Politecnico di Torino

Master's degree in Mathematical Engineering

Markov Chain Model for Football Analytics

Work of:
Formento Emanuele s282411

Collaboration of:
Barbiero Federico (Deltatre)
Supervision of:
Prof. Bibbona Enrico

Academic Year 2021-2022

Abstract

This work arises from a thesis proposal launched by the company Deltatre, whose objective is to extract key performance indicators of a team or a player through a Markov chain model from the data collection carried out on a football competition. Each match is modeled as a Markov chain with suitable states and transitions. The Markov chain theory is used to create the model, while other mathematical tools as chi-square distribution and confidence intervals are used to check the goodness of results. The creation of statistics is inspired by the expected threat theory introduced by Karun Singh.

The data preprocessing begins with understanding the data available and those useful for analysis: the most important are the ball position and the player and the team in ball possession, but also other information is used, as the type of event and the phase of the match.
Then the states defining the model are choosen, composed by field areas plus two additional states, the goal and the lost ball; the number of field states depends on the subdivisions on both sides of the field, creating a $m \times n$ grid which can be represented using blurred and defined heatmaps.

Once defined the states, a transition matrix is needed, or rather his estimation, to complete the Markov chain model; this is done computing the frequencies matrix, whose entries include the count of how many times the ball does a transition from a state to another, and then normalizing it to obtain a probabilities matrix, having all sums on rows equal to 1.
The transition matrices can be computed taking data from a single match or from the entire tournament; in this last case it is used also to compute the stationary distribution of the Markov chain.
But these results come from estimates, so it is critical to explore their robustness through the Goodman method, obtaining heatmaps of absolute and relative confidence interval amplitudes.

In the last chapter the idea of Karun Singh is applied to the model, obtaining the expected threat related to the tournament edition.
Then it is used to find the dominance of teams during the match, which can be done in different ways and it gives a dynamic and clear idea of the behaviour of the match.
Some possibilities are to group or to weight different values of the expected threat with respect to established criteria, for example subdividing the match in minutes or actions and adding the expected threat in these parts.
The idea of dominance is also applied to players, considering their expected threat both during a match or during the tournament.
Then to make fairer players performances, two new improvements are introduced: a normalization for minutes played, so that those who play more minutes or matches are not more advantaged than others, and a contribution of players to the expected threat, computing their gains not just based on where they touch the ball, but also where they pass or steal it, thus encouraging even less offensive players.

Introduction

About company

Born in 1986 from an idea of Giampiero Rinaudo and Luca Marini, Deltatre is the world's leading sports and entertainment technology provider, offering graphics, data, OTT and live broadcast solutions.
It counts more than a thousand employees in offices spread across 19 cities around the world and it has received more than 200 awards in its history.
It works with mostly international clients having business in several sports, supporting them in all key steps of a process.
About a match for example, its contribution starts finding useful statistics and information to introduce the teams or the players before the start.
It continues during the event or its break, e.g. showing highlights or real-time analysis on the action just taken.
It finishes with the data collection and analysis after the match, providing statistics and insights on it.

Aim of the work

Given its extremely innovative nature, the company is constantly looking for new metrics or key performance indicators that may help a better game understanding.
The first step is to utilize mathematical tools, precisely a Markov chain and their properties, to create a model for football matches applicable to the large amount of data in the company database, containing matches of different tournaments and years.
In this way the Markov chain model is used to extract statistics about the match, which can be aggregated into tournament statistics, e.g. considering all matches of a team in the competition, or disaggregated into player statistics, that can be again grouped to obtain tournament statistics player by player for example.
The second step is to find the new metrics, which can be done taking inspiration from what is offered by the model, what is requested by clients or business, but also trying to imagine alternative ways to explain the match, stimulating the creativity.

Contents

Chapter 1

Theoretical prerequisites

1.1 Markov chain

"A Markov chain or Markov process is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event." [Eng17]
Its main characteristic is *"that satisfies the Markov property (sometimes characterized as "memorylessness")"* [Gag17] so in other words, conditional on the present state of the system, its future and past states are independent.

Since the system changes randomly, about the future it is generally impossible to predict with certainty the next steps of a Markov chain at a given point, unlike the asymptotic statistical properties of the system, which can be predicted.

Moreover, there are different types of Markov chains according to how the time is characterized (discrete or continuous) and the cardinality of the state space (countable or continuous).

1.1.1 Formal definition

A discrete-time Markov chain is a stochastic process with the Markov property, that is a sequence of random variables $X_1, X_2, X_3, \ldots$ which satisfies

$$\mathbb{P}(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \mathbb{P}(X_{n+1} = x \mid X_n = x_n) = P(x_n, x),$$

so the probability of moving to the next state depends only on the present state and not on the previous history.
It holds if conditional probabilities are well defined, that is

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) > 0.$$

P is the transition matrix of the process (previously evaluated in the entry (x_n, x), that is the transition from state x_n to state x), which together with a state space $\mathcal{S}$ and an initial state i_0 or a distribution across the state space π_0 uniquely identifies a Markov chain.[Wike]

1.1.2 States

The state space of the chain $\mathcal{S}$ is a finite set including the possible values of X_i.

Given two states i and j, j is reachable from i ($i \to j$) if $\exists m \geq 0$ s.t. $p^{(m)}(i,j) > 0$, while they are called *communicating* if $i \to j$ and $j \to i$.
If all states of the space are reachable with each other, the Markov chain is irreducible.
Given the finite cardinality of the state space, if the Markov chain is irreducible and aperiodic, Perron–Frobenius theorem states that there is a unique stationary distribution π and *"P^k converges to a rank-one matrix in which each row is the stationary distribution π, that is*

$$\lim_{k\to\infty} P^k = \mathbf{1}\pi,$$

where $\mathbf{1}$ *is the column vector with all entries equal to 1".*[Wike]
Finally if $\lim_{k\to\infty} P^k$ is found, then the stationary distribution of the Markov chain can be easily determined for any starting distribution.

If a state i is aperiodic and positive recurrent, it is defined as ergodic; *"in other words, if it is recurrent, it has a period of 1 and finite mean recurrence time.*
If all states in an irreducible Markov chain are ergodic, then the chain is said to be ergodic." [Wike] If a finite and irreducible Markov chain has an aperiodic state, it is ergodic.
In general, *"a Markov chain is ergodic if there is a number N such that any state can be reached from any other state in any number of steps less or equal to N.*
In the case of a fully connected transition matrix, where all transitions have a non-zero probability, this condition is fulfilled with $N = 1$.
A Markov chain with more than one state and just one out-going transition per state is either not irreducible or not aperiodic, hence cannot be ergodic". [Wike]

1.1.3 Transitions

The changes of state of the system are called transitions and the probabilities associated with them are called transition probabilities.
If the state space is finite, they can be included in the transition matrix, such that $p_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i)$.
"Since each row of P sums to one and all elements are non-negative, P is a right stochastic matrix". [Wike]

Usually in real cases the transition matrix is not given because it is really difficult to know exact transition probabilities from a state to another.
Then it is necessary to find their estimates, which is a relatively straightforward process if sequence of states for each individual transition can be observed.

In general, denoting n_{ij} the number of observations in state i at $t-1$ and in state j at t, the probability of going from i to j is:

$$p_{ij} = \frac{n_{ij}}{N_i},$$

that is nothing but the proportion of observations started in state i and ended in state j on all those started in state i.[Jon05]

1.1.4 Assumptions and other properties

The Markov chain considered is assumed to have two important properties:

- time-homogeneous: $\mathbb{P}(X_{n+1} = x \mid X_n = y) = \mathbb{P}(X_n = x \mid X_{n-1} = y) = p_{yx} \quad \forall n$, so the probability of the transition is independent of n.
- stationary: $\mathbb{P}(X_0 = x_0, X_1 = x_1, \ldots, X_k = x_k) = \mathbb{P}(X_n = x_0, X_{n+1} = x_1, \ldots, X_{n+k} = x_k) \quad \forall n, k$

"If the Markov chain is time-homogeneous, then the transition matrix P is the same after each step, so the m-step transition probability can be computed as the m-th power of the transition matrix, P^m ". [Wike]
Precisely in this case, taking an instant n then

$$p^{(m)}(i,j) = \mathbb{P}(X_{n+m} = j \mid X_n = i) = \mathbb{P}(X_m = j \mid X_0 = i) =$$

$$\sum_k \mathbb{P}(X_m = j \mid X_1 = k, X_0 = i) \cdot \mathbb{P}(X_1 = k \mid X_0 = i) = \sum_k p^{(m-1)}(i,k) \cdot p(k,j)$$

The trajectory of a Markov chain is the set of realizations of the stochastic process; given the initial distribution it is possible to compute the probability of obtaining a certain trajectory $i_1, i_2, \ldots, i_n$, that is

$$\mathbb{P}(X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) =$$

$$p(i_{n-1}, i_n) \cdot p(i_{n-2}, i_{n-1}) \cdot \ \ldots \ \cdot p(i_0, i_1) \cdot \mathbb{P}(X_0 = i_0)$$

The hitting time is that starting in a given state or set of states until the chain arrives in the same or another given state or set of states.
"The distribution of such a time period has a phase type distribution.
The simplest such distribution is that of a single exponentially distributed transition". [Wike]
"For a subset of states $A \subseteq \mathcal{S}$, the vector k^A of hitting times (element k_i^A represents the expected value that starting in state i the chain enters one of the states in the set A) is the minimal non-negative solution to [Nor97]

$$k_i^A = 0, \ i \in A, \quad -\sum_{j \in \mathcal{S}} q_{ij} k_j^A = 1, \ i \notin A".$$

Finally, another assumption is that the embedded Markov chain is considered, because the process is measured in microseconds, so it is almost continuous-time, but for the analysis the interest is given by the jump process from an event to another.

1.1.5 Embedded Markov chain

"One method of finding the stationary probability distribution π of an ergodic continuous-time Markov chain Q is by first finding its embedded Markov chain (EMC)".[Wike]
Let's denote with Q also the transition matrix of the Markov chain having entries q_{ij}.
The EMC, sometimes called jump process, is a regular discrete-time Markov chain.

The one-step transition probability matrix of the EMC, T, has entries t_{ij} representing the conditional probabilities of transitioning from state i into state j.
"One way to find these conditional probabilities is computing

$$t_{ij} = \begin{cases} \frac{q_{ij}}{\sum_{k \neq i} q_{ik}} & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

Then T may be written as

$$T = I - (\text{diag}(Q))^{-1} Q,$$

where I is the identity matrix, and $diag(Q)$ is the diagonal matrix formed by selecting the main diagonal from the matrix Q and setting all other elements to zero". [Wike]
To compute the stationary probability distribution vector, it should be found φ such that

$$\varphi T = \varphi, \qquad \varphi > 0 \quad \forall i \in \mathcal{S}, \quad \|\varphi\|_1 = 1,$$

and then

$$\pi = \frac{-\varphi(\text{diag}(Q))^{-1}}{\|\varphi(\text{diag}(Q))^{-1}\|_1}.$$

1.2 Probabilities robustness

In section 3.3 the problem of the quality of the probabilities found is addressed because they come from an empiric model, precisely from the collection of frequencies of the transition from one state to another and their normalization on rows.
A quite simple but efficient strategy to quantify this problem is the so-called *'Goodman'* method [Goo65].
As resumed in the source code [Per13], it *"is based on approximating a statistic based on the multinomial as a chi-squared random variable.*
The usual recommendation is that this is valid if all the values in 'counts' are greater than or equal to 5.
There is no condition on the number of categories for this method."
So there is a sort of weak condition on the frequency of a transition but not on the minimum number of transitions to be considered (condition also present in another method explained in [Per13], the *"sison-glaz"* one).

In the abstract of the article is underlined as this method provides better confidence intervals (better means shorter) than those suggested in the immediately preceding years, precisely by Quesenberry and Hurst in the 1964 and by Gold in 1963.

But before going into details of this method, it is necessary to introduce a couple of theoretical concepts.

1.2.1 Chi-square distribution

"The chi-squared distribution (also chi-square or χ^2-distribution) with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables". [Wikb]

The formal definition is that if $Z_1, \ldots, Z_k$ are independent, standard normal random variables, then the sum of their squares is distributed according to the chi-square distribution with k degrees of freedom, usually denoted as

$$Q = \sum_{i=1}^{k} Z_i^2 \sim \chi^2(k) \text{ or } \chi_k^2.$$

This distribution has a positive integer parameter k which specifies the number of degrees of freedom (that is the number of random variables Z_i being summed). [Moo74]

"The chi-square distribution is a special case of the gamma distribution and it is one of the most widely used probability distributions in inferential statistics, notably in hypothesis testing and in construction of confidence intervals". [Moo74]

"A chi-square test is a statistical hypothesis test that is valid to perform when the test statistic is chi-squared distributed under the null hypothesis, specifically Pearson's chi-squared test and variants thereof.
Pearson's chi-squared test is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table". [Wikc]

Suppose that n observations in a random sample from a population are classified into k mutually exclusive classes with respective observed numbers x_i, for $i = 1, 2, \ldots, k$ and a null hypothesis gives the probability π_i that an observation falls into the $i - th$ class.
So we have the expected numbers $m_i = n \cdot p_i \ \forall i$, where

$$\sum_{i=1}^{k} p_i = 1, \quad \sum_{i=1}^{k} m_i = n \cdot \sum_{i=1}^{k} p_i = n.$$

Pearson in [Pea00] proposed that assuming the correctness of the null hypothesis, as $n \to \infty$ the limiting distribution of the quantity below is χ^2-distributed

$$X^2 = \sum_{i=1}^{k} \frac{(x_i - m_i)^2}{m_i} = \sum_{i=1}^{k} \frac{x_i^2}{m_i} - n$$

1.2.2 Confidence intervals

"A confidence interval (CI) is a range of estimates for an unknown parameter", "computed at a designated confidence level (the 95% confidence level is most common, but other levels, such as 90% or 99%, are sometimes used)". [Dek05]
"The confidence level represents the long-run proportion of corresponding CIs that contain the true value of the parameter". [Wikd]

Formally *"let X be a random sample from a probability distribution with statistical parameter θ, which is a quantity to be estimated, and φ, representing quantities that are not of immediate interest.*
A confidence interval for the parameter θ with confidence level γ, is an interval $(u(X), v(X))$ determined by random variables $u(X)$ and $v(X)$ with the property:

$$\mathbb{P}\{u(X) < \theta < v(X)\} = \gamma \quad \forall(\theta, \varphi) \ .$$

The number γ, whose typical value is close to but never greater than 1, is sometimes given in the form $1-\alpha$ (or as a percentage $100 \cdot (1-\alpha)\%$), where α is a small positive number, often 0.05." [Dek05]

So taking the 95% confidence interval as an example, a confidence interval can be interpreted in two different ways:

- in terms of a **long-run frequency** in repeated samples: *"Were this procedure to be repeated on numerous samples, the proportion of calculated 95% confidence intervals that encompassed the true value of the population parameter would tend toward 95%."*[CD74]
- in terms of **probability** related to a theoretical sample: *"There is a 95% probability that the 95% confidence interval calculated from a given future sample will cover the true value of the population parameter."*[Ney37]

1.2.3 Quesenberry and Hurst method

Suppose that estimated probabilities $\hat{\pi}_{ij}$, $i, j \in \{1, 2, ..., k\}$ of transition from one state to another are computed as in paragraph 3.1.2, then in Quesenberry and Hurst's article [Que64] the confidence interval $[\pi_{ij}^-, \pi_{ij}^+]$ is proposed, where

$$\pi_{ij}^- = \frac{(A + 2n_{ij} - (A(A + 4n_{ij}(N_i - n_{ij})/N_i))^{1/2})}{2(N_i + A)},$$

$$\pi_{ij}^+ = \frac{(A + 2n_{ij} + (A(A + 4n_{ij}(N_i - n_{ij})/N_i))^{1/2})}{2(N_i + A)}.$$

The parameter A is the upper $\alpha \times 100$-th percentile point of the chi-square distribution with $k-1$ degrees of freedom, while N_i is the sample size of state i and n_{ij} is the observed cell frequency of the transition from state i to state j.
These confidence limits π_{ij}^-, π_{ij}^+ are derived simply as the two solutions of the following quadratic equation in π_{ij}

$$\left(\frac{n_{ij}}{N_i} - \pi_{ij}\right)^2 = \frac{A\pi_{ij}(1-\pi_{ij})}{N_i}, \qquad i, j = 1, 2, ..., k$$

In the article it is not explained the provenance of the formula, but it is clearly related to the use of a chi-square distribution for the binomial test; *"for large samples the binomial distribution is well approximated by convenient continuous distributions as the normal or precisely the chi-square and these are used as the basis for alternative tests that are much quicker to compute, such as Pearson's chi-square test"*. [Wika]
So given that a binomial distribution can be asymptotically approximated with a normal distribution, it holds that

$$Z = \frac{(n_{ij} - \pi_{ij}N_i)}{\sqrt{N_i\pi_{ij}(1-\pi_{ij})}}, \quad Z \sim \mathcal{N}(0,1)$$

Squaring both sides is obtained Pearson's cumulative test statistic χ^2, which asymptotically approaches a chi-square distribution:

$$\chi^2 = \frac{(n_{ij} - \pi_{ij}N_i)^2}{N_i\pi_{ij}(1-\pi_{ij})}$$

Then doing some manipulations the quadratic equation with χ^2 instead of A is obtained.

$$\chi^2 = \frac{(n_{ij} - \pi_{ij}N_i)^2}{N_i\pi_{ij}(1-\pi_{ij})} \implies \chi^2\pi_{ij}(1-\pi_{ij}) = \frac{N_i^2(\frac{n_{ij}}{N_i} - \pi_{ij})^2}{N_i}$$

$$\implies \frac{\chi^2\pi_{ij}(1-\pi_{ij})}{N_i} = \left(\frac{n_{ij}}{N_i} - \pi_{ij}\right)^2$$

Since solutions π_{ij}^-, π_{ij}^+ have to be related to the $1-\alpha$ confidence interval, χ^2 must be substituted by A as upper $\alpha \times 100$-th percentile point.

