
We know that $u_{i,j}$ could be $(0, 0)$, $(1, -1)$ and $(-1, 1)$. Let's suppose that $u_{i,j} = (1, -1)$ is the utility we get in the Nash equilibrium condition (the case $(-1, 1)$ is basically the same), so the second player is losing and since we have a Nash Equilibrium he should have no incentive to unilaterally deviate from the pure strategy. However for the previous lemma it's easy to observe that for the second player there exists an action, and then a pure strategy, that can make him draw. So (e_i, e_j) was not a Nash Equilibrium, since there are better options. We have a contradiction.
So at the end a Nash Equilibrium implies a draw. □

4.3.1 Case $N = S$

We are assuming $S > N$, but it is interesting to see this case.
For $N = S$, if both players take the vector (1,1,...,1) composed by N ones (which exists since N=S) we have a Nash Equilibrium (with the players both using a pure strategy).

4.3.2 Case $S > N$

Lemma 2. *If $S > N$ there is not pure strategy Nash equilibrium.*

Proof. We can show it proving that for $S > N$ for each action of a player there is always a possible action in the other player's range of action that gives him the possibility to win.
Using the notations given above each possible action for a player can be written as a vector $(S_1, S_2, ..., S_n)$. If $S > N$ for the Pigeonhole principle there exist a battlefield i such that $S_i \geq 2$.

Let's now suppose without loss of generality that one of the players is playing $(S_1, S_2, S_3, ..., S_n)$ with $S_1 \geq 2$. Now we can see that the distribution $(S_1 - 2, S_2 + 1, S_3 + 1, ..., S_n)$ wins over $(S_1, S_2, ..., S_n)$. In fact if the other player decides to distribute is players like that he is going to lose on the first battlefield, to win on the second and on the third and to draw on the other ones. So he will have 2 local victories, a local loss and the rest are going to be draws.
So we can say that since a Nash Equilibrium has to be a draw and in this case a player has the possibility to win, he has an incentive to move from the hypothetical Nash Equilibrium strategy. □

So it's easy to conclude that for $S > N$ in the starting condition we gave for the Blotto game there are not pure strategy Nash Equilibria.

4.4 Case $N > 2$, mixed strategy

For what we have seen in the previous sections to prove that a Nash Equilibrium exists always for the Blotto Game in our conditions we have to prove that a mixed strategy Nash Equilibrium exists, since in same cases a pure strategy equilibrium does not.

Taking directly this result from the original Nash paper [Nas51] we have the following theorem:

Theorem 1. *For each finite game there exist a Nash equilibrium*

We note that with finite game Nash refers to a game with finite actions sets. In particular in the Blotto game with integer number of soldiers for each battlefield we have a number of actions equal to $\binom{N+S-1}{S-1}$ and since N and S are finite we will have a finite number of actions. So we will have the existence of a Nash Equilibrium, which will be a mixed strategy Nash Equilibrium.

4.5 Other observations

We can notice that this game can have some slight modifications which can completely change everything. For example we can work on a Blotto game where each battlefield has to have at least 1 soldier. In this case the result we got about the pure strategy Nash Equilibria can change but at the end we will still have the existence of at least one Nash Equilibrium (as proved in 4.4).

We can also work on other versions, like for example a Blotto version without any drawing possibility on each battlefield. To do something like that we can invent a Blotto game where in case of the same number of soldiers the winning player on the field is decided randomly (for example 1/2 probability for the first and 1/2 for the second to win on a drawing battlefield).

Finally we can also work on Blotto games with different numbers of soldiers between the players or a game allowing not integers distributions in the 'fields'. In these ways we can formulate different kind of problems which can become models of different real world situations.

Chapter 5

A modified version of the Blotto Game: Budget Constrains

In this chapter we will analyse a modified version of the game encountered in the previous chapters and we will follow the ideas of the paper D. Kvasov, Contests with limited resources, Journal of Economic Theory 136 (2007) 738 – 748 [Kva07], which is developed in the context of limited resources.

In this case the payoff of each player depends also from the number of soldiers involved, and this gives sense to strategies based on the usage of only a part of the total number of the soldiers. Also Kvasov in the paper talks about resources instead of soldiers, this makes sense in particular (if we try to interpret the math into real world applications) when we have continuous allocation of objects/resources over each battlefield (we will keep calling them battlefields to avoid more confusion about terminology, but we could for example talk about investment on objects) instead of dealing only with integers.

5.1 Explanation of the game

In this section we will explain the game introduced by Kavsov. As in the previous cases we will have two players (to deal easily with them in the project they will be called player a and player b, or in more general context i and $-i$), with budgets B_1 and B_2, where $B_1 = B_2 = B$ and N fields in which to distribute the budgets.

Each field could have different values v_j, but we will suppose that all of them have the same value v.

Definition 4. *We define as an action for the player i, with $i \in \{a, b\}$ vector $x_i = (x_1^i, x_2^i, ..., x_N^i)$, with x_j^i the part of the budget allocated by player i in the field j and such that $x_1^i + x_2^i + ... + x_N^i \leq B$*

So the first difference we can note is that in the previous model we had the sum equal to B.

5.1.1 Payoffs and pure and mixed strategies

Given a game and two actions x^i and x^{-i}, the payoff for the player i is defined as follow:

$$Payoff(x^i) = \sum_{j:x_j^i \geq x_j^{-i}} v - \sum_j x_j^i \tag{5.1}$$

Note that in the constant-game version we saw before we don't have budget constrains, so the payoff will be just

$$Payoff(x^i) = \sum_{j:x_j^i \geq x_j^{-i}} v. \tag{5.2}$$

In this case it is easier to just count the number of victories and give 1 to the player with more victories and -1 to the other one, as we did previousky.