When $N_i \to \infty$ the fraction of cases included in the confidence interval will be at least $1-\alpha$.

But there are some difference with respect to different values of k: for $k = 2$, that fraction is $1-\alpha$ (and it coincides with that of usual large-sample confidence interval for the parameter of a binomial distribution), but for $k > 2$ the fraction will be greater than $1-\alpha$, so it is become just a lower bound, not the exact value.

1.2.4 Goodman method

The idea behind this improvement was to replace A with B, defined as the upper $(\alpha/k) \times 100$-th percentile of the chi-square distribution with one degree of freedom (while A was the $\alpha \times 100$-th of that with $k-1$ degrees of freedom).
By doing this the fraction of observations out of the confidence interval for a single π_i is $\frac{\alpha}{k}$ and therefore that for at least one of them is α (or less).

In addition to providing shorter confidence intervals than those given by Quesenberry and Hurst, the preceding remarks allow to obtain a more accurate bound for the usual fraction.
Let α' denote the probability that a chi-square variate with one degree of freedom exceeds A, then the fraction of observations out of the confidence interval for a single given parameter $\hat{\pi}_i$ is α' itself and that for at least one of the k confidence intervals is $k\alpha'$ (or less).
For $k > 2$, it was found that $k\alpha' < \alpha$ at the usual probability levels (e.g. $\alpha = 0.01, 0.05, 0.1$); this indicates that the upper bound of α, which is given by Quesenberry and Hurst for the fraction of observations out of the confidence interval for a single $\hat{\pi}_i$, should be replaced when $k > 2$ by the more accurate bound $k\alpha'$.

A more general set of simultaneous confidence intervals is obtained by replacing A by B_i in the confidence interval for $\hat{\pi}_i, i = 1, 2, ..., k$, where B_i is the upper $\beta_i \times 100$-th percentile of the chi-square distribution with one degree of freedom (while A was the $\alpha \times 100$-th of that with $k-1$ degrees of freedom) and $\sum_{i=1}^{k} \beta_i = \alpha$. Clearly in the special case where $\beta_i = \frac{\alpha}{k}$ and $B_i = B$, the simultaneous confidence intervals introduced earlier is obtained. [Goo65]

1.3 Expected threat

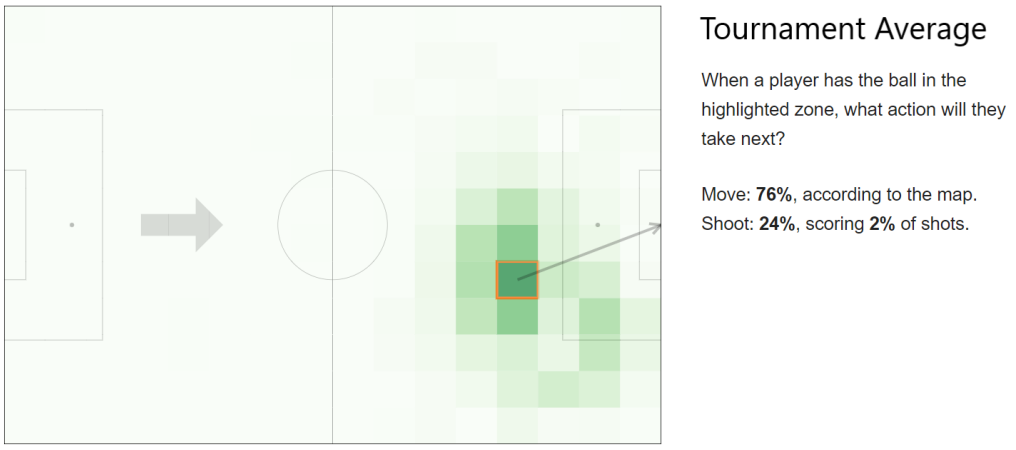
Born from a brilliant idea of Karun Singh, student of the Cornell University, the expected threat is a new interesting key performance indicator (KPI) for *"modelling team behaviour in possession to gain a deeper understanding of buildup play."* [Sin19].
The keypoints for the Karun framework are the following:

- reward individual player actions
- operate on event-level data
- reward actions independent of the end outcome of the possession
- reward moving the ball not just into positions with high goal probability, but also into "threatening" positions that can in turn lead to dangerous positions with high likelihood.

After these modelling assumptions were made, he would like to *"assign a threat value to every location on the pitch"*[Sin19]; it is not a new idea in football analytics, but the novelty is the way it is done.
Field areas are identified by a couple (x, y), where x is related to the long side of the field and y to the short one; so it is a sort of couple of discrete Carthesian coordinates having the origin of the system on the bottom left corner.
Karun selected for every zone (x, y) four attributes:

- **move probability** $m_{x,y}$: when a player has possession in zone (x, y), it is how often he opts to move (i.e. pass or dribble) the ball as next action.
- **shoot probability** $s_{x,y}$: when a player has possession in zone (x, y), it is how often he opts to shoot as next action.
- **move transition matrix** $T_{x,y}$: in the cases where the player moves from zone (x, y), it is the probability that he moves to each of the other zones (z, w).
  It is quite different from usual transition matrices, having starting states on rows and ending states on columns, because here the starting state (x, y) is fixed out of the matrix, which instead includes possible values of z on rows and those of w on columns.
  So the sum to 1 is not obtained by rows as usual but considering all entries of the matrix.
- **goal probability** $g_{x,y}$: when the player shoots from zone (x, y), it is the probability that the shot turns into a goal.

Then from a position (x, y), the first possibility is to shoot and score with probability $s_{x,y} \times g_{x,y}$, the second one is to pass the ball following the move transition matrix and consider the threat from the new position (z, w).



This reasoning is encoded in the iterative formula

$$xT_{x,y} = (s_{x,y} \times g_{x,y}) + (m_{x,y} \times \sum_{z=1}^{m} \sum_{w=1}^{n} \mathcal{T}_{(x,y)\to(z,w)} xT_{z,w}),$$

where m and n are dimensions of the long and short side of the field respectively, while $xT_{x,y}$ is precisely the expected threat of the zone (x, y), the new KPI including all possible source of threateness.

It is clear that computing the expected threat directly solving the iterative formula seems quite hard due to the presence of expected threat values in both sides of the equation; so Karun has taken advantage of the iterative formula itself to find a practical neat workaround.
The initial condition is

$$xT_{x,y} = 0 \quad \forall x = 1, \ldots, m, \quad y = 1, \ldots, n$$

to which it is applied the iterative formula; the result is

$$xT_{x,y} = s_{x,y} \times g_{x,y} \quad \forall x = 1, \ldots, m, \quad y = 1, \ldots, n$$

so at the first iteration the expected threat is nothing but the expected goal $xG_{x,y}$, another KPI largely used in football analytics.

Applying this formula iteratively, Karun *"found 4-5 iterations to be sufficient for reasonable convergence, though this may vary based on your dataset."*[Sin19] Of course the more steps are made the more combinations of possible actions grow; by the way another interpretation of the expected threat after n iterations is the probability of scoring within the next n actions, due to the iterative nature of the formula.

Results

Chapter 2

Preprocessing

2.1 Data understanding

Before starting with the manipulation of data, it is good practice to focus on their meaning and structure.

In this case the work will be based on a football competition, whose data are organized in the following way (see B.1 in appendix):

1. the outermost level includes just the year of the competition; the analysis will regard just one of the three years of data available.
2. for every year there is the list of all matches played, temporally ordered and marked by progressive numbers.
3. finally for every match there is a set of files including all information about it.

For the analysis will be mostly used the *Marks* file, introduced in the next paragraph, which deserves a more specific knowledge, while the content of the others can be resumed as follows:

- ***Lineups***: it includes two dictionaries with teams information, each having the *TeamID*, the status of *Home* or *Away* team and some information about players as *PlayerID* or *Role*, distinguishing between players on the field and those on the bench.
- ***MatchData***: it includes some technical information about the match as *GroupID* and *StadiumID*, in addition to the ID of match and teams playing it.

- ***MatchInfo***: it includes other background information about the match but not so related to the game, for example the geometry of the field as well as weather conditions (wind speed, temperature, humidity, etc...) .

- ***Phases***: it regards the two halves of the match, for each of which there is information as the effective duration of the half, that of the injury time and *LeftTeamID*.

2.1.1 Marks file

It contains the list of all events happened during the match, starting from the coin toss to the final whistle.
Every event has a dictionary structure containing about 20 keys and their values are again dictionaries, lists or even lists of dictionaries, so the structure can be very complex.
Let's go into detail with the most useful information for the analysis:

- ***Tags***: it is a list of strings (usually one) which briefly describes the event, using expression as *BallTouch*, *Goal*, *ThrowIn*, *Substitution*; for the analysis the first two type of events are the more relevant.

- ***OfficialTimeUTC***: it is a string containing the date and the hour of the event in format *'YYYY-MM-DDTHH:MM:SS.MLSZ'*, where *MLS* are milliseconds.

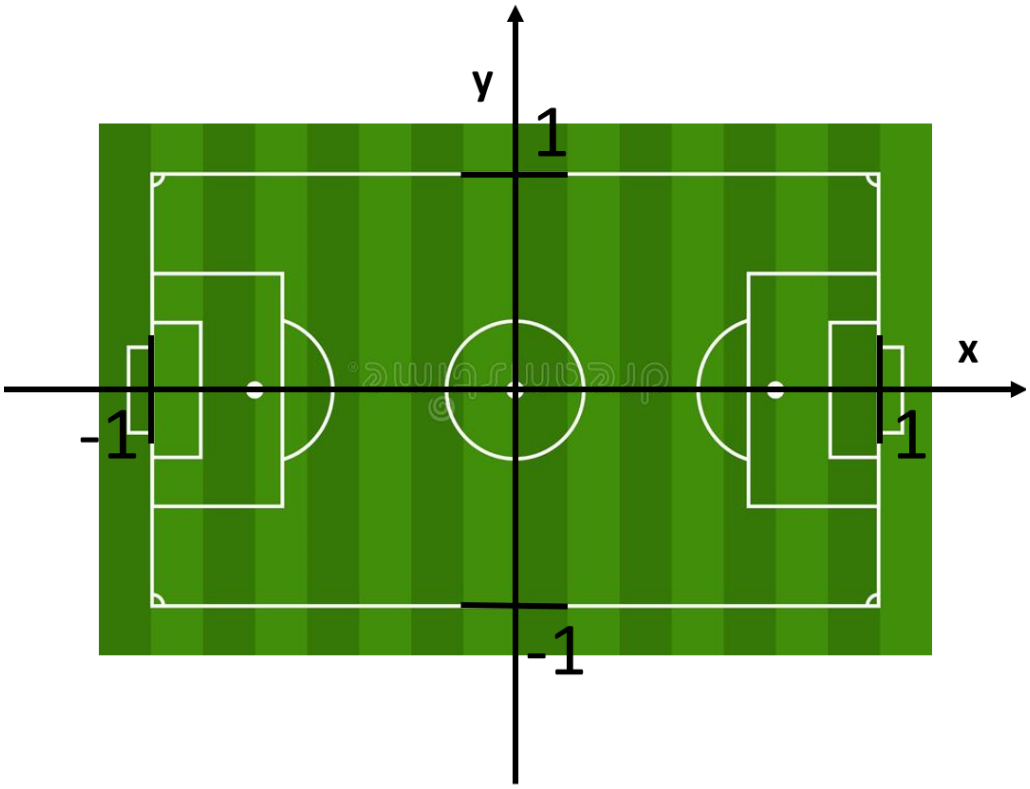
- ***Phase***: it is a sort of recall of the information in the *Phases* file related to the event, as the half in which is happened (1 or 2) and the team attacking from the left to the right of the field (*LeftTeamID*).

- ***Subjects***: it is a list of teams and players involved in the event so it has a quite variable content: in the case of *BallTouch* there are just IDs of team and player performing the balltouch, in the case of *Substitution* there are the IDs of players who exchange, while in the case of *Tackle* IDs of team and player performing the tackle are shown but also those which have suffered it.

- ***PositionNormalized***: it is a dictionary having as keys X and Y positions where the event happens (normalized because they are both between -1 and 1).

2.2 Position extraction

Given the data structure, the first goal is to capture the ball movements to map them on the playing field.
The ball position is contained in a dictionary having as keys the two-dimensional axis X and Y and as values the position on that direction.
The values are coordinates of a Carthesian system where the origin is located in the center of the field and positions are normalized on intervals $[-1, 1]$, so the field is represented by a square $[-1, 1]^2$ also if in fact is a rectangle (whose actual size is contained in the *MatchInfo* file); this is a key point to keep in mind for next steps.



2.2.1 Single match

The extraction of ball positions should be done match by match to achieve the relative statistics and to see its behaviour.
During a match there are lots of events for which the ball position is not registered, e.g. substitutions or warnings, so it is necessary to carry out a skimming of not available or out-of-field values.
The initial point is to consider just events having as *'Tags'* the keywords *'Balltouch'* and *'Goal'*, distinctly because there are some goals which are not ball touches (e.g. those of head), but then are joined (see code A.1 the appendix).
Subsequently there are controls about *'Position'*, to check if there are "not-a-number" positions (NaN), and *'Subjects'*, to check if there are all keys needed *'Verb'*, *'Performed'*, *'SubjectID'* and their values are valid (see A.2).
After this data cleaning process, it is possible to see its effects comparing the number of initial events with those remaining: from 3836 to 2222, so more than 40% of events would have hindered the analysis, leading to bias results.
From the final *"balltouches"* data frame a list of couples containing the valid ball coordinates during the match can be extracted (see B.2).

2.3 Field subdivision

The main aim is to map the extracted coordinates in the game field, not with their effective value (avoiding a not very significant great assembly of points)

but discretizing the field in different regions and assigning coordinates to the respective area.
In this way it is possible to lay the groundwork for a model that allows to represent field areas as states of a dynamic process.

The simplest solution is to create an uniform grid and to apply it on the field obtaining $m \times n$ areas, where m is the number of subdivisions along the longest direction and n those on the shortest direction; then every couple of coordinates is assigned to its area (x corresponds to the longest direction, y to the shortest one).

A problem that may emerge from this grid subdivision is the so-called "curse of dimensionality"; as m and n grow to create a more refined grid, also the dimension of the problem grows as $m \times n$, not linearly, causing computational inefficiencies.

Another problem is the robustness of results, that depends on the frequencies at which transitions between field areas are visited and that decreases with the increase of the problem dimension (see paragraph 3.3).

To avoid these problems the initial grid is 6×3, even if a more refined one could be surely better graphically (see paragraph 2.4.2).

In addition to the field areas, other states are needed to record goals scored and lost balls; but the last is necessary just when the analysis is focused on a team, while if both teams are considered the state "lost ball" for the team A coincides with a ball possession in a certain field area for the team B.

Moreover, the data cleaning process showed that some ball positions were registered out-of-the-field, e.g. before a corner or a throw-in, so it was created a sort of frame around the field to include these events, assigning them to the nearest field area.
The list B.3 includes examples of these cases, where position on X, Y or both do not stay within range $[-1, 1]$.
The idea is shown in the following image and coded in the function *field_ statistics*, that will be analyzed later.