We will follow the definitions of pure and mixed strategy we gave before. In particular a player chooses a pure strategy if he chooses a single action with probability 1 and we say that he chooses a mixed strategy if he chooses from at least two different actions, with both of them with probability > 0.

So the pure strategies are in bijection to all the possible different actions if a player chooses only one of them with probability 1. So the number of strategies will be the same as the of the number of the vectors x^i such that $x_1^i + x_2^i + ... + x_N^i \leq B$.

Let's note that we do not have a limitation on the typology of allocations like in the previous chapter, ie we are not working with integer allocations. We are dealing with real allocations, in other words $x_j^i \in \mathbb{R}$ for both players and each field j. So in particular we will have that each action is a vector $x^i \in \mathbb{R}^N$. In particular the family of these vectors (so the possible actions), that now on we will call Z^i, is composed by an infinite number of elements. For this reason since there is a bijection between the pure strategies and the actions, the number of the pure strategies is infinite.

Before going on let's focus for a moment on the assumption we made of real number based allocations. It is possible to avoid studying the integer case, but still for example remain in $\mathbb{Q}$, but it is easier to deal with the real case. Also even in real life applications this constitute a good model even if not strictly realistic. This is the reason we will focus only on the real case on this chapter and not in slight modifications.

The existence of infinite possible actions implies not only that the pure strategies are infinite, but also the mixed ones will be so. We observe also that previously we defined the Nash equilibria of a game of two players mathematically as a couple of probability vectors, where for pure strategies we had the basis vectors e_j. Even if we will deal with the same definition of Nash equilibrium (every player has no incentive in changing strategy), seeing it as a couple of probability vectors with finite number of components is a limitation.

Let's first see that a mixed strategy for a player i can be seen as a N-variate joint distribution G_i from the space of the possible actions Z^i to $[0, 1]$. This follows naturally from the existence of infinite mixed strategies, from the continuous allocations and from the definition itself of mixed strategy. In fact, observe that this means that to each action we are giving a value in $[0, 1]$, so we are choosing between different actions in 'probability'.
In particular we will define the N-variate probability density function as $g_i : Z^i \to \mathbb{R}$ and the 1-dim marginal distribution functions of G_i as $G_{ij} : [0, B] \to [0, 1]$, which given a possible investment over a field (which is going to be between 0 and B) gives back the probability of having invested that amount of resources. We could define it as follow:

$$G_{ij}(x) = \int_0^x (\int_0^B ... \int_0^B g_i(z)dz_j)dz. \tag{5.3}$$

This is the distribution function of the bids or investments of the player i for the field j. (Obviously we can also define the corresponding density functions $g_{ij} : [0, B] \to \mathbb{R}$).
Note also that in case of different resources we can use B_i instead of B, where B_i represents the resources for the player i.

Defining the mixed strategies as probability distributions is quite natural since they were finite probability vectors in the discrete case.

Now we observe that a Nash Equilibrium in this game is going to be a couple of N-variate joint distributions (G_i, G_{-i}).

Now we can note that we have not talked about draws in general and local draws in the battlefields.

We note that since we have continuous payoffs, with two player with identical budgets the probability of having a draw (so two identical payoffs) is 0.
As a consequence to this also the study of the local draws on the fields becomes meaningless and how we choose to deal with with these cases does not affect the final result. We can give a battlefield payoff $v_j = 0$ to each player or otherwise we can choose with probability 1/2 to which player give the payoff v, as suggested in the Kvasov paper [Kva07].

5.2 Observation regarding the study of the Equilibria in the proposed game

In this section we will focus on some general result regarding the Nash equilibria in this game.

5.2.1 Non Existence of Pure Strategy Nash Equilibria

Even if we cannot base our results on the previous chapters, where we based part of the study of pure strategies on a lemma asserting the existence of actions leading to a draw, we can still say that there are not pure strategies equilibria. Moreover in this case since we are not dealing only with integers and since in some cases the players are not promoted to use all their budget the non existence of equilibria is true for each budget B and each number N of battlefield (≥ 2 obviously).

To see the non existence of equilibria we will make three observations:

1. Investing a much higher budget than the opponent on a field is not a wise choice for a player. In fact the player can win on that battlefield with a lower portion of the budget invested, still if the budget is at least greater than the one invested by the opponent.

2. If player i has invested in total less than the other player he can still invest the difference on determinate fields and may have the possibility to win

3. Finally if both player are obliged to invest the same budget and do not want (or cannot) invest more, it's easy to show that the players have better strategies, following which is easy to overcome the opponent.
   It's enough to copy the other player strategy, and since we are dealing with pure strategy we mean basically copying the other player action. Then it is enough to choose a field with non-zero investment on it, without loss of generality let's say field 1. Let's suppose this field has an investment $h \leq B$, then the player can add an investment $\frac{h}{N-1}$ to each other field and leave a 0 investment on the first one. In this way the player wins on $N-1$ battlefields and looses on 1.

5.2.2 Existence of Mixed Strategy Equilibria in terms of margins

We start with an analysis of the marginal distributions associated with the joint distributions representing the mixed strategies. This analysis will give us enough information to state an existence result of the mixed strategy equilibria.

We are working on mixed strategies so our payoffs are actually expected payoffs, but we will keep calling them payoffs.

The payoff for a player is going to be

$$Payoff = \sum_j vP[x_j^i \geq x_j^{-i}] - \sum_j x_j^i, \tag{5.4}$$

where of course the second part is the 'payment' and the first part is what a player expects to gain from the fields, which depends on the probability (P) of the player i to invest more than the player $-i$ on the

field j. It is quite obvious that (considering the fact that there are no synergies among objects [Kva07] and considering that in particular we are considering continuous distributions) we will have that:

$$Payoff = \sum_j [vG_{-ij}(x_j^i) - x_j^i]. \tag{5.5}$$

Let's try to explain this better. We are saying that the probability of a player of investing more than the other player $P[x_j^i \geq x_j^{-i}]$ is equal to the probability for the other player to invest $0 \leq x \leq x_j^i$ on the field j, which is $P[0 \leq x \leq x_j^i] = G_{-ij}(x_j^i)$.
This means that the payoff of a player depends, as stated by Kvasov in the paper, on the marginal distribution of the other player mixed strategy.

Following [Kva07], now we can now enunciate the following proposition:

Proposition 1. *Let's take $N > 2$, if we define $m = min\{v, \frac{2B}{N}\}$ then:*
Necessary and sufficient condition for a joint distribution $G_i : Z^i \to [0, 1]$ to constitute an equilibrium strategy is that all its 1-dim marginal distributions are uniform over $[0, m]$. In this case we have also that:

1. *All the equilibria have as a result the same payoff,*
2. *The payoff is $\frac{N}{2}(v - m)$.*

Also as stated in the paper the uniformity conditions of the marginal distributions is sufficient for a joint distribution to be an equilibrium in zero-sum games too.

It is important to notice a difference between this game and the classic Blotto Game, which is related to definition of m.
The introduction of m is necessary due to the existence of budget constrains, in fact it does not allow to a player to invest on a field more than the value of the field itself v. In a context where we do not care about our investment, as the game we studied in the previous chapters, for a player there are not such limitations, but it is clever to not invest more than $\frac{2B}{N}$.

5.2.3 Again about Mixed Strategies

The analysis we did in the previous subsection does not give us a complete study of the equilibria in terms of joint distributions. As stated in the paper this is due to the presence of budgets limitation, that leads to the construction (as seen before) of uniform marginal distributions over $[0, m]$. This makes impossible to construct the multivariate joint distribution from the margin. We will not investigate further on the reasons.

It is now important to say that given an equilibrium in a zero-sum game, with connected supports of its 1-dim marginal distributions, this is also an equilibrium for an appropriately scaled non-zero sum game (Lemma 1, [Kva07]).

So if we identify the equilibria for a non-zero game we have solved our problem if we are dealing with small budgets, such that $m = \frac{2B}{N}$ (So for $B \leq \frac{Nv}{2}$). If budgets are large (so $\frac{Nv}{2} < B \leq Nv$) new equilibria, in addition to the ones inherited from the zero-sum game, can occur.

5.3 The case $N = 3$

In this chapter we will take a look at the case with 3 battlefields. As seen before there are not pure strategies Nash Equilibria. We know that in case of joint distributions with uniform 1-dim margins over $[0, m]$ we have mixed strategies equilibria.

For what seen before the joint distributions can be found in the case of a zero-sum game and scaled, but

there are going to be more equilibria for large budgets. In fact in that case the game is more flexible and players are more free in their choices. This translates in the possibility of constructing various possible equilibria for certain budgets.

We will focus on the zero-sum game strategies that are going to be reusable also in non-zero sum games. (We also remember that for small budgets the equilibria coincide).
Gross and Wagner, [OG50], proposed a geometric interpretation of the research of equilibria for this 3 fields games, which we report here:
We take a triangle, whose sides represents the value of the fields, we construct a circle inside the triangle and an hemisphere above the circle. Then we take a point of the hemisphere, chosen from a density unif. distributed on the hemisphere. Now we project the point on the circle and calculate the areas of the resulting 3 triangles we can obtain connecting the projection with the sides of the first triangle. Now we divide the investment over the fields proportionally to the areas we calculated.
In the paper by Gross and Wagner are given many arguments and reasoning, we are not going to investigate to prove that we can get an equilibrium in this way. Also Borel has approached this problem in [E.B38]. At the end of the day, the two solutions have both uniform marginal distributions over connected supports (so they can be inherited in the non-zero sum game) and are both based on random choices over the following set:

$$H(B) = \{x^i : 0 \leq x^i_j \leq \frac{2B}{3}, x^i_1 + x^i_2 + x^i_3 = B\}. \tag{5.6}$$

Note that for each j we are choosing in $[0, m]$, with $m = \frac{2B}{3}$.
With this we conclude our study of the variation of the Blotto Game proposed by Kvasov.

Chapter 6

Regret Minimization Learning

In this chapter we will discuss the idea of using a regret learning approach (that we can call also no-regret since we are trying to act in a way to reduce regrets). Then we will also provide the regret learning algorithm and discuss it. Finally we will show that it converges to a set which is the correlated equilibrium set. Before proceeding we will give some definitions. (We will take them from the notes of the module and from the article [TWN13])

6.1 Some definitions

The notion of Nash Equilibrium can be generalized obtaining the the Correlated Equilibrium. So we will have $NE \subset CE$. A Correlated Equilibrium could be explained by introducing an intermediator, who gives instruction on the choice of an action to both the players. The choice will be random and can be seen as a matrix (p_{ij}) (which is a probability distribution with all non-zero entries and sum over the elements equal to 1). So trying to generalise as much as we can and following the notes' notation:

Definition 5. *If we have two players A and B and suppose that they have respectively m and n possible actions we will say that a joint distribution (p_{ij}) is a correlated equilibrium for the bimatrix game (A, B) if*

$$\sum_k a_{i'k} p_{ik} \leq \sum_k a_{ik} p_{ik} \tag{6.1}$$

and

$$\sum_l b_{lj'} p_{ij} \leq \sum_l b_{lj} p_{lj} \tag{6.2}$$

for all i, i', j, j'

So in other words the probability distribution is a Correlated Equilibrium if no player has an incentive to deviate from the the instruction extrapolated by the intermediator intervention.