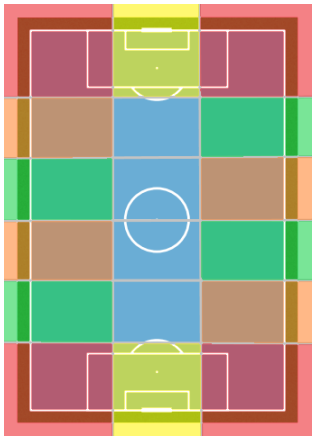
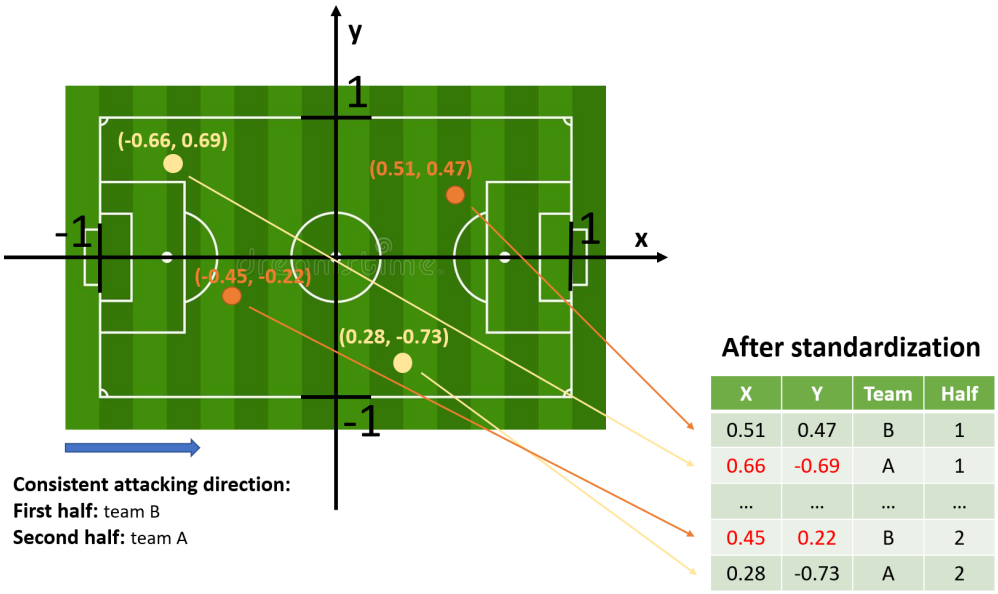


Figure 2.1: Areas distinguished by corner, behind football goal and lateral (both orange and green)

Another aspect to pay attention to is the field perspective for both teams: if a field area is seen by team A as an upper-right one, for team B is a lower-left one, but after the half-time it is in reverse, therefore it is essential to standardize these

changes of point of view.

The strategy was to take as reference system the offense from left to right or from top to bottom (respectively if in the field representation the long side is horizontal or vertical). Then the ball coordinates for the team whose offense is consistent with the reference system are let unchanged while the ball coordinates of the other team are mapped using the function $f : (x, y) \to (-x, -y)$, that is just their reflection with respect to the origin of the Carthesian reference system. For example let's assume that the analysis is about team A and that in the first half it is attacking from the right to the left of the field; then f is applied to ball coordinates of the first half, but not to those of the second half, because after the break there is the change sides so the team A will attack consistently to the reference system.



2.3.1 Single team in a match

In this framework the focus is to analyze the match behaviour from the viewpoint of one of the two teams; then there are three types of states in which the ball could stay:

- **goal**: to find goal events the function *is_ goal* is used (see A.3), which simply check if there is the string *'Goal'* in the columns *'Tags'* of the event.
  After this event naturally the state will be *lost* because the ball will pass to the other team (except very rare cases where the goal scored by team A is the last event of the first half and the second half kick-off is beaten by team A itself); so *goal* is an almost surely non-recursive state.

- **lost**: after selecting the events related to that team, to find when the ball is lost by the team the idea is to check indices with the function *is_ consequential* (see A.4) such that if the difference between an index and the following is greater than one it means that in the middle there are events related to the other team, so the ball was lost.
  A key point is that a single state *lost* could correspond to many events of the other team, so there is a state *goal* or *position*, then a state *lost* and again a

state *goal* or, most probably, *position*; for this reason *lost* is a non-recursive state.

- **position (X,Y)**: in this case the values of the coordinates X and Y are simply shown.

A way to represent the connection between states is a graph, useful also to better understand the transition process that will be analyzed in the next chapter.

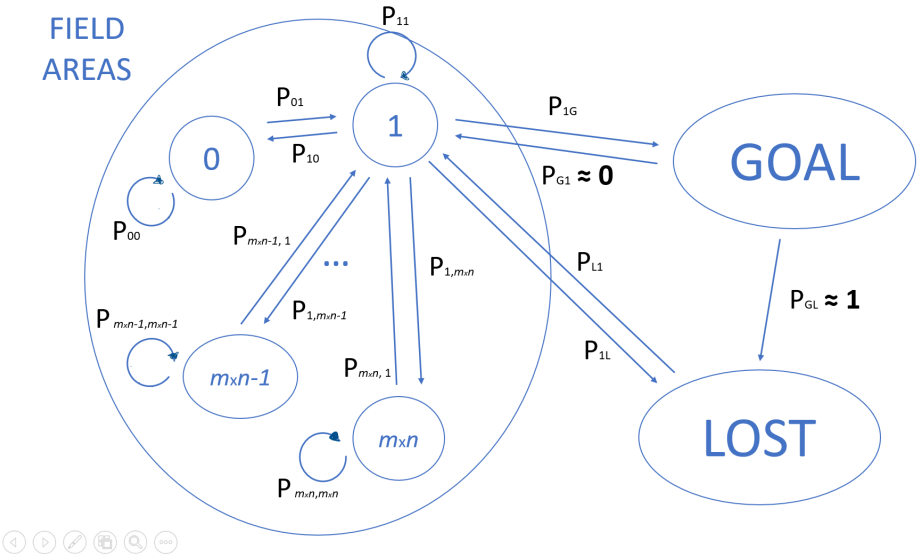


Figure 2.2: To avoid too much confusion not all areas are drawn: there are just the connections between the state 1 and those drawn and self-loops on "field states", because *goal* and *lost* are not recursive states.

Then, to assign these states, the piece of code A.5 is used, where *team_A* is a data-frame containing all events related to the team A.
It is really important to notice that counts the order in which if conditions are written in the for cycle: indeed usually both *goal* and *lost* events have their *position* so if that condition was written before, it could create misunderstandings. Similarly *goal* should precede *lost* because in the case of own goal the ball is not in possession, so if they were inverted the states would be *lost*, losing the fact that a goal was scored.

This framework is included in a function *find_states* (see A.6), which takes as input the json file with all information and the team to be analysed and it provides as output a list of states and an array of left team, the team playing from left to right, practically about one half with a team and the remaining, after the half-time, with the other.

Then results are used to find the field area for each state in the list through the function *field_statistics* (see A.8), which takes as input the number of subdivisions of long and short field side, the outputs of previous function *states* and *left_team* and again the team, recalling that the focus is actually the single team in the match.
Instead outputs are two numpy arrays:

- **field_zone**: it contains the field areas of states provided as input, so the length is the same of the list *states* and the possible values are $m \times n$ inside the game field plus two dummy areas, the *lost* one, signed by $m \times n + 1$, and the *goal* one, signed by $m \times n + 2$.

- **count**: it counts of how many times events were in the areas, so it is a vector with length $m \times n + 2$ whose sum on items is equal to *field_zone* length.

The matching between states and field areas is immediate in the case of *lost* $\longleftrightarrow m \times n + 1$ and *goal* $\longleftrightarrow m \times n + 2$, while is not so easy with *position* states. The first step is to understand if the team considered is attacking from left to right or not, done through the condition (see A.7), where after the else is applied the reflection function to the position.
Then there are two for cycles which run through all the field zones searching the one corresponding to the position provided; it is important to notice that in the case of contour areas are added controls using if conditions to check if the position is eventually in a frame around the field.

2.4 Results representation

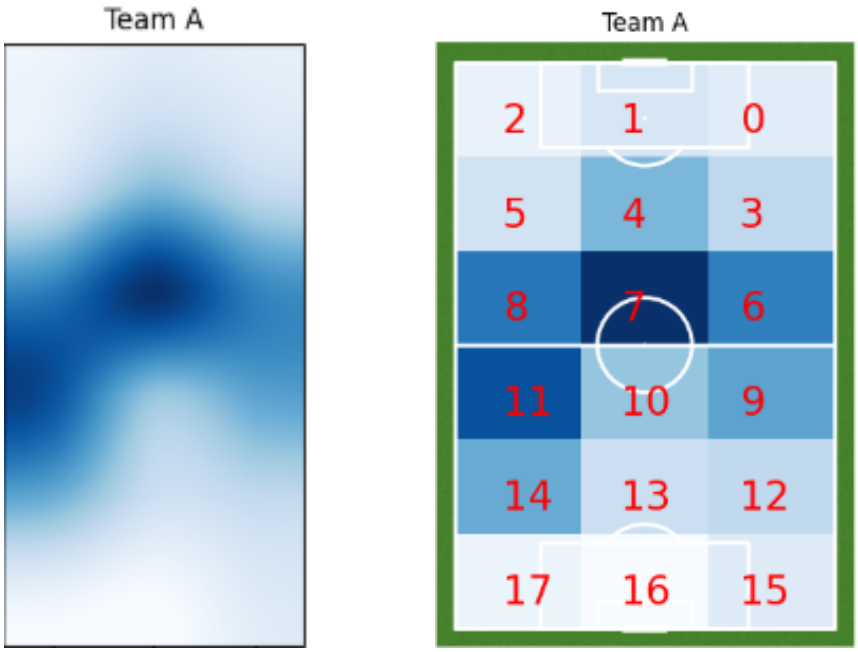
Until now the focus was on the extraction of information from data to better understand the behaviour of the match, but it is equally important the right representation of what was found to make it understandable even to people who have not directly worked there or who do not have the technical means to understand it.
A good graphical representation tool could be the heatmap, in which the vector *count* is used to create a grid.
Two different types of heatmap can be plotted:

- **blurred**: it is more qualitative and useful to see the statistics as in a continuous framework

- **defined**: it is more quantitative, because the subdivision in areas and the relative values are clear,

The function *plot_statistics_on_field* (see A.9) shows the heatmaps, taking as input the dimension of the subdivision, the vector *count* and the subject of the statistics (called team because in the first part statistics were on teams, but they can be also on the entire tournament), while the outputs are precisely the heatmaps.

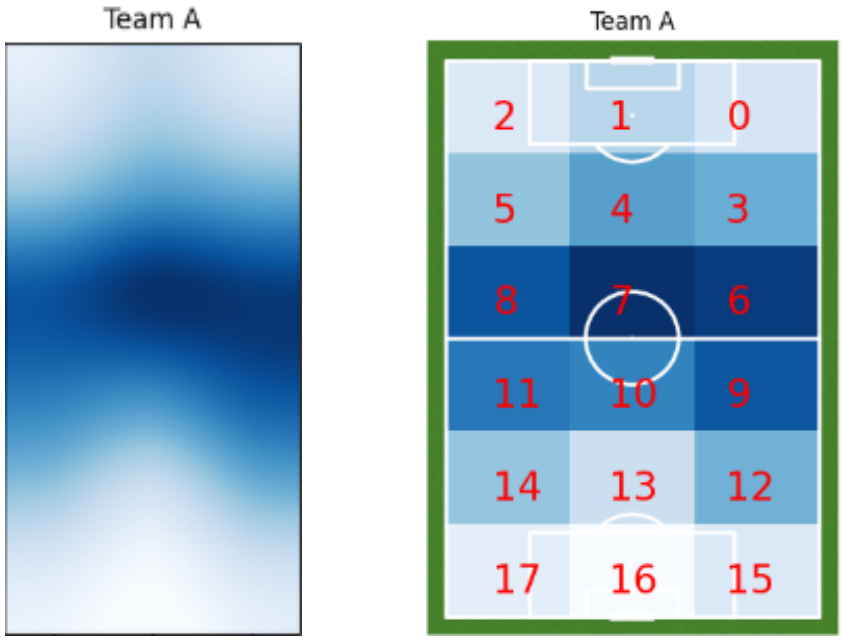
These are an example of output applied to a match played by the team A displaying the offense behavior during the game (from top to bottom).



2.4.1 Single team in all matches of a tournament

It is possible to extend the analysis done on a single match to the entire tournament, putting together the information arising from all games played.
It could be useful to better understand the behaviour of the team not just in a single apparition but on average along the competition.
The idea is to collect all matches played by a team and to iterate on them the process described above.
To avoid the manual search in the tournament folder the function *glob* of the homonymous package is used (see A.10), passing the team code as searching term.

Practically the *field_zone* vectors of all matches found and saved in *path_list* are linked one to the other as a single match throughout the tournament.
This solution leads to a little bias passing from a match to another, but it is negligible with respect to the chained vector dimension.
Instead the *count* vector is the resulting sum of count vectors of all matches.
Then applying it to all tournament matches of the team A are obtained the following plots



It is interesting to compare the performance of two teams, understanding the different ways to play during the tournament.

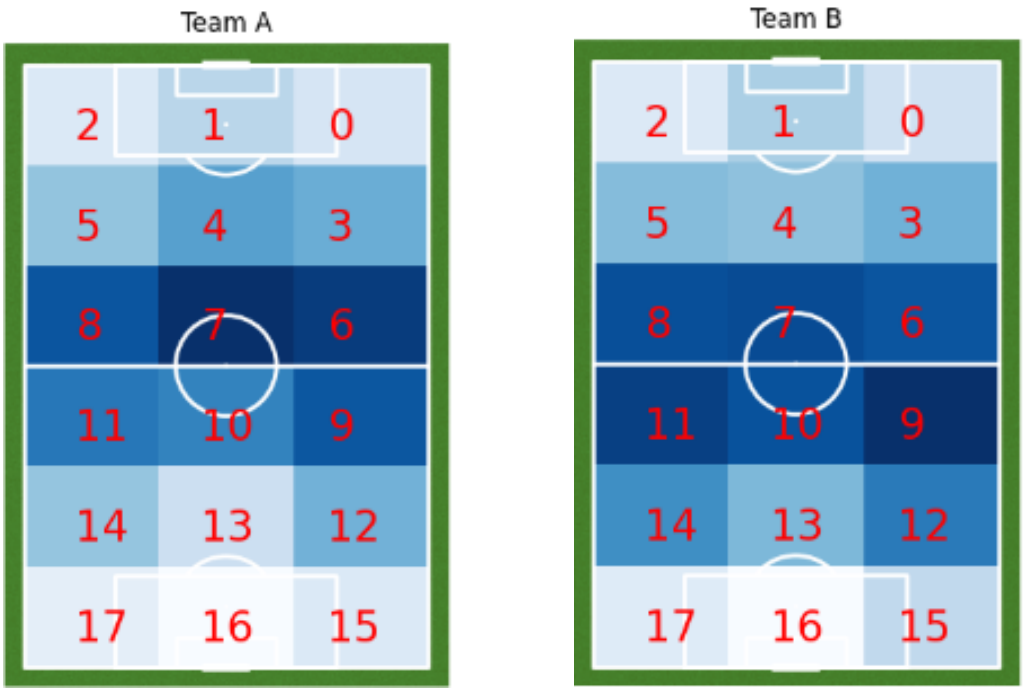


Figure 2.3: The game of team A is more concentrated in the classic construction area of the game, the number 7, while that of team B has a more distributed maneuver in the central areas of the field

2.4.2 Grid refining

Until now the field has been subdivided into 6×3 areas to avoid data sparse problems, which may emerge when considering just a single or not many matches, e.g. in the case of a team playing just an initial part of the tournament before being eliminated.
A possible refinement of the grid is obtained duplicating subdivisions in both dimensions, resulting in an almost quadruplication of states (from 20 to 74); then *field_zone* and *count* vectors are recomputed changing parameters m and n of *field_statistics* and *plot_statistics_on_field* function to show results (of a single match on the left, of the tournament on the right).