Now we introduce the concept of regret.
When a player chooses an action which gives as a result an utility u, he can regret not having chosen one of the other actions, which could have given to him a better payoff. So if we call the payoff of a certain action i $u(i)$, we have:

Definition 6. *When a player chooses the action i the regret associated to an action j is $u(j) - u(i)$.*

Let's notice that obviously if j is a better action for the player then the regret will be positive, if the action j has the same payoff as i or if $i = j$ obviously the player will not regret not having chosen j. Finally if j is a worse option the regret will be negative. However we will not care about this extreme situations in our algorithm and we will consider only positive regrets. (which are the only ones which make sense for our problem).

6.2 The regret learning algorithm

The idea of this algorithm is to the regrets to choose the following action. In particular we do not want to be predictable, so we cannot choose directly the action we regretted not taking the most. Instead we will base our algorithm to the so called regret matching. In poor words we choose the next action in probability proportionate to the regrets accumulated. In particular the cumulative regrets are used to build a probability vector, which is used to determine the next strategy.

So we initialize a vector, with the each entry associated to the cumulative regret of an action, putting all the entries equal to 0. We also initialize a vector to store the strategy profile sum.
Then we start a loop that repeats for a decided number of times (or until it reaches a result we want) that:

1. Applies, when possible (if at least a regret entry is positive) a strategy based on regret-matching (this is done based on a probability vector obtained from the cumulative regrets vector). Otherwise it applies a strategy based on an uniform random choice between the actions (this is what happens during the first iteration).

2. Adds the strategy profile to our vector of strategy profile sum and builds the actions based on the strategy profile. It computes the regrets and adds them to the vector of cumulative regrets

Finally it gives back an average strategy profile (dividing by the number of the iterations) which is going to be used for the next iteration.

6.3 Convergence

Following the course notes and in particular chapter 7, and also the paper [TWN13] (and also in particular in [HMC00]) we can say over time if both players use this algorithm then for $t \to \infty$ then it will converge almost surely to the correlated equilibrium set of the game.

6.4 Implementation

In the appendix there is the code implementation for the algorithm of regret learning in the case of the Blotto Game with $(S, N) = (5, 3)$, as asked in the exercise 2.6 of the paper [TWN13]. In particular it computes all the possible actions, which in our case with are going to be:

[0 5 0]	[0 0 5]	[5 0 0]
[0 1 4]	[1 0 4]	[0 4 1]
[4 0 1]	[1 4 0]	[4 1 0]
[3 2 0]	[0 2 3]	[3 0 2]
[2 0 3]	[0 3 2]	[2 3 0]
[1 1 3]	[1 3 1]	[3 1 1]
[1 2 2]	[2 1 2]	[2 2 1]

(6.3)

So in our case we have 21 different actions (each action is represented by a 3-vector like $[0, 5, 0]$).

Then the code implements the algorithm and can be suited to different situations. For example using the function `getmixedtraining` we could use the regret matching for a player given a fixed opponent strategy. We can also focus on the evolution of the regrets over time.

6.4.1 Convergence

I have mainly focused on the case when the code runs the algorithm for both the players. In particular the couple of vectors I am going to converge (calculated after 10000 steps) is

([0.01481352 0.01473103 0.0147605 0.04254354 0.04264115 0.04282689 0.04273496 0.04264191 0.04253664 0.06028629 0.06027119 0.06034453 0.06040605 0.0604184 0.06030343 0.05459298 0.05474647 0.0547431 0.05788473 0.05780025 0.05797246] [0.01473729 0.01472699 0.01474625 0.04250549 0.04263618 0.04265758 0.04274585 0.04265996 0.04266419 0.06040281 0.06012874 0.06037721 0.06036511 0.06026568 0.06042968 0.05459903 0.05475823 0.05479635 0.05786353 0.05794615 0.05798771])

which as stated in [TWN13] is a Nash Equilibrium.

I have also coded a function that generates a plot that shows the 'convergence'.
This is shown taking the couple of strategy vectors obtained by the code after 10000 iterations and then using a loop that computes the strategy vectors for different number of iterations (until 10000). Then we sum for each step of the loop the norms of the the difference between the strategy vectors obtained with 10000 iterations and the vectors obtained with the number of iterations of the selected step.
This shows how the distance to the strategy with 10000 iterations evolves while the number of iterations increase. Finally I plot putting the x-axis the number of the iterations and on the y-axis the distance. I obtain the resulting plot in the figure 6.1, which clearly shows a convergence

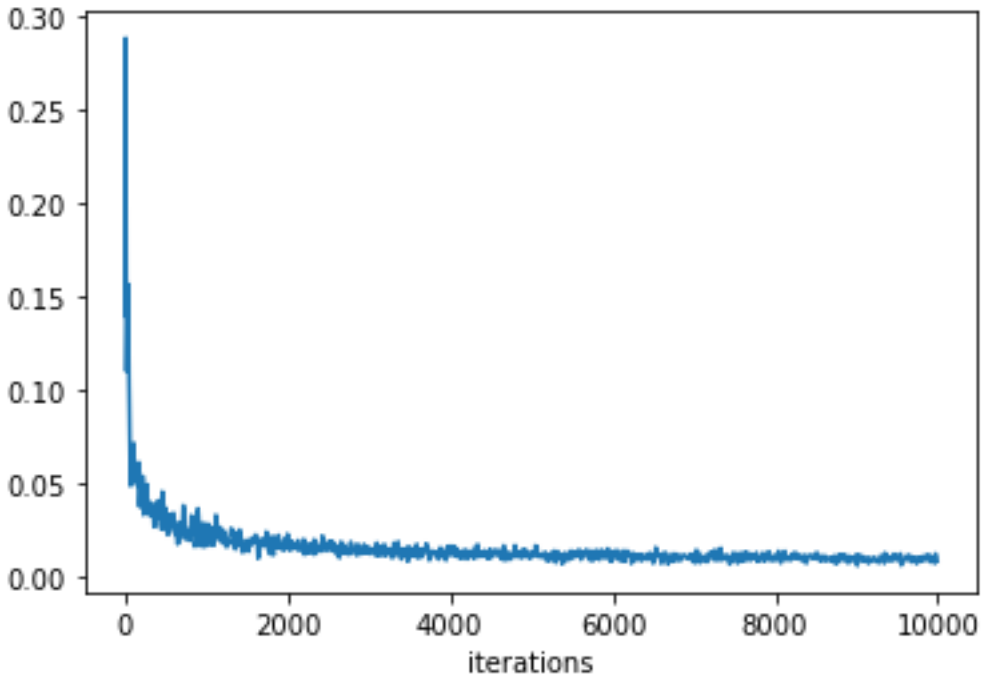

Figure 6.1: Convergence (section 6.4.1)