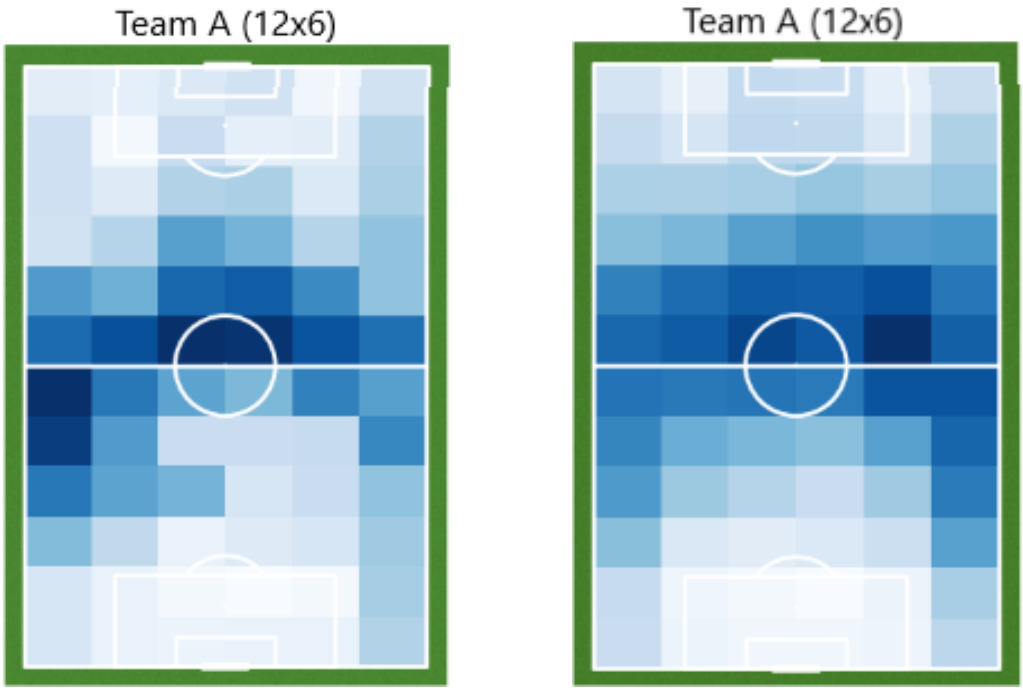


Figure 2.4: The left plot has some strange behaviours on the side lanes and it is much more irregular than the right one because there are far fewer observations, less than 10 in many areas, arising the problems mentioned above, while with more data the graphical result is better than with 6×3 grid.

Chapter 3

Markov chain

3.1 Transition matrix

Previously, the vectors contained the field areas where events happened, while now begins a new phase of the work; the idea is to create transition matrices representing the passage from one state to another, modeling the system not with just a list of states where the ball is but more dynamically, focusing on the interaction between different areas.

Matrices are structured having the starting states on rows and the final ones on columns, so they are squared with dimensions $[m \times n + 2]^2$

3.1.1 Frequencies matrix

The first type of transition matrix is the frequencies one, in which the general entry (i, j) represents how many times the ball passed from field area i to field area j.
It is quite easy to create it because it is enough to consider *field_ zone* previously found and take couple by couple all its items and starting from a zero matrix add one to the entry (i, j) every time there is a couple of consecutive areas i and j.
To do it the function *transit_ matrix* is used (see A.11), where the last part of the function is for saving the frequencies matrix in an Excel file.
This process works well especially in the case of single team in all matches of a tournament, obtaining the a large sample of frequencies; here there is that referred to the team A.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	89	27	1	40	10	6	9	3	0	1	1	0	0	0	0	0	0	0	60	0
1	34	177	40	36	68	49	18	10	9	5	18	4	0	0	0	0	0	0	155	0
2	0	24	107	1	11	50	2	3	7	0	3	2	0	0	0	0	0	0	61	0
3	25	21	2	284	65	7	113	39	5	7	2	0	0	0	0	0	0	0	80	0
4	6	30	6	67	379	64	62	144	82	2	10	4	0	3	1	0	0	0	59	0
5	0	29	26	7	81	372	6	37	142	0	1	8	0	0	2	0	0	0	68	0
6	5	8	0	84	42	8	704	137	19	170	36	4	11	5	1	0	0	0	123	0
7	0	3	0	18	72	20	144	761	158	83	148	61	12	8	4	0	0	2	85	0
8	0	4	4	1	58	88	20	162	736	2	41	217	0	5	16	0	0	0	106	0
9	0	0	0	2	4	0	113	34	4	601	116	11	125	26	2	12	1	1	112	0
10	0	1	0	0	5	0	17	105	23	87	507	118	48	66	38	5	1	9	104	0
11	0	1	0	1	4	7	3	47	135	9	110	715	4	21	155	1	1	2	124	0
12	0	0	0	0	0	0	1	3	0	69	23	0	358	36	6	57	13	4	125	0
13	0	0	0	0	1	0	1	2	1	4	19	10	30	225	33	12	17	19	143	3
14	0	0	0	0	0	0	0	1	3	2	18	82	2	56	433	4	13	70	131	0
15	0	0	0	0	1	0	0	0	0	0	1	0	33	6	0	58	20	3	95	2
16	0	0	0	0	0	0	0	0	0	0	0	1	0	3	2	5	13	4	78	13
17	0	0	0	0	0	0	0	0	0	0	0	2	0	2	40	2	27	89	109	0
18	88	298	85	110	118	108	144	88	136	122	78	101	73	58	82	63	13	68	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0

So for example during the tournament players of team A have passed the ball from area 8 to area 5 for 88 times, lost the ball for 61 times from area 2 and scored a goal from area 13 for 3 times.

This matrix is also useful to verify the successful cleaning of the data because it can be noticed that transition from *lost* states to itself and from *goal* states to itself are both zeros, due to the fact that they are non-recursive states, as well as transitions from lost to goal.
Finally the sum of goals scored by team A during the competition is visible on the cell $(19, 18)$, the transition from *goal* to *lost*, rather than calculating the sum of values in the last column, including transition from all states to goal (clearly they are equal).

3.1.2 Probabilities matrix

The second type of transition matrix is the probabilities one, in which the general entry (i, j) represents the probability that the ball passed from field area i to field area j.
It is really similar to the previous but, as seen in paragraph 1.1.3, a normalization is needed because probabilities of going to another state must sum to 1; this operation is implemented in the function *prob_ origin_ matrix* (see A.12), which takes as input the frequencies matrix previously found and normalizes it with respect to rows (precisely to obtain sum on columns equal to 1 for every row).

Once again the result related to the team A can be shown in the following matrix:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0,36032	0,10931	0,00405	0,16194	0,04049	0,02429	0,03644	0,01215	0	0,00405	0,00405	0	0	0	0	0	0	0	0,24291	0
1	0,05457	0,28411	0,06421	0,05778	0,10915	0,07865	0,02889	0,01605	0,01445	0,00803	0,02889	0,00642	0	0	0	0	0	0	0,2488	0
2	0	0,08856	0,39483	0,00369	0,04059	0,1845	0,00738	0,01107	0,02583	0	0,01107	0,00738	0	0	0	0	0	0	0,22509	0
3	0,03846	0,03231	0,00308	0,43692	0,1	0,01077	0,17385	0,06	0,00769	0,01077	0,00308	0	0	0	0	0	0	0	0,12308	0
4	0,00653	0,03264	0,00653	0,07291	0,4124	0,06964	0,06746	0,15669	0,08923	0,00218	0,01088	0,00435	0	0,00326	0,00109	0	0	0	0,0642	0
5	0	0,03723	0,03338	0,00899	0,10398	0,47754	0,0077	0,0475	0,18228	0	0,00128	0,01027	0	0	0,00257	0	0	0	0,08729	0
6	0,00368	0,0059	0	0,0619	0,03095	0,0059	0,51879	0,10096	0,014	0,12528	0,02653	0,00295	0,00811	0,00368	0,00074	0	0	0	0,09064	0
7	0	0,0019	0	0,0114	0,0456	0,01267	0,0912	0,48195	0,10006	0,05256	0,09373	0,03863	0,0076	0,00507	0,00253	0	0	0,00127	0,05383	0
8	0	0,00274	0,00274	0,00068	0,03973	0,06027	0,0137	0,11096	0,50411	0,00137	0,02808	0,14863	0	0,00342	0,01096	0	0	0	0,0726	0
9	0	0	0	0,00172	0,00344	0	0,09708	0,02921	0,00344	0,51632	0,09966	0,00945	0,10739	0,02234	0,00172	0,01031	0,00086	0,00086	0,09622	0
10	0	0,00088	0	0	0,00441	0	0,01499	0,09259	0,02028	0,07672	0,44709	0,10406	0,04233	0,0582	0,03351	0,00441	0,00088	0,00794	0,09171	0
11	0	0,00075	0	0,00075	0,00299	0,00522	0,00224	0,03507	0,10075	0,00672	0,08209	0,53358	0,00299	0,01567	0,11567	0,00075	0,00075	0,00149	0,09254	0
12	0	0	0	0	0	0	0,00144	0,00432	0	0,09928	0,03309	0	0,51511	0,0518	0,00863	0,08201	0,01871	0,00576	0,17986	0
13	0	0	0	0	0,00192	0	0,00192	0,00385	0,00192	0,00769	0,03654	0,01923	0,05769	0,43269	0,06346	0,02308	0,03269	0,03654	0,275	0,00577
14	0	0	0	0	0	0	0	0,00123	0,00368	0,00245	0,02209	0,10061	0,00245	0,06871	0,53129	0,00491	0,01595	0,08589	0,16074	0
15	0	0	0	0	0,00457	0	0	0	0	0	0,00457	0	0,15068	0,0274	0	0,26484	0,09132	0,0137	0,43379	0,00913
16	0	0	0	0	0	0	0	0	0	0	0	0,0084	0	0,02521	0,01681	0,04202	0,10924	0,03361	0,65546	0,10924
17	0	0	0	0	0	0	0	0	0	0	0	0,00738	0	0,00738	0,1476	0,00738	0,09963	0,32841	0,40221	0
18	0,04801	0,16258	0,04637	0,06001	0,06438	0,05892	0,07856	0,04801	0,0742	0,06656	0,04255	0,0551	0,03983	0,03164	0,04474	0,03437	0,00709	0,0371	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 3.1: For example the probability to pass from field area 17 to 14, that is a back passage on the right lane, is almost 15%, while that of scoring a goal from area 13 is less than 1%.

3.1.3 Total tournament edition

The tools just seen can be applied to the outermost level of this analysis, the entire tournament edition; the results may be useful for comparison with those of other years but also to have a sort of average approach to attacking of the teams along the competition.

The structure of A.13 is the same for the analysis about a single team during the competition but extended to all teams, similar to what happened passing from one match of a team to all its matches.
In this case an external for cycle is added, iterating on a dictionary composed by all teams participating in the competition and their relative code.
It is possible to see graphically the average way to play of teams during the competition (the first couple), but also a comparison with the approach to attacking by single teams (the second couple):

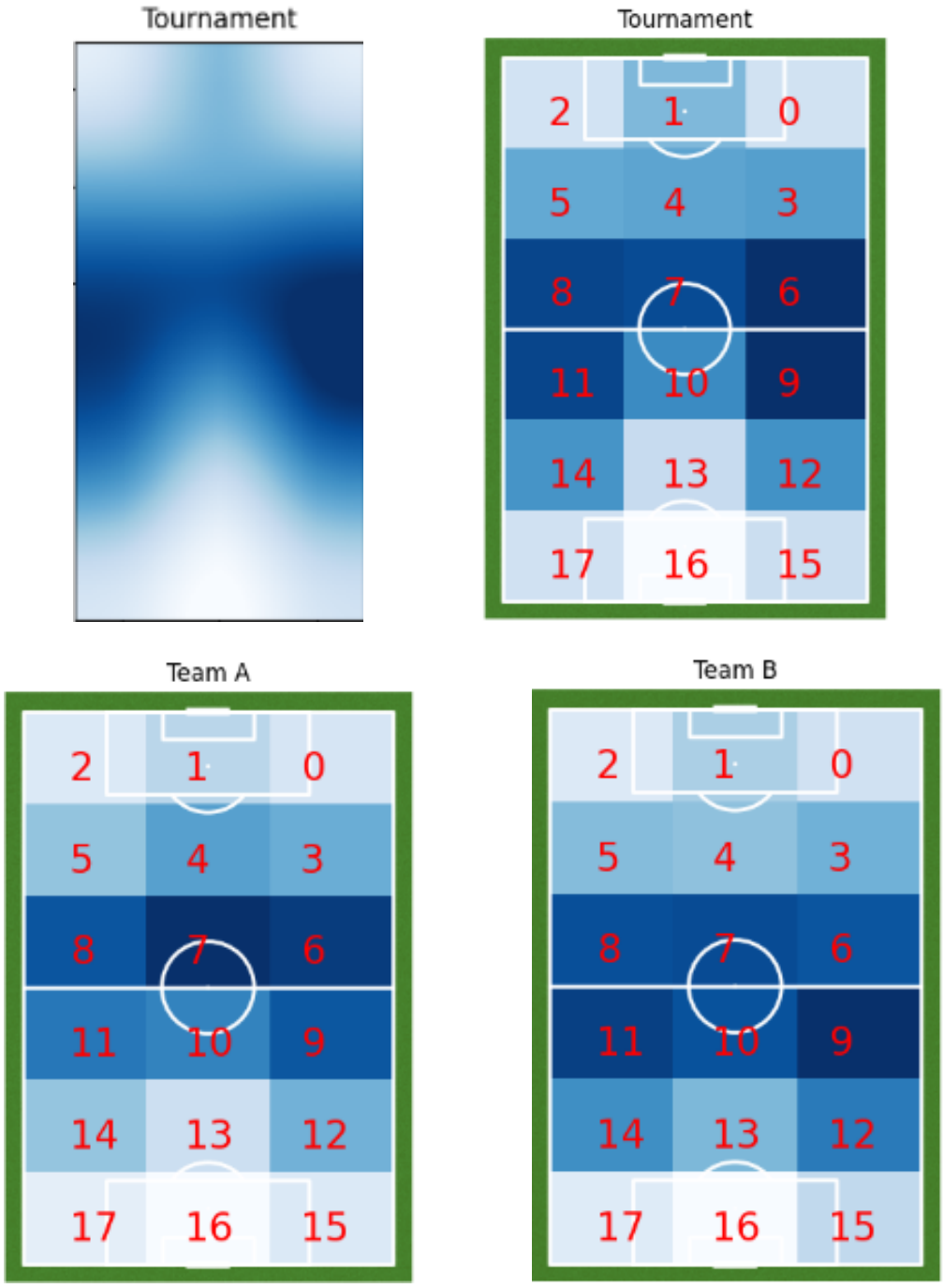


Figure 3.2: It is interesting to notice that the tournament plot is almost symmetric, with a negligible dominance of on the left lane, while for team A this dominance is more pronounced, as it can be seen from the darker colour of areas 3 and 12 with respect to 5 and 14 respectively. Moreover seeing the lighter colour of area 6 and the darker of area 11, the team B seems to slightly prefer a more advanced position on the right side.

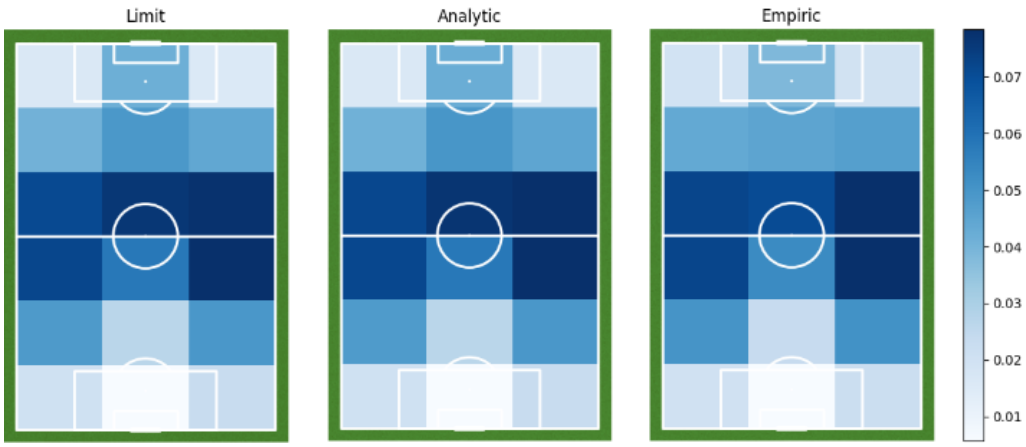
3.2 Stationary distribution

The analysis on the entire tournament has led to the study of a new tool that could be useful in the continuation of the work: the stationary distribution.
Until now charts were about how many times the ball passed on different field areas, while here the focus is to find the stationary probabilities to be in field areas given a random instant of the match.
This aim can be attained in three different ways:

- **limit distribution**: based on the fact that the chain is ergodic because it is finite, irreducible (taking every couple of states they are mutually reachable) and aperiodic (the chain period is equal to 1), so the limit distribution is also the stationary one.
- **analytic distribution**: based on the solution of the linear system $\pi = \pi P$, derived by the definition of stationary distribution.
- **empiric distribution**: more pratical because the stationary distribution is obtained normalizing the *count* vector found in the previous subsection.

In the piece of code A.14 after a common first part to import the transition matrix, let just notice that in the limit method starting from a vector in which all states are equiprobable, the transition matrix is iterated for k times, where theoretically k tends to infinity but in practice it only takes about ten iteration to reach convergence.