6.4.2 Speed of convergence

In this section we show the results of the plotting function `speedconvergence()`. In particular in the following plot we can see how even 1000 iterations are already enough for a good approximation. Finally it is also interesting to note that even if computing the distance from the strategy with 10000 iterations can lead to different scalings of errors, the plot still has a similar structure.

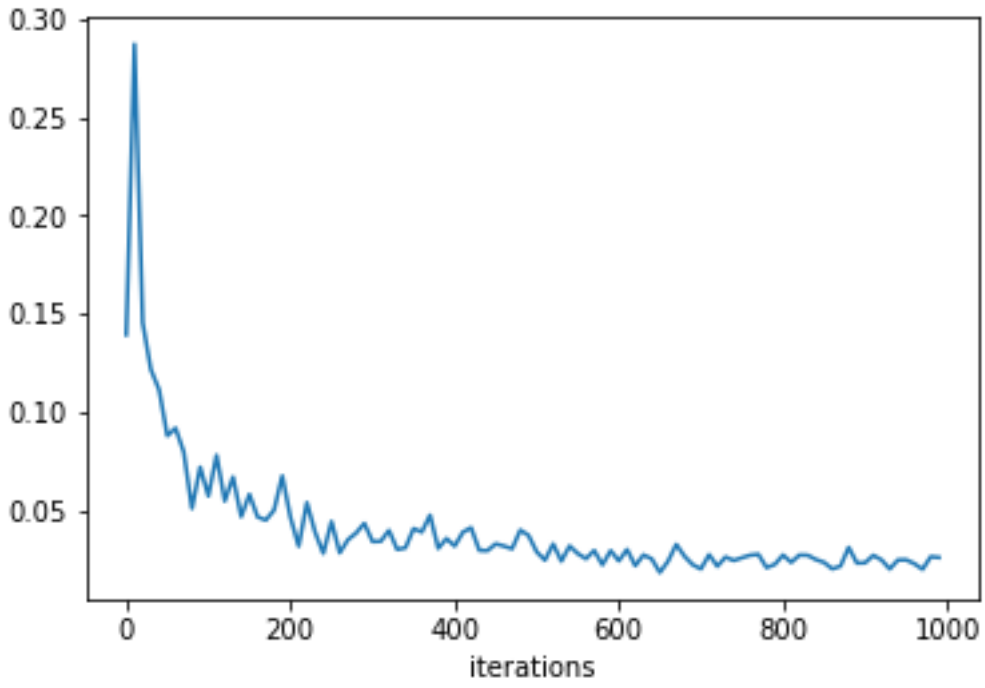

Figure 6.2: convergence (section 6.4.2)

Chapter 7

Mastery Component - Other modifications of the Blotto Game and Meta Games

This chapter constitutes the Mastery Component of this project, even if some mastery parts are already present in the previous chapters. It deals with some slight modifications of the Blotto game to check the resilience of the regret algorithm and finally the main part is about an empirical analysis of the Blotto Game.

7.1 External unexpected events and regret learning: a mastery analysis

Is evident from this project that Blotto game, even if at first sight simple is one of the most important topics of research in the field of Game theory nowadays. This is due to its flexibility, in fact slight modifications can lead to different models, which can have applications in economy, finance, elections, sociology and others. We saw that a lot of differences may occur if we pass from a discrete to a continuous version of the game and even if we introduce limitations over the budgets.
However we are still far from creating a real-world model, and this mainly for a reason: uncertainty. We are not considering the possibility of random external events that can change the result of our game. This however happens every day in all the subjects we cited before, form economics to sociology and so on.

Let's stick with the classic example of the Blotto Game and let's keep the rules we gave in the first chapters, we can notice that if we see the game as a game of soldiers and fields we can for example have a casual loss of a soldier due to illness or other reason.

7.1.1 Resilience of Regret Learning

The aim of this section is to suppose this possible and to see what happens in the context of the regret learning algorithm implementation and in particular to test its resilience in real-world situations. Can it resist to external unexpected events? We will focus on some examples.

In particular we will base our analysis on the same type of plots we created in the previous chapter. So we introduce a new function that we will call `stocpayoff`, which is based on the function `payoff` and contains a new line, which adds to the action vector of player A, for each component a random element between -K and K. So if we add a positive number $n < K$ to the component j of the vector of the action of A, we are saying for example that the other player has lost n soldiers and so our player will have an

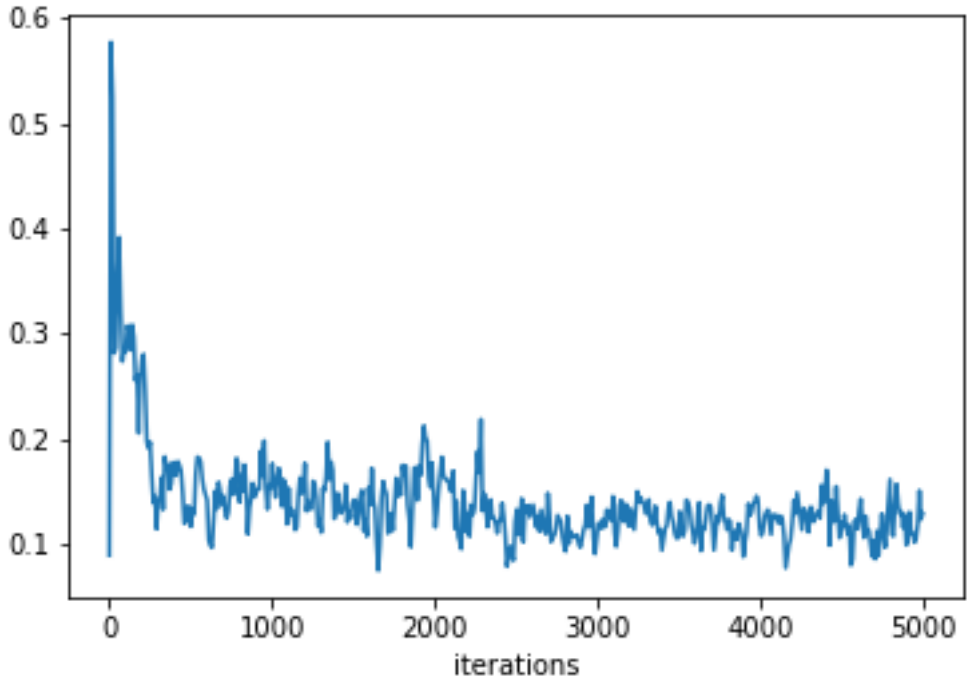

Figure 7.1: $K = 2$

advantage. On the other hand if we add a negative number we are giving an advantage (due to soldier loss, environmental reasons) to the other player.

It is important to note that we will stick with our payoff giving (1 to the player that wins more local battle, -1 to the losing player and 0 in case of a draw) and we are not adding payoffs for each field. In this situation, differently from some situations above, this can actually make a difference, but we won't focus on this aspect.

Taking again $(S, N) = (5, 3)$, we will plot as in the previous chapter the distance between the strategy with 10000 iterations, to see how in distance from this strategy the strategies of our algorithm evolve increasing the number of iterations (we will arrive to 5000/6000). In particular we will see three cases in which we suppose our unexpected events to have a weight of $K = 2, 3, 4$ (We mean that we can add randomly to each field for a player from $-K$ to K). Note that for $K = 4$, we are almost at the limit of max number of soldiers for field, so we are saying that random unexpected external events (if equal to + or -4) can almost revert every field's local battle.

The plots are 7.1,7.2,7.3 and we can notice that even if they seem flatten around a strategy vector, still the distance fluctuates much more than in the previous examples. Moreover the evolution seems to get worse for grater K.