The comparison between the results can be done again through the function A.9, obtaining



The stationary distribution obtained by the limit distribution and by solving the linear system are identical, while the empiric is almost the same except for a little difference in the center-defensive areas of the field, that is a qualitative proof that the three methods are almost equivalent.
The elapsed times are almost negligible so this is not a valid discriminator, but imagining to be on a large scale the more efficient is the iterative one, because the analytic one requires the solution of a linear system, heavier than a matrix product, while the count one is the highest probably due to the import of the Excel file.

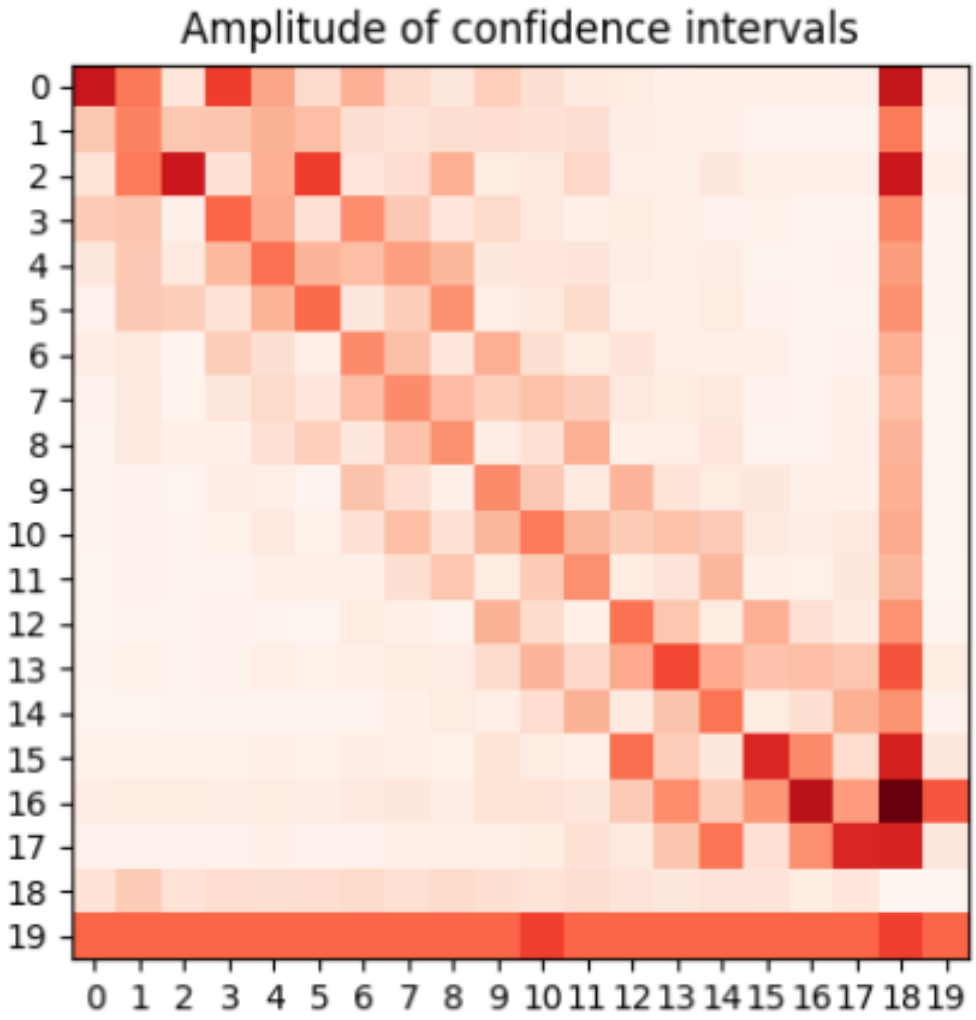
```
Limit time
Elapsed time: 0.001977 seconds.

Analytic time
Elapsed time: 0.006979 seconds.

Count time
Elapsed time: 0.031848 seconds.
```

The real differences between these methods are reliability and conditioning of the results; iterative and analytical ones are subject to the propagation of any errors present within the transition matrix, whose poor conditioning is not to be excluded given its construction based on empirical data.
So because of its nature, the empirical method is the most stable, since it does not depend on the transition matrix but only on the count vector in the various field areas.

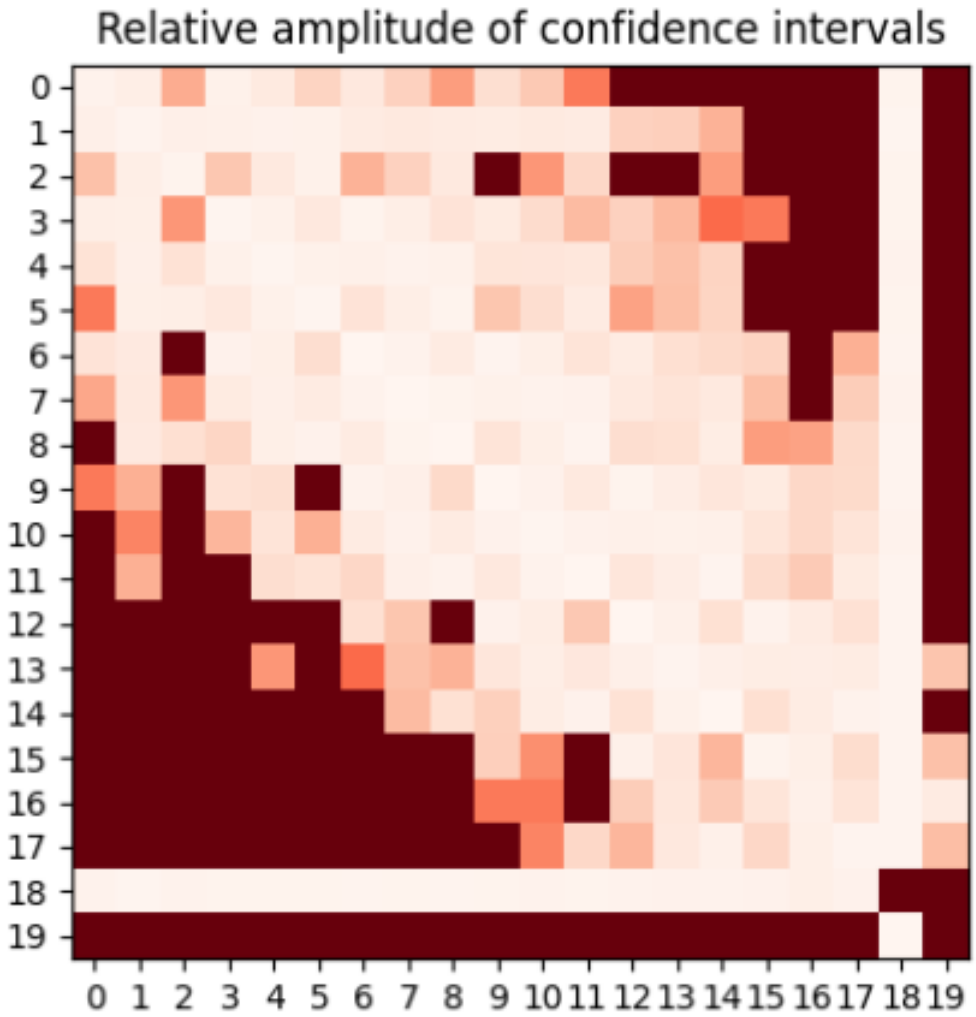
3.3 Application of robustness to probabilities matrix

The possible strategy presented in section 1.2 is implemented in the piece of code A.15, which starts with controls on inputs α and *counts* and initialization of n and k.
Then B (*chi2* variable), the new upper $(\alpha/k) \times 100$-th percentile of the chi-square distribution with one degree of freedom, and *delta*, which is the common term preceded by a minus or a plus in the confidence interval limits formulas, are computed.
$[\pi_i^-, \pi_i^+]$ can be computed in vectorial form (*region* variable) following again those formulas.

Before using function *multinomial_proportions_confint* (see A.15), it is necessary to import the frequencies matrix and to find the probabilities one (see A.16). Then a three-dimension matrix is initialized to save lower and upper confidence interval limits for all transitions.
But the function A.15 takes as input just one row at a time; thus *proportions* vector and confidence intervals resulting are coherent with probabilities computed in *prob_trans_matrix*.
Externally a for cycle is iterated on rows and results are saved on a "slice" of *region* (see again A.16); from these confidence intervals it is generated another matrix including their amplitude, a simple measure of goodness of the estimated probability.
The matrix can be visualized through an heatmap, which is quite different as long as expected: entries having high frequencies are those with larger confidence intervals.

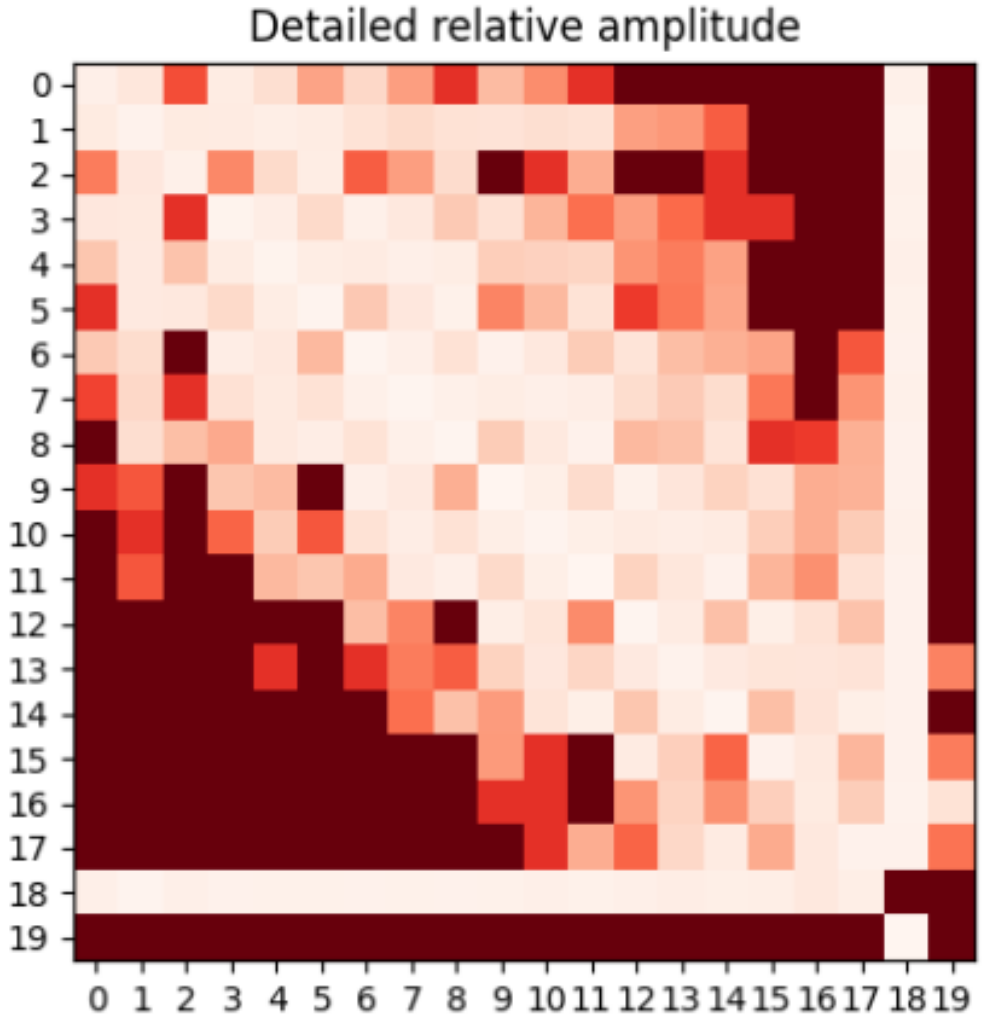


A possible explanation could be that high-frequency transitions are also the most likely ones, so a large absolute amplitude of the intervals turns out to be in fact small if evaluated relatively.
Then a new matrix *rel_ampl_matrix* is created in A.17, having entries equal to the ratio between entries of *amplitude_confint_matrix* and *prob_trans_matrix*. Moreover it is added an *if* condition to control the initial hypothesis *"values in 'counts' are greater than or equal to 5"*, because in this case the measure of robustness of these entries is not valid.
A *max_elem* is saved and their value is put at $2 \times max\_elem$ by hand to make the difference with respect to "valid" values.



Now the heatmap is much more similar to what expected, with low values near the diagonal and high values on those in upper-right and lower-left corners.
Row and column 18 are almost white and this could seems inconsistent with other states near the edges of the matrix, but it should be remembered that state $m \times n + 1$ corresponds to *lost*, which has an high frequency from and to all other states.

To better understand the behavior of relative amplitudes in the area near the diagonal it can be created a little more detailed heatmap:



3.4 Application of robustness to stationary distribution

The method presented in section 1.2 is also useful to understand the consistency of results about the stationary distribution.
Considering just the distribution found following the empirical method, which has been inferred to be the most stable, in the piece of code A.18 the function A.15 is applied to the *count* vector, previously found with A.14.
The absolute and relative confidence intervals for *stationary_count* vector (nothing but the normalization of *count*) are then computed in the for cycles obtaining the following plots

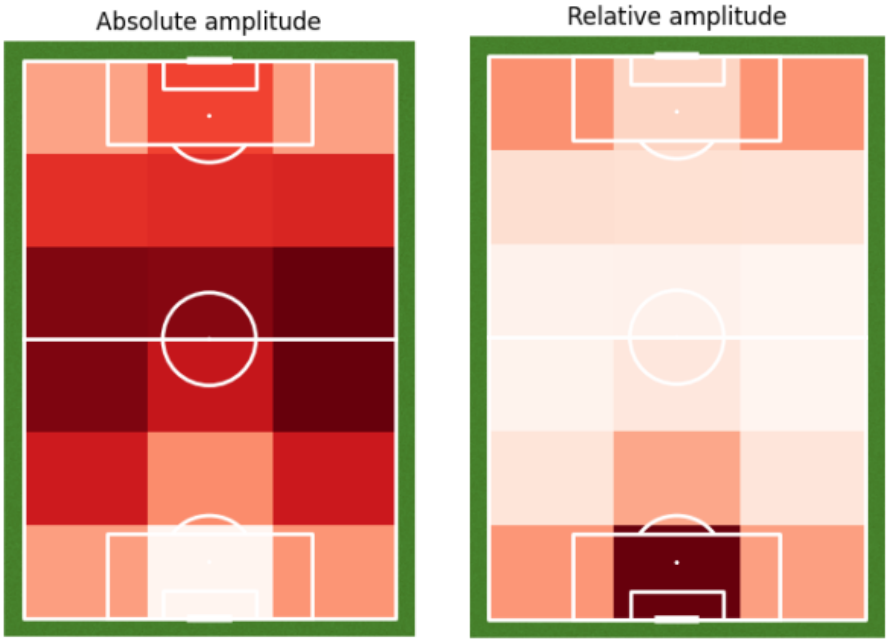


Figure 3.3: The pattern of absolute amplitude of confidence intervals is really similar to that of stationary distribution, stating that they seem directly proportional.
The relative amplitude instead reveals how the less visited areas are those with the highest values, especially the 16-th area, in front of the opponent's goal.
Its relative amplitude is more than 15% of the stationary distribution value, while others are lower than 10%.

Chapter 4

Application of expected threat to probabilities matrix

The aim is trying to replicate the idea of Karun Singh presented in section 1.3 exploiting what was previously found.
The starting point is quite different, because here the probabilities of moving or shooting were not distinguished from each other but only the passage from one state to another, while goal probabilities and transition matrix can be easily derived.
In particular, probabilities are the last column of the matrix because it includes all probabilities to pass from a field zone to the state *goal*.