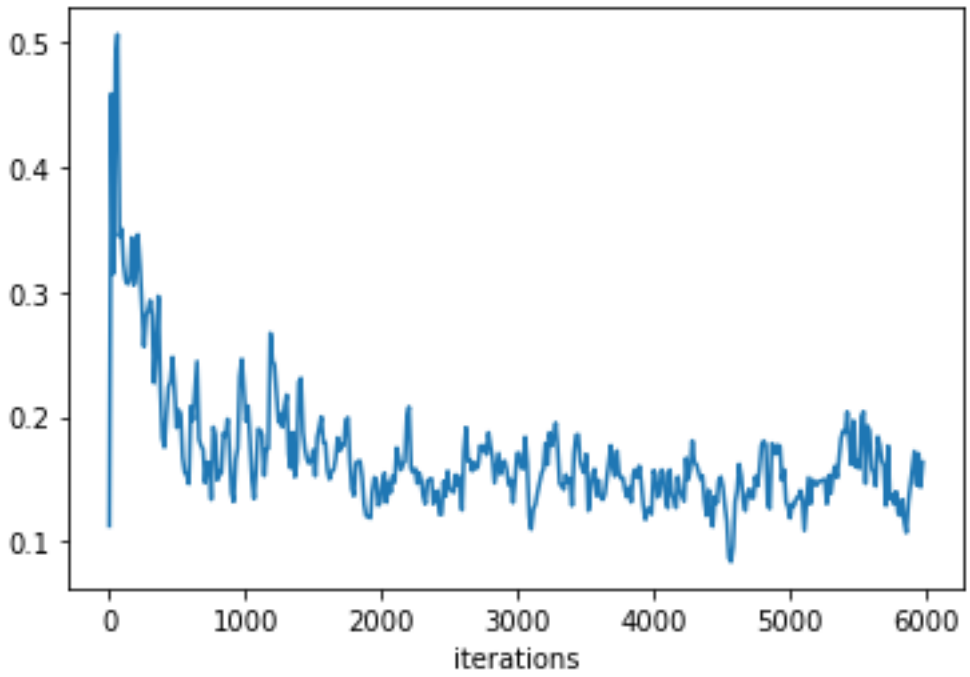

Figure 7.2: $K = 3$

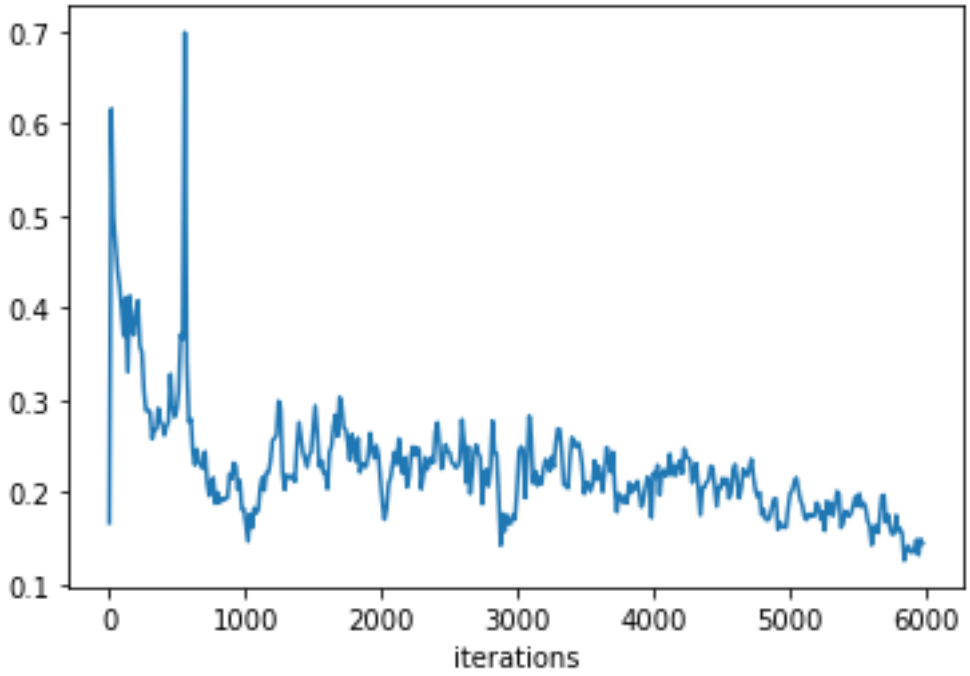

Figure 7.3: $K = 4$

Appendix A

First Appendix

```
import numpy as np
import scipy as scipy
import scipy.special
from itertools import combinations
from itertools import permutations
from numpy import random
import matplotlib.pyplot as plt

def flatten(t):
    return [item for sublist in t for item in sublist]

def partitionfunc(S,N,l=0):
    '''Creating actions can be seen as a partitioning problem
    S is the integer to partition, N is the length of partitions, l is the min partition
        element size'''
    if N < 1:
        return
    if  N== 1:
        if S>= l:
            yield (S,)
        return
    for i in range(l,S+1):
        for result in partitionfunc(S-i,N-1,i):
            yield (i,)+result

def actions(S,N):
    '''This function provides a matrix A, which contains for each row one of the possible
        actions
    This function is based on the partition function'''
    #Firstly we compute the number of possible actions
    numb_actions = int(scipy.special.binom(S+N-1,N-1))
    #We create a matrix with all the possible partitions
    B = np.array(list(partitionfunc(S, N)))
    #note that B does not make difference between actions like (5,0,0) and (0,5,0), so we need
        to settle this
    l = []
    m = len(B)
    for i in range(m):
       l.append(list(set(list(permutations(B[i,:])))))

    final_list = flatten(l)
```

```
    #now we can create A
    A = np.array(final_list)
    return A

def payoff(actionA,actionB):
    '''Given an actionA of the player A and an action B of the player B
    computes the payoffs'''
    #size of action vector
    N = actionA.shape[0]
    #we count the victories and the losses, so we initialize two variables
    victories = 0
    losses = 0
    for i in range(N):
        if actionA[i]>actionB[i]:
            victories += 1
        if actionA[i]<actionB[i] :
            losses +=1
        #we add nothing for draws
    if victories>losses:
        return 1
    if losses>victories:
        return -1
    else:
        return 0

def regrets(A, actionA,actionB):
    '''Given an actionA of the player A and an action B of the player B
    it computes the regrets confronting with the matrix A storing all the actions'''
    #Firstly we compute the number of possible actions directly from the matrix
    numb_actions = A.shape[0]
    #we initialize a vector in which we will store the regrets for each action
    regret_vector = np.zeros(numb_actions)
    #now we will check the payoff/utility for our action and we will confront it with all the
        other actions
    pay = payoff(actionA, actionB)
    #with a loop we will confront all the actions, we will assign the regret as the difference
        between the payoffs
    for i in range(numb_actions):
        regret_vector[i] = payoff(A[i], actionB)-pay
    return regret_vector

def getmixedtraining(A,iterations, strategyB):
    ''' Given the opponent strategy it does a sortt of 'training', returning at the end the
        average strategy profile'''
    #Firstly we compute the number of possible actions directly from the matrix
    numb_actions = A.shape[0]
    ind = [k for k in range(numb_actions)]
    #we introduce a new vector containing the regrets sum, as explained in the Lanctot paper's
        algorithm (now we initialize it to 0)
    regret_sum = np.zeros(numb_actions)
    #as in the algorithm we also introduce a strategy sum algorithm (we initialize it as a
        vector of zeros)
    strategy_sum = np.zeros(numb_actions)
    #we remember that as explained in the algorithm we can't use a strategy based regret
        matching sometimes, so we need to introduce a uniform strategy
    u = np.zeros(numb_actions)
    for i in range(numb_actions):
        u[i] = 1/numb_actions
```

```
    for j in range(iterations):
        if regret_sum.sum()>0:
            strategy = regret_sum/regret_sum.sum()
        elif regret_sum.sum() == 0:
            strategy = u
        #now following the algorithm I sum the strategy I obtained to the strategy sum vector
        strategy_sum += strategy
        #The opponent plays using his strategy
        indicatorB = random.choice(ind,p=strategyB)
        chosenBaction = A[indicatorB,:]
        #now I choose randomly an action using the strategy (which is a mixed strategy and so a
            probability vector)
        indicatorA = random.choice(ind, p=strategy)
        chosenAaction = A[indicatorA,:]
        #regrets
        r = regrets(A,chosenAaction, chosenBaction)
        #only the positive regrets will be added to the sum vector
        for t in range(numb_actions):
           if r[t]>0:
             regret_sum[t] += r[t]
    #Now I will normalize the sum vector for the strategy and return medium strategy
    if strategy_sum.sum()>0:
        medstrat = strategy_sum/strategy_sum.sum()
    elif strategy_sum.sum() == 0:
        medstrat = u
    return medstrat

def Blotto(A,iterations):
    '''this puts together what we have done until now computing the regret learning algo for
        both players in the blotto game context'''
    #Firstly we compute the number of possible actions directly from the matrix