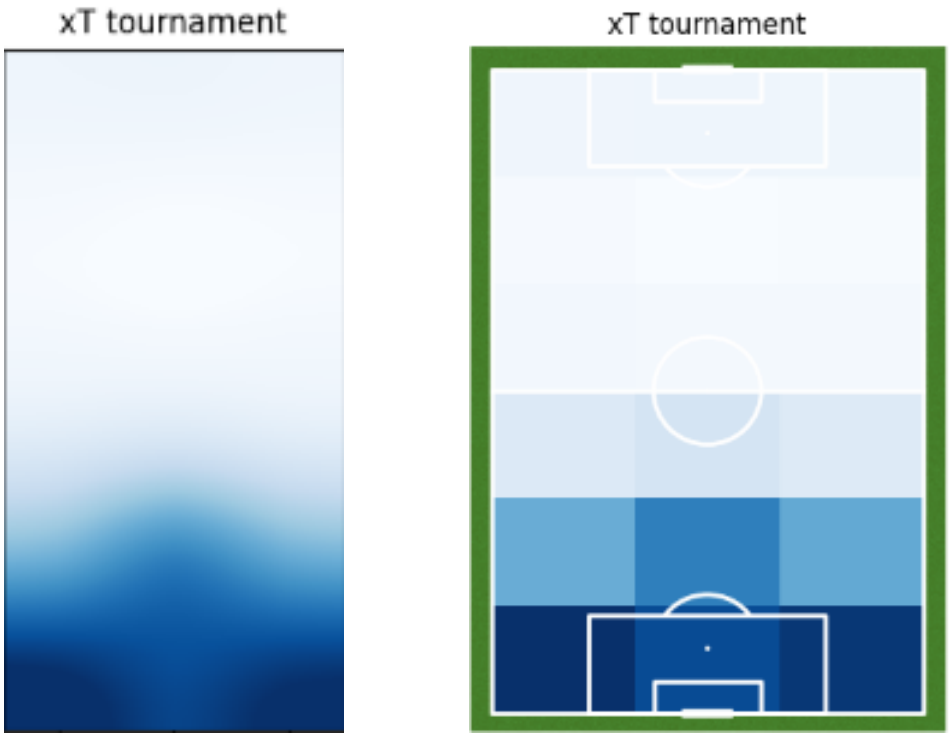
The function *exp_thr* (A.19) is used for this purpose, to find the expected threat and to save it in an Excel file; the input n represents the number of iterations, which is basically how many times is done the matrix-vector product between transition matrix and expected threat vector.

The reasoning behind this simplification is that at the iteration 0, that is the initial condition, the question should be "What is the probability of scoring a goal with another 0 passes?", therefore shooting directly from the area where the ball is; clearly the answer is given by the last column of the matrix.
In general for step k the question is the same with k passes, but it can be reduced to "What is the probability of scoring a goal with another pass from the state $k-1$?"; that is the explanation of the recursive formula

$$xT_{(x,y)} = \sum_z \sum_w \mathcal{T}_{(x,y)\to(z,w)} xT_{(z,w)}$$

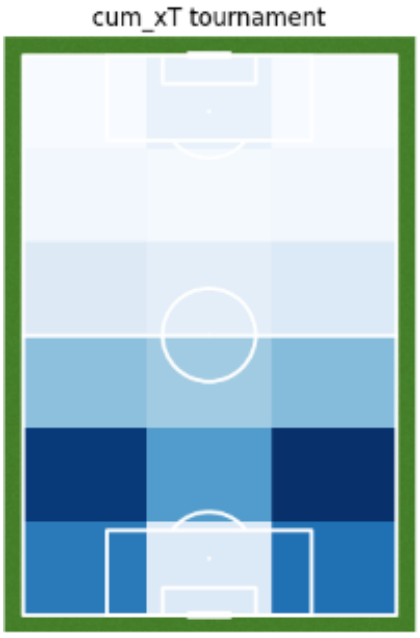
contained in the for cycle.

The xT matrix is obtained running this function with $n = 10$, that seemed to be a good parameter, and it is interesting to analyze the results: going forward with the iterations, there is a sort of normalization among all field areas, due to the fact that considering 10 iterations, so 10 passages before the goal, the initial area begins to become irrelevant.
For this reason, it was decided (in a qualitative way) to take the second iteration, which can be visualized through the usual function *plot_statistics_on_field*:



It is interesting a comparison between the expected threat found at the second iteration and goal probabilities, the initial condition, to appreciate the improving of information about the goal, passing from its mere realization to its construction in the last and usually more decisive steps.

A last consideration can be done regarding the normalization of the expected threat with respect to how many times the ball passes over areas, such that is given importance to areas more frequented, measuring it with *count* vector, to have a better idea of from where comes the majority of danger situations.



4.1 Comparison between expected threat and lost probabilities

It is interesting to spent few lines seeing the relationship between this two factors. The expectation is that they are directly proportional because when a team plays near the opposing area clearly the probability of losing the ball is higher than near its area; this is confirmed by the following chart, where the points correspond to every field area and their coordinates are the expected threat and the lost probability.

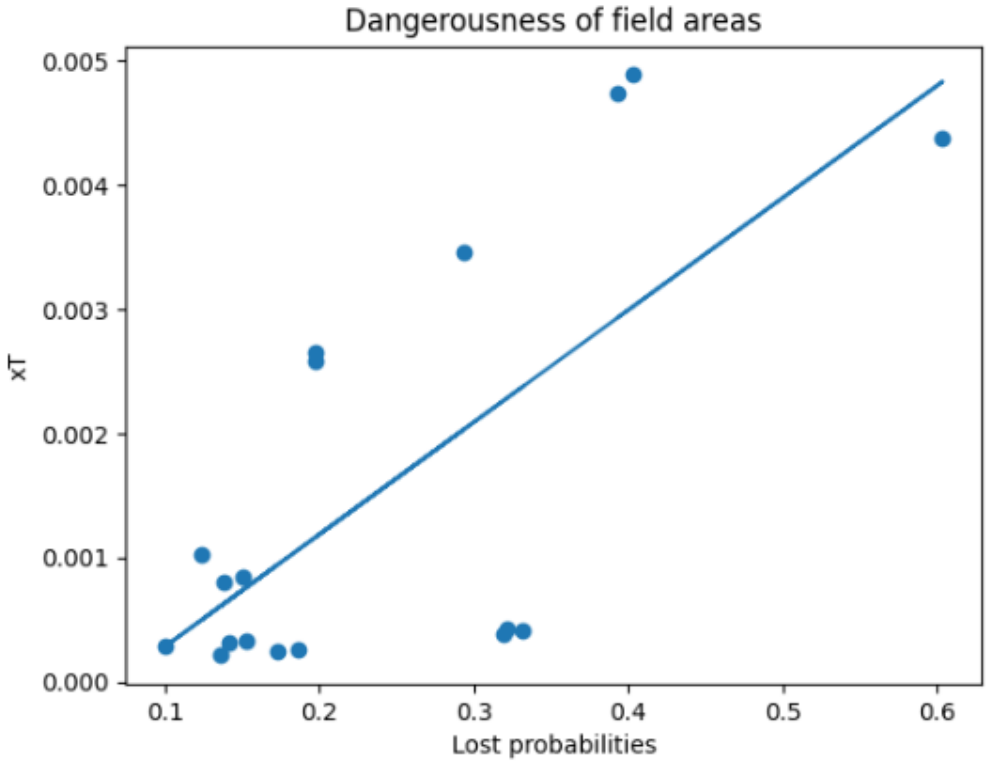


Figure 4.1: The increasing trend is shown by the regression line of these points.

There is a clear distinction between "safe" areas, characterized by low lost probabilities but also low expected threat, while there are areas more or less "inviting" depending on whether they are above or below the line. To understand which are those areas, the vectors should be analyzed, whose representation on the pitch is shown in the following charts.

So the worst areas are the three having lost probabilities between 0.3 and 0.4 but expected threat lower than 0.001, that are the number 0, 1 and 2.
The other group of "bad" areas is composed by those having lost probabilities between about 0.13 and 0.2 but expected threat lower than 0.0005, that are the number 3, 4, 5, 6, 8, while the field area 7 is at the beginning of the line.
The last not so inviting area corresponds to the isolated point with more than 0.6 as probability of losing the ball, that is the number 16.

Regarding "good" areas, there is a group of three points between 0.1 and 0.2 of lost probabilities and around 0.001 of expected threat including areas number 9, 10 and 11. The remaining five points over the line are the best areas in which playing on the field, because expected threat values are high relatively at the probabilities of losing the ball; they are the number 12, 13, 14, 15 and 17.

4.2 Match dominance

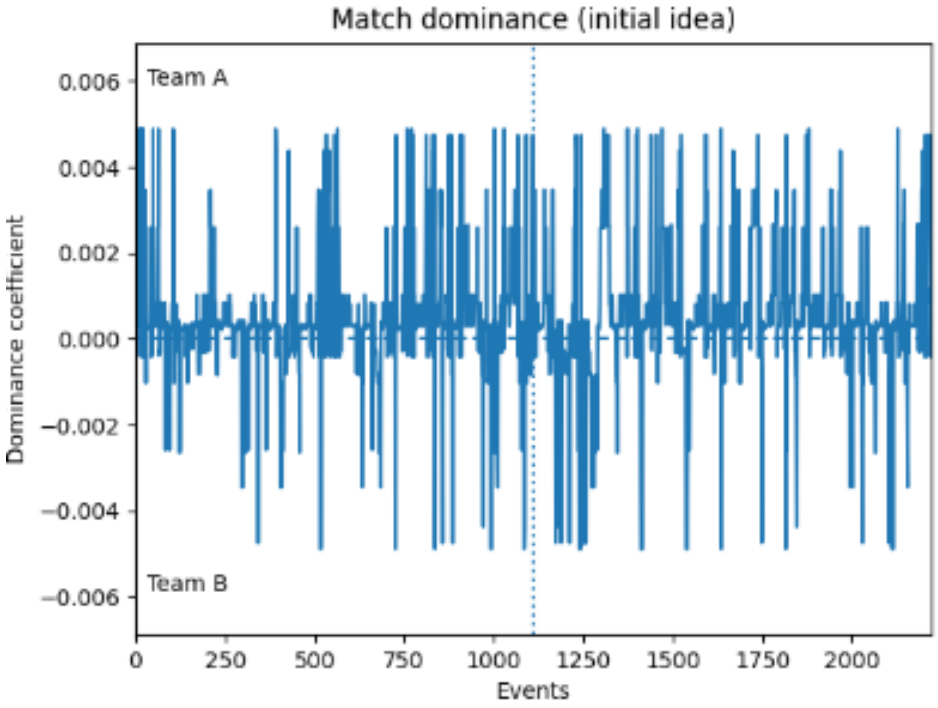
The expected threat can be applied to better understand the behaviour of a match, providing a more dynamic and interesting point of view of the crucial phases.
But considering the match from a complete point of view, it is not possible to continue the analysis thinking team by team, there are some changes to be carried out.

4.2.1 Both teams in a match

Firstly it disappears the state *lost*, because when a team loses the ball it means that the ball possession is now kept by the other team, so instead of *lost* there is a state *position* referred to the other team.
As direct consequence a new information about the team holding ball is needed; the new function *find_states_match* (A.20) has an additional output *team*, a list including the team relating to states.
So for each event the state, the team and the left team are known, grouped in three different lists.

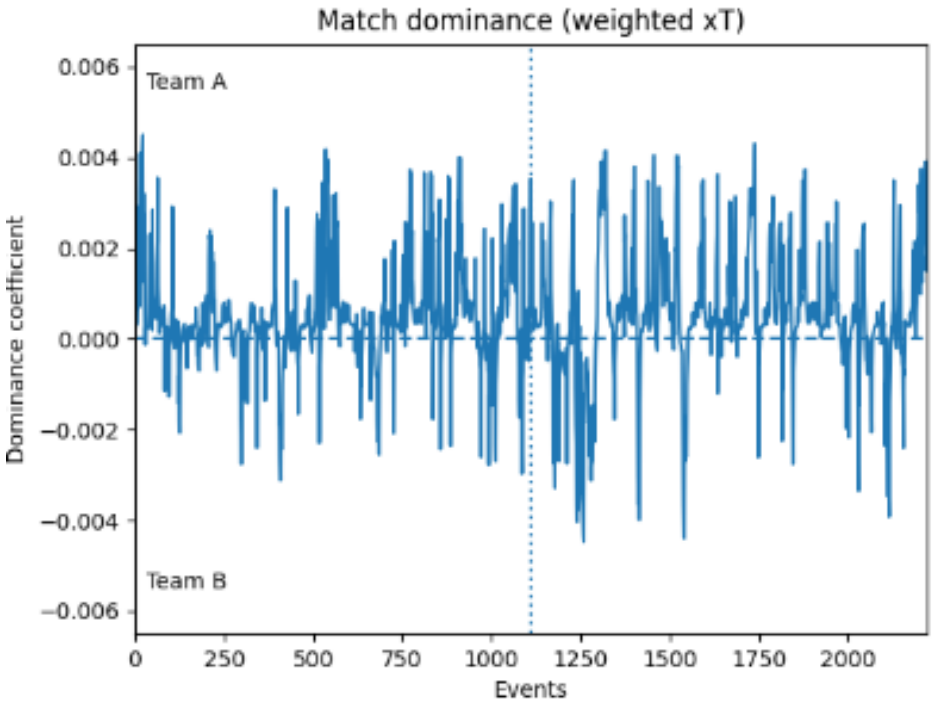
Secondly the team changes event by event, so in the new function *field_statistics_match* (A.22) the condition A.7 becomes A.21.
Finally the output *count* is omitted because it does not make much sense write down how many times the ball is in a certain area without distinction between teams.

So these functions provide the inputs of this process, which are applied to the expected threat found in the previous section following the piece of code A.23.
The idea is to iterate on *field_zone* items taking the expected threat in that field area with the positive sign if the team playing on it is the first team touching the ball (not very relevant here, it only counts graphically), the negative sign otherwise.
The representation of this result is not very significant; watching carefully can be noticed that the team represented at the top seems to dominate the other, but this is not enough.



Another idea was to compute the match expected threat with a weighted sum of the expected threat in the actual field area and the previous ones (preceded by plus or minus following the same criterion as before).
In this way a leveling in the chart due to equal values of expected threat is avoided.
In A.24 the first for cycle is a kind of initialization of the match expected threat, while the second includes the weighted sum.
It is designed in such a way as to halve the importance of the expected threat the further away from the current observation but to maintain the sum of weights equal to 1; the choice was to consider the actual and the three previous observations, so a 4-upla of weights satisfying these properties is $[8/15, 4/15, 2/15, 1/15]$.
Clearly it is not the unique possibility because properties are satisfied also by n-uple as $[24/45, 12/45, 6/45, 3/45]$ or $[27/40, 9/40, 3/40, 1/40]$ if the importance is not halved but divided into three or even $[4/7, 2/7, 1/7]$ if considered just the

two previous observations, but after a qualitative comparison between some of them the choice was $[8/15, 4/15, 2/15, 1/15]$.



The chart is slightly more understandable than the initial one but it is still difficult to get a better idea about certain moments of the game.

A solution could be to group values of expected threat with respect to their timestamp (it will be done by k minutes) in order to have a clearer chart.
In A.25 the time range of the match is divided in *time_segments*, on which the external for cycle iterates, while the range of the internal cycle is the entire *time_stamp* vector but the if condition makes sure that just the values on the i-th time segment are taken.

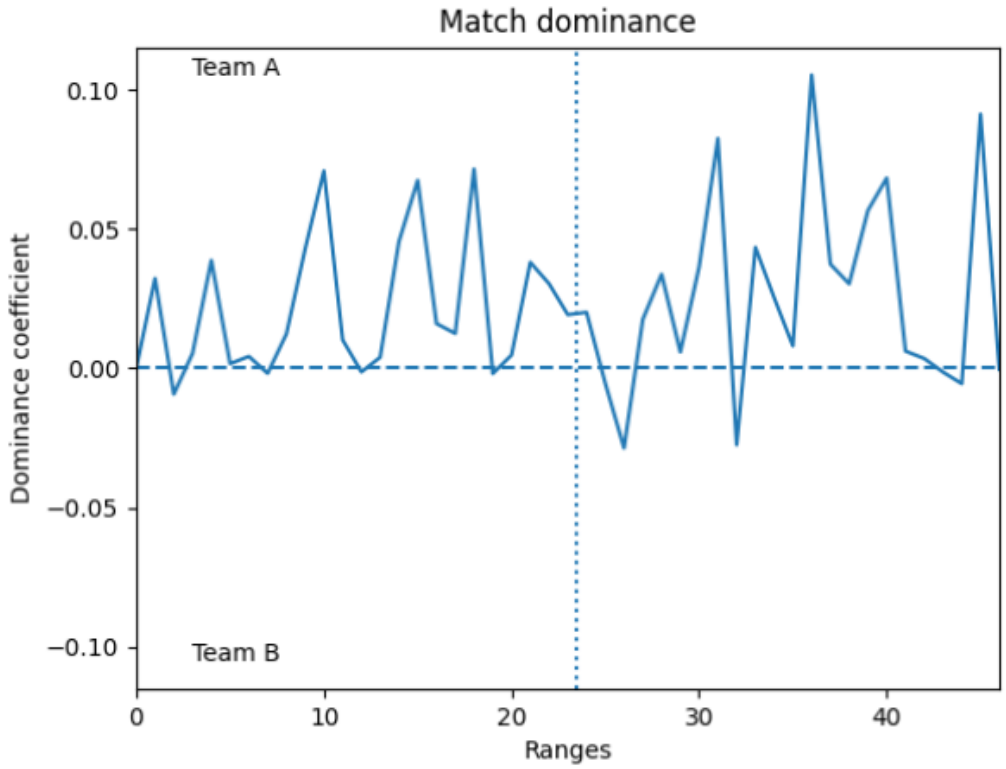


Figure 4.2: Ranges on the x-axis indicate the parts in which is splitted the match, that are nothing but the match duration measured in minutes divided by parameter k and overestimed.