    numb_actions = A.shape[0]
    ind = [k for k in range(numb_actions)]
    #Now we will need vectors forstoring the regret sums for both the players
    Aregret_sum = np.zeros(numb_actions)
    Bregret_sum = np.zeros(numb_actions)
    #same for the strategy sum vectors
    Astrategy_sum = np.zeros(numb_actions)
    Bstrategy_sum = np.zeros(numb_actions)
    #as before the uniform strategy
    u = np.zeros(numb_actions)
    for i in range(numb_actions):
        u[i] = 1/numb_actions

    for j in range(iterations):
        if Aregret_sum.sum()>0:
            Astrategy = Aregret_sum/Aregret_sum.sum()
        elif Aregret_sum.sum()==0:
            Astrategy = u
        if Bregret_sum.sum()>0:
            Bstrategy = Bregret_sum/Bregret_sum.sum()
        elif Bregret_sum.sum()==0:
            Bstrategy = u
        #now following the algorithm I sum the strategies I obtained to the strategy sum vectors
        Astrategy_sum += Astrategy
        Bstrategy_sum += Bstrategy
        #action choice
```

```
        Aindicator = random.choice(ind, p=Astrategy)
        chosenAaction = A[Aindicator,:]
        Bindicator = random.choice(ind, p=Bstrategy)
        chosenBaction = A[Bindicator,:]
        #regrets
        Ar = regrets(A, chosenAaction, chosenBaction)
        Br = regrets(A, chosenBaction, chosenAaction)
        #only the positive regrets will be added to the sum vectors
        for t in range(numb_actions):
           if Ar[t]>0:
              Aregret_sum[t] += Ar[t]
           if Br[t]>0:
              Bregret_sum[t] += Br[t]
    #now we conclude
    if Astrategy_sum.sum()>0:
        Amedstrat = Astrategy_sum/Astrategy_sum.sum()
    elif Astrategy_sum.sum() == 0:
        Amedstrat = u
    if Bstrategy_sum.sum()>0:
        Bmedstrat = Bstrategy_sum/Bstrategy_sum.sum()
    elif Bstrategy_sum.sum() == 0:
        Bmedstrat = u
    return (Amedstrat, Bmedstrat)


def convergence():
    '''this function deals with the convergence and plots a plot showing how the convergence
        evolves through the iterations'''
    #I build the action matrix for our case (S,N) = (5,3)
    A = actions(5,3)
    #I want to show that the algorithm comverges, I calculate two strategies after 10000 steps
        and I will confront everything with them
    a,b = Blotto(A,10000)
    print(a,b)
    l = []
    #I compute a confrontation between a,b and different steps with different numbers of
        iterations
    for i in range (1000):
        c,d = Blotto(A,10*i)
        l.append(np.linalg.norm(a-c)+np.linalg.norm(b-d))
    #now we plot
    plt.plot([10*i for i in range(1000)],l)
    plt.xlabel('iterations')
    plt.ylabel('closeness to the case with 100000 iterations')
    plt.show()



def speedconvergence():
    '''This function is a modified version of the previous one which aims to show that less
        than 10000 steps are enough'''
    #I build the action matrix for our case (S,N) = (5,3)
    A = actions(5,3)
    #I want to show that the algorithm comverges, I calculate two strategies after 10000 steps
        and I will confront everything with them
    a,b = Blotto(A,10000)
    print(a,b)
    l = []
```

```
    #I compute a confrontation between a,b and different steps with different numbers of
        iterations
    for i in range (100):
        c,d = Blotto(A,10*i)
        l.append(np.linalg.norm(a-c)+np.linalg.norm(b-d)) #different type of errors could be
            taken in account
    #now we plot
    plt.plot([10*i for i in range(100)],l)
    plt.xlabel('iterations')
    plt.ylabel()
    plt.show()



#run one of the followings:
convergence()
speedconvergence()

#the external factor based payoff function. To use it we need to substitute K with a number of
    our choice, which will give the max amount of external randomness per field. Then we can
    use it in the previous code instead of the function payoff, with opportune substitutions
stocpayoff(actionA,actionB):
    '''Given an actionA of the player A and an action B of the player B
    computes the payoffs considering eventual external factors, which can be of the order of
        E'''
    #size of action vector
    N = actionA.shape[0]
    #we count the victories and the losses, so we initialize two variables
    victories = 0
    losses = 0
    actionA += np.array(random.randint(-K,K,size=N))
    for i in range(N):
        if actionA[i]>actionB[i]:
            victories += 1
        if actionA[i]<actionB[i] :
            losses +=1
        #we add nothing for draws
    if victories>losses:
        return 1
    if losses>victories:
        return -1
    else:
        return 0
```

Bibliography

[E.B38] J.Ville E.Borel. Application de la théorie des probabilitiés aux jeux de hasard. *Gauthier-Villars,Paris*, 1938.

[HMC00] SERGIU HART and ANDREU MAS-COLELL. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[Kva07] Dmitriy Kvasov. Contests with limited resources. *Journal of Economic Theory*, 136:738–748, 2007.

[Nas51] John Nash. Non-cooperative games. *The Annals of Mathematics, Second Series*, 54(2):286–295, 1951.

[OG50] R. Wagner O. Gross. A continuous colonel blotto game. *RM-408, RAND Corp., Santa Monica*, 1950.

[TWN13] Marc Lanctot Todd W. Neller. An introduction to counterfactual regret minimization. 2013.