The graphic representation of the case $k = 2$ is much better than before; now there is a clear dominance between of the team A with respect to the team B and the moments in which one team prevails over the other are more outlined.

A further improvement is to distinguish teams not just with the horizontal line but colouring areas created by their expected threat, to draw a dash line at the half-time and to set $k = 1$ so that numbers on the x-axis have the meaning of minutes of the match.

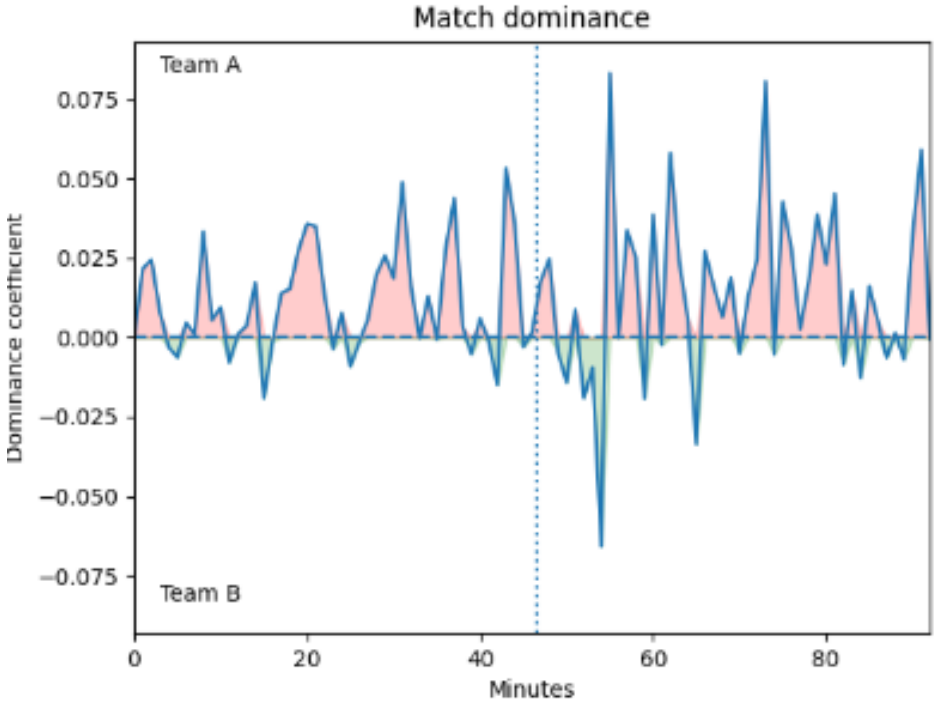
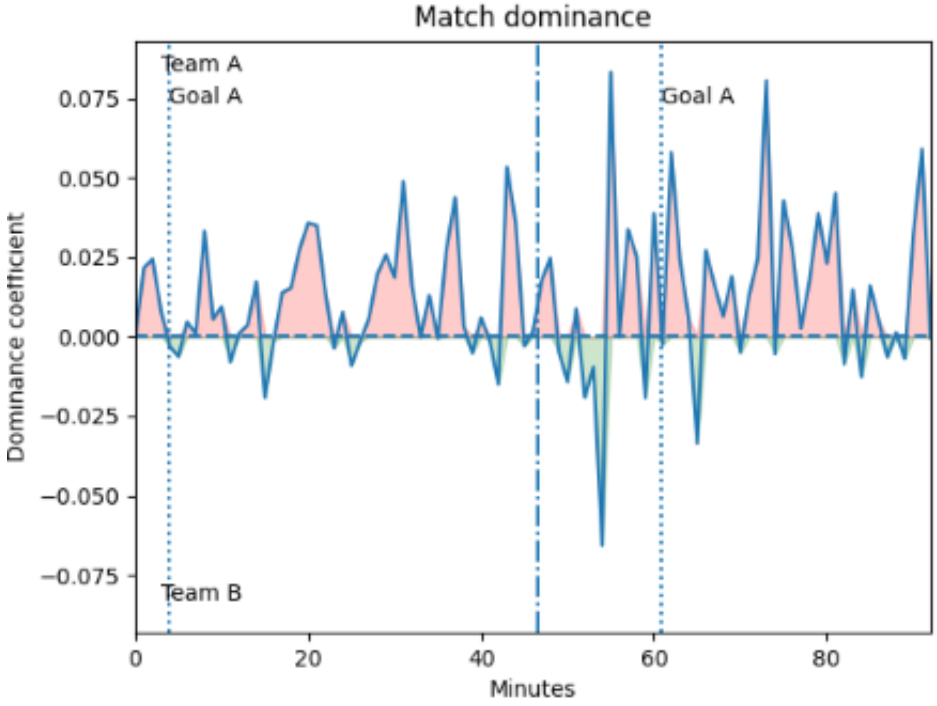


Figure 4.3: The result of the match was 2-0 for the team A

Finally it is possible to add vertical lines corresponding to the minute of the goals done, that could be useful to better understand the behaviour of the match; a team can reach the goal due to its dominance in the match or it can gain dominance thanks to the goal itself.

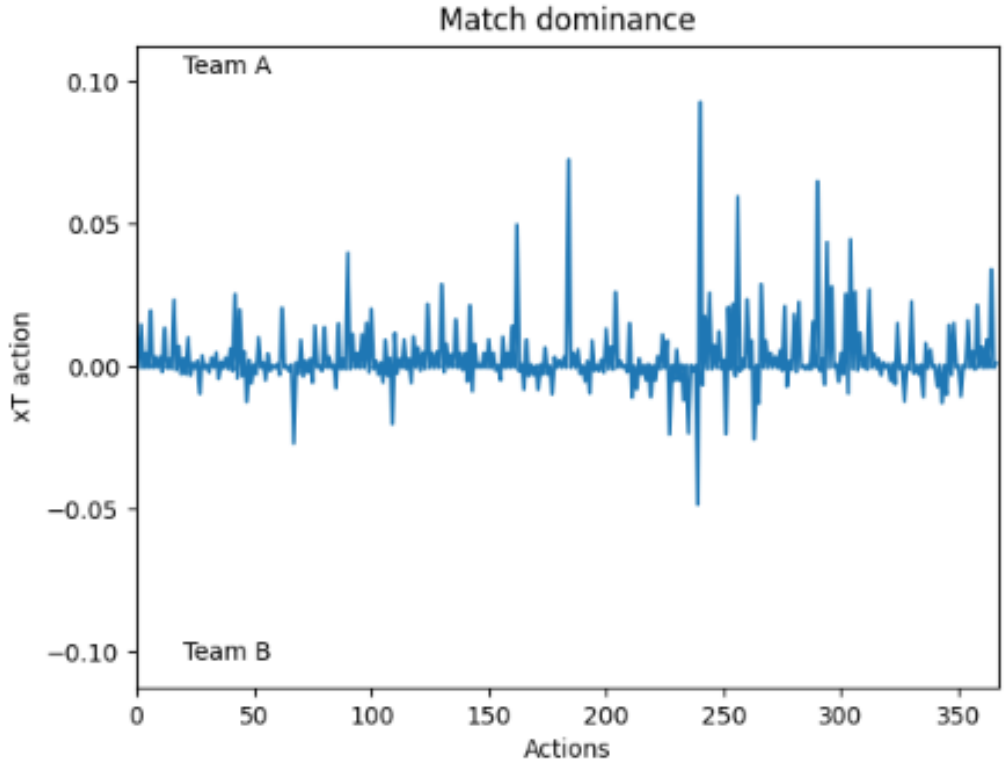


4.2.2 Subdivision by actions

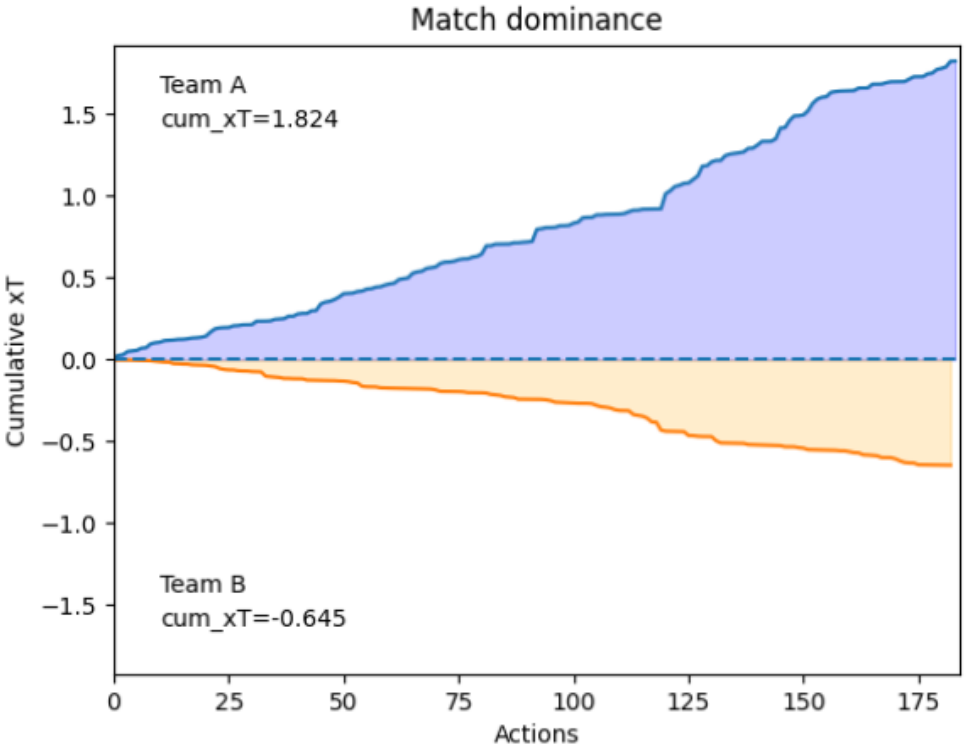
Another approach to the match analysis is to consider actions played in turn by teams, grouping consequential events of the same team in a unique action.
The aim is again to measure the dangerousness of teams during the match but not dividing it by minutes and "mixing" different attacks of the teams; now the strategy is to compute the expected threat of an action by doing the sum of the expected threat of every event that compose the action.
It seems to be a good trade-off between possession and verticalization because a long actions in areas away from the goal or dangerous but short actions do not represent a real dominance in the match.

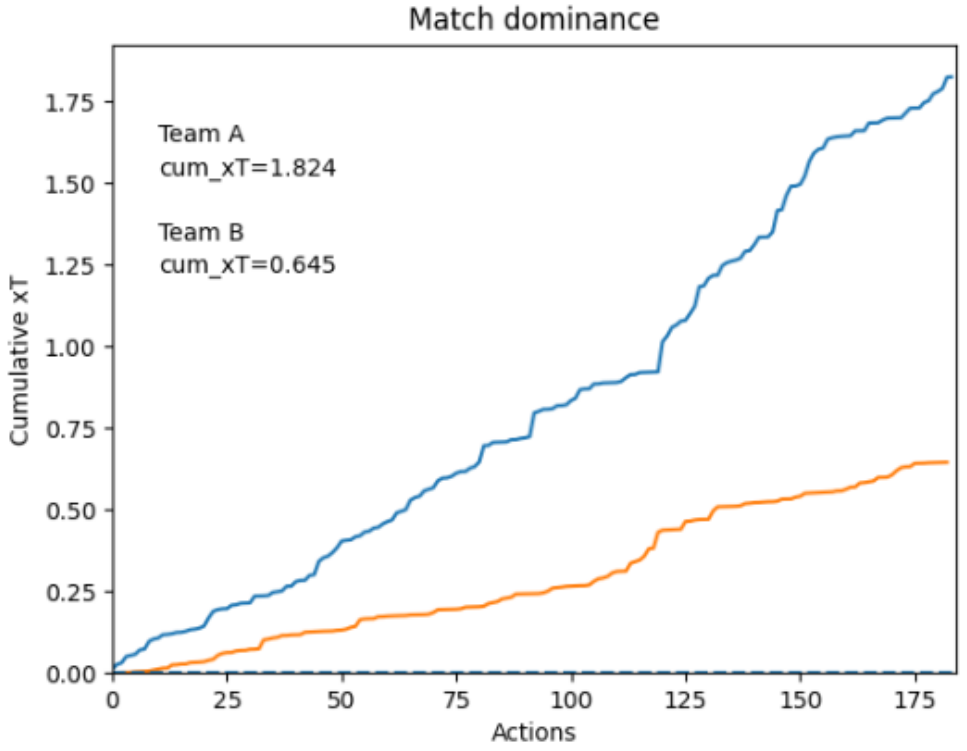
The function *act_dur* (see A.26) is built to obtain a vector counting the number of events for each action; the first for cycle includes if conditions to discriminate between cases of same team in consecutive events in which the action continues and those of different teams in which a new action is initialized.
The other outputs are the average length of actions *mean_action_A* and *mean_action_B*, so a split of the *count_action* vector is needed.

A not very significant result is shown in the following chart, similarly to the first attempt in paragraph 4.2.1.



A solution is again to aggregate information: it is possible to create a cumulative vector of expected threat for each team to bring out measure like the expected threat created during the entire match or even just during certain situations.
The piece of code A.27 can be added to the previous function *act_dur*, while the charts are two different ways of representation of these statistics:





4.3 Player dominance

Analyzing the match it could be interesting to go into more detail about the contribution of each player, not just stopping at team level.

The initial idea is to compute the expected threat produced by the player similar to how it has been done for actions: when a player is the protagonist of an event is added the expected threat of the field area where the event takes place.
The implementation consists of a function *exp_thr_player* (see A.28) where by swiping the *player* vector, which includes the player protagonist of every event in the match, three lists are created for the players themselves and their relative expected threat and team.

The function *role_extraction* is built to show results through a chart and not to use the player ID to identify him, assigning roles followed by sequence numbers according to them position in the *player_list* vector (see A.29).
The piece of code A.30 is written to create the following bar chart:

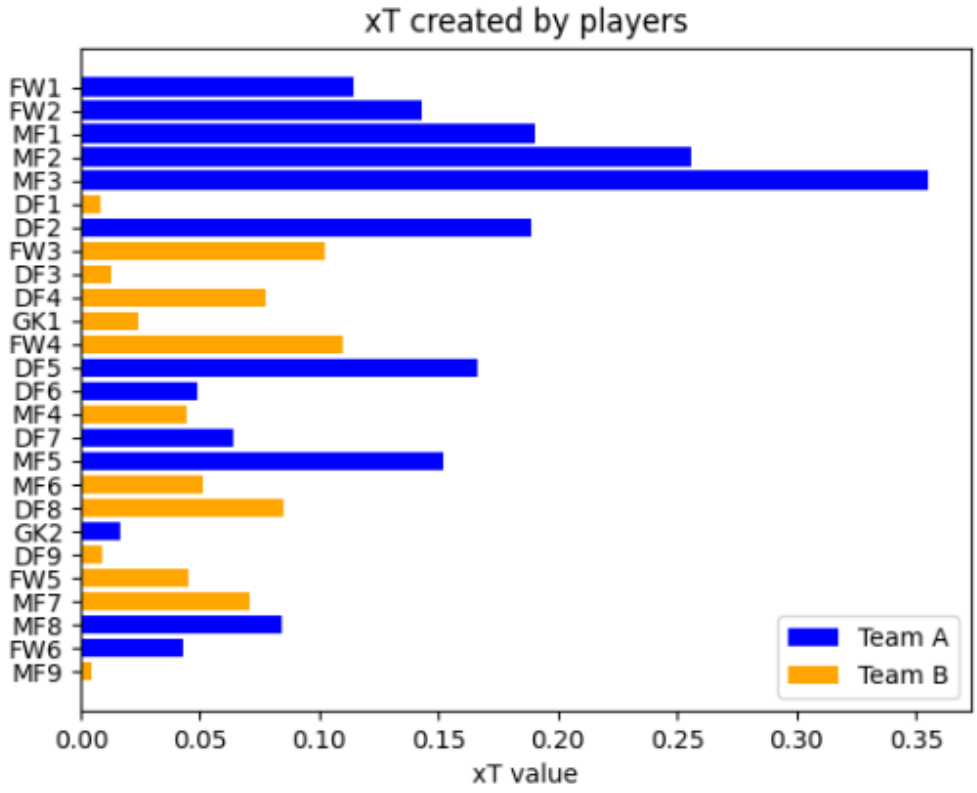


Figure 4.4: The third midfielder is the player with the highest expected threat during the match.

4.3.1 Total tournament edition

The logical continuation of the analysis is to evaluate the performance of players not just in a match, but during all the competition.
In the piece of code A.31 it is necessary to separate teams in the match because the analysis on the tournament edition is done team by team.
Then the external for cycle is iterated on teams (or rather a dictionary containing their names and codes), while in the for cycle on i the function *exp_thr_player* is applied to lists *same_team* and *same_team_player*.
In the for cycle on k partial results are added to finally obtain the vector xT_player, containing the expected threat of players of a team during the competition.

The results are again shown through the bar chart A.32, where there is no

longer the distinction between teams because this analysis is distinct for each of them.

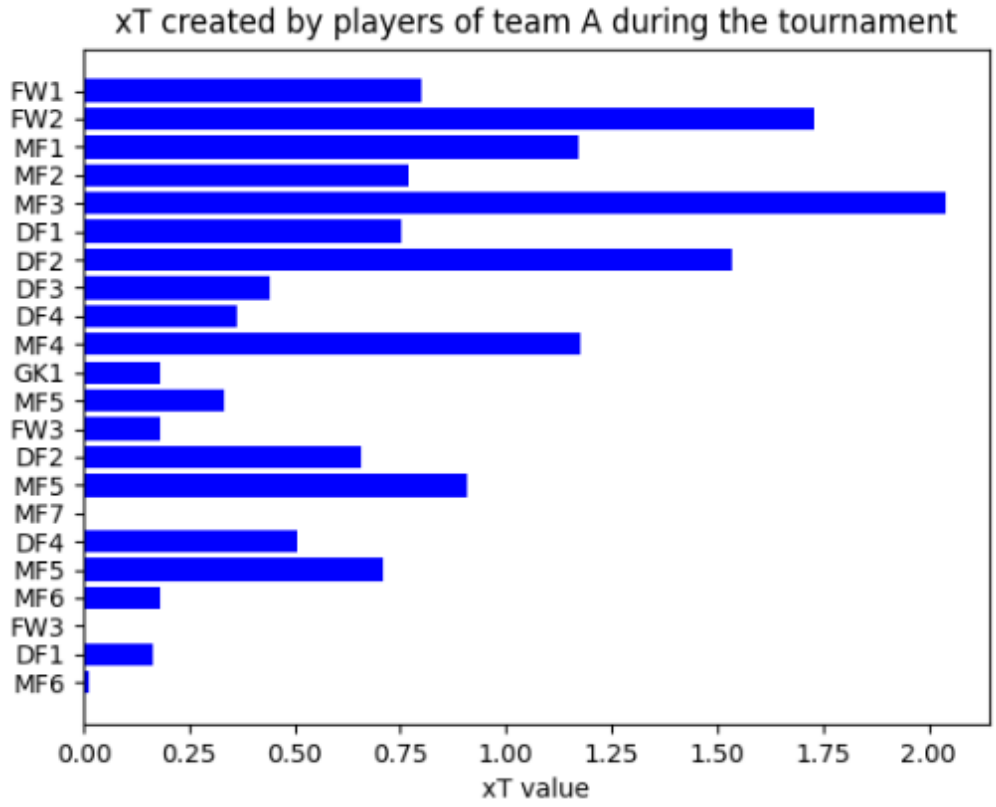


Figure 4.5: The player of team A with the highest expected threat during the tournament is again the third midfielder.

4.3.2 Normalization for minutes played

An additional factor that can be considered is how much a player was present on the field, because him absolute impact on the tournament depends also on the number of match played, which until now has not been considered.
This information is extracted in the framework A.33, checking whether or not the player belongs to the list of those substituted and entered.
The result is a vector *min_on_field*, which at the end of the for cycle is used to normalize the vector *xT_player*.
The outcome would have unit of measurement equal to xT per minutes (xTpmin), while often the statistics are reported on the regulatory duration of a match, hence the final multiplication for 90 minutes obtaining the xT per match or xTp90min.

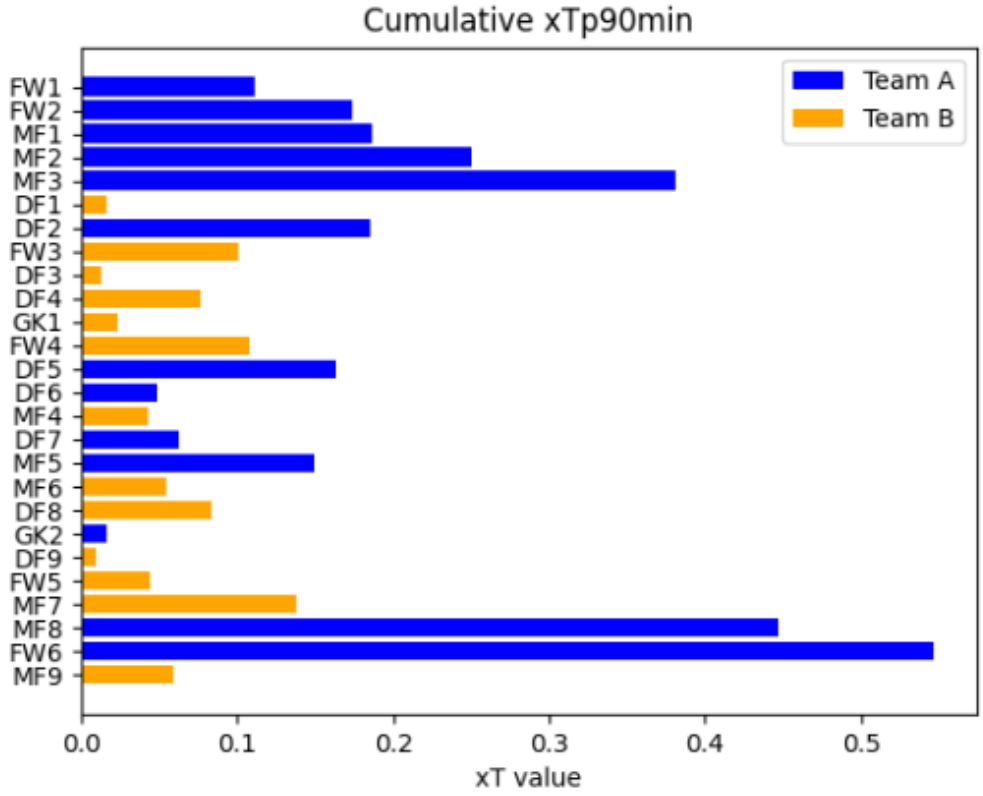


Figure 4.6: It is interesting to see how this normalization gives importance to the new entries of team A, that probably have had a good impact in the last minutes of the match.
However, they enter when other players are more tired so there might be a little bias.

This analysis can be extended to the entire competition similarly to before.
But doing the normalization on the match it becomes relevant just the impact of the new entries (so few players), while on the tournament that of players not always deployed on the field, which are many more and with more variability.

In the piece of code A.34 the structure of external for cycles come again from the previous paragraph, while internal cycles to compute vector *min_on_field* has the same structure of A.33.

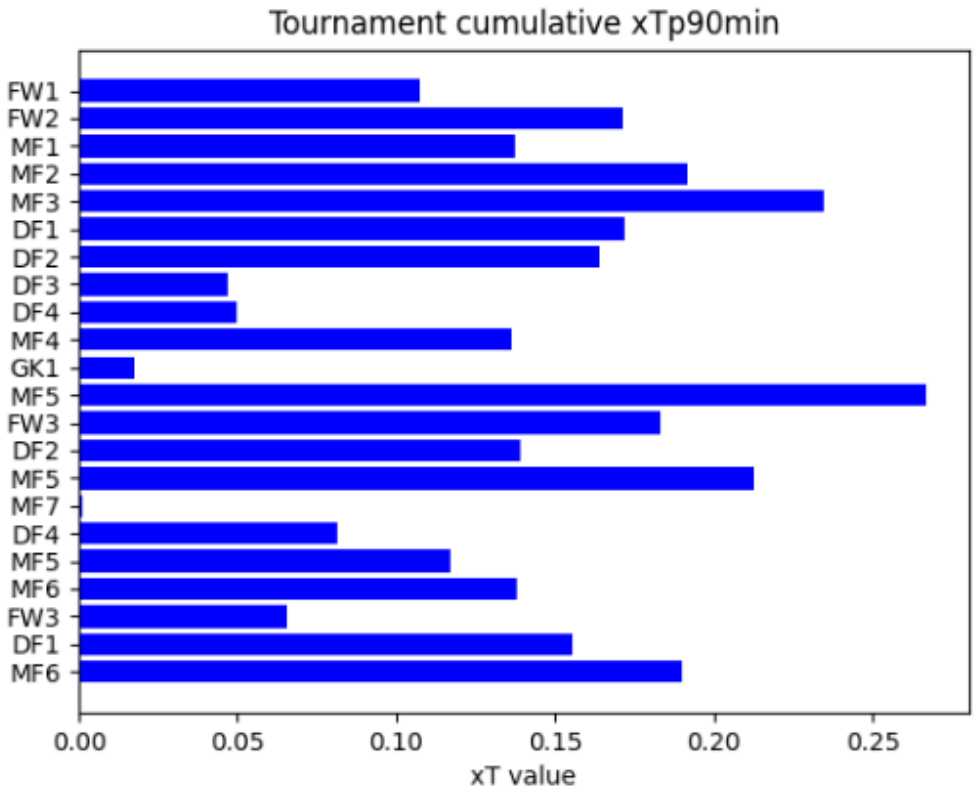


Figure 4.7: These results can be considered the most consistent until now regarding the player performances during the competition.
They confirm that midfielders, in particular MF5 and MF3, were the most threating in the team.

4.3.3 Expected threat contribution

Another interesting analysis is to evaluate a player for his contribution to the action moving the ball from an area to another, not just for the expected threat of the field area where he is located.

The function *contribution* (see A.36) has been built for this purpose; its structure is really similar to that of *exp_thr_player*, but instead of $tixT$ of the actual field area the addends are values of *diff_xT* in the actual and the consequent field area, computed with a for cycle at the beginning of the function.

If conditions on *field_zone* values are inserted to avoid bias in results because when there is a goal the expected threat becomes near to 0 (almost impossible to score another goal immediately after having made one).
The strategy adopted is to break the process at the ball touch before the goal until there is a new possession, so couples [*field area before goal, goal*] and [*goal, lost*] are skipped.

There are other conditions on the value of vector *team*, in particular on the actual, previous and next elements, to make possible, for a player of team A:

- if the possession is gained, the addition of the expected threat in the previous position of team B
- if the possession is lost, the subtraction of the expected threat in the next position of the team B

The bar chart has a similar structure to the figure 4.4 but now values can be negative.

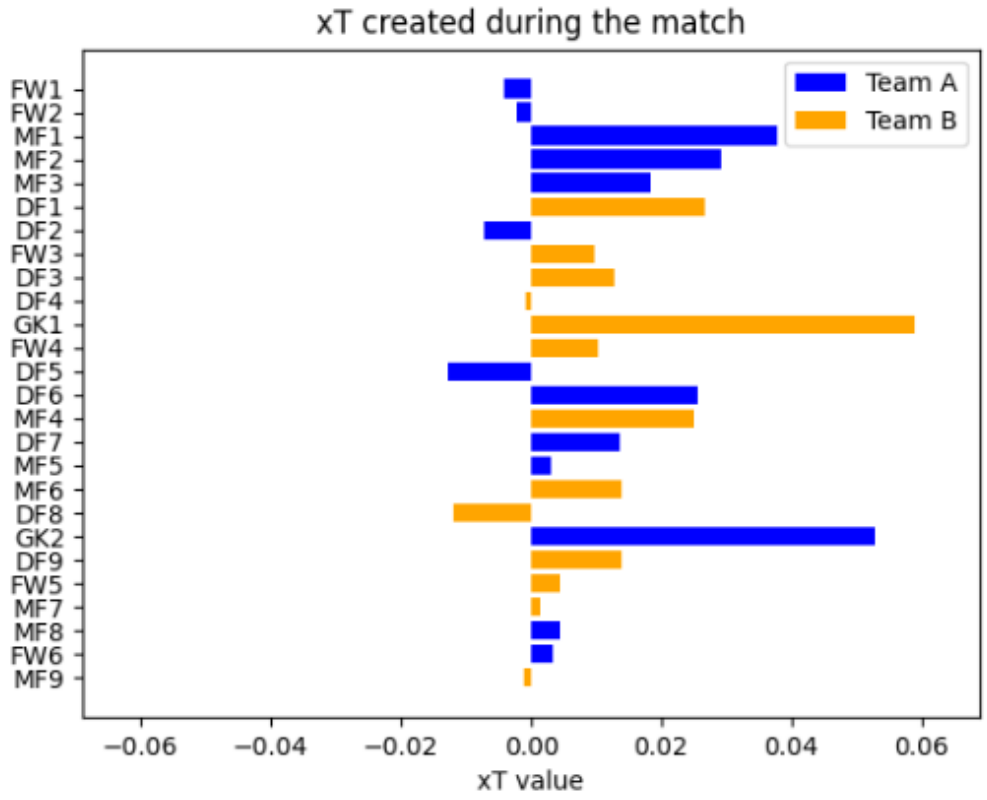


Figure 4.8: With respect to the previous approach, goalkeepers and defenders acquire great importance in the contribution to the match.

Bibliography

[CD74] Hinkley D.V. Cox D.R. *Theoretical Statistics*, pages 49,209. Chapman Hall, 1974.

[Dek05] Cornelis; Lopuhaä Hendrik Paul; Meester Ludolf Erwin Dekking, Frederik Michel; Kraaikamp. *A Modern Introduction to Probability and Statistics*. Springer Texts in Statistics, 2005.

[Eng17] Oxford Dictionaries | English. Markov chain | definition of markov chain in us english by oxford dictionaries, 2017.

[Gag17] Paul A. Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*, pages 1–235. John Wiley Sons, 2017.

[Goo65] L.A. Goodman. *On simultaneous confidence intervals for multinomial proportions*, volume 7, pages 247–254. Technometrics, 2 edition, 1965.

[Jon05] Matthew T. Jones. Estimating markov transition matrices using proportions data: An application to credit risk. *IMF Working Paper*, 219, 2005.

[Moo74] Franklin A.; Boes Duane C. Mood, Alexander; Graybill. *Introduction to the Theory of Statistics*, page 241–246. McGraw-Hill, 1974.

[Ney37] J. Neyman. Outline of a theory of statistical estimation based on the classical theory of probability. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 236:333–380, 1937.

[Nor97] J. R. Norris. *Markov chains*, pages 108–127. Cambridge University Press, 1997.

[Pea00] Karl Pearson. Introduction to the theory of statistics. *Philosophical Magazine*, Series 5. 50 (302):157–175, 1900.

[Per13] Josef Perktold. Tests and confidence intervals for binomial proportions. `https://www.statsmodels.org/dev/\_modules/statsmodels/stats/proportion.html#multinomial\_proportions\_confint`, 2013. Created on 1 March 2013.

[Que64] Hurst D.C. Quesenberry, C.P. *Large-sample simultaneous confidence intervals for multinomial proportions*, volume 6, pages 191–195. Technometrics, 1964.

[Sin19] Karun Singh. Introducing expected threat. `https://karun.in/blog/expected-threat.html`, 2019.

[Wika] Wikipedia. Binomial test. `https://en.wikipedia.org/wiki/Binomial_test`.

[Wikb] Wikipedia. Chi-squared distribution. `https://en.wikipedia.org/wiki/Chi-squared_distribution#cite_note-Lancaster1969-8`.

[Wikc] Wikipedia. Chi-squared test. `https://en.wikipedia.org/wiki/Chi-squared_test`.

[Wikd] Wikipedia. Confidence interval. `https://en.wikipedia.org/wiki/Confidence_interval`.

[Wike] Wikipedia. Markov chain. `https://en.wikipedia.org/wiki/Markov_chain`.

Appendix A

Code

A.1 Event extraction

```
marks = pd.read_json(json_file)

balltouches = marks[marks['Tags'].apply(lambda x:
    np.all([*map(lambda l: l in x, ['BallTouch'])]))]

goals = marks[marks['Tags'].apply(lambda x:
    np.all([*map(lambda l: l in x, ['Goal'])]))]

balltouches = pd.concat([balltouches, goals],
    join='inner', ignore_index=True)
```

A.2 Event validation

```
balltouches['is_Position_nan'] = balltouches['X'].apply(lambda x:
    1 if not np.isnan(x) else 0)

balltouches['is_Subjects_nan'] = balltouches['Subjects'].apply
    (lambda x: 1 if collections.Counter(['Type', 'Verb',
    'SubjectID']) == collections.Counter(list(pd.DataFrame
    ([l for l in x]).columns)) else 0)

balltouches = balltouches[balltouches['is_Subjects_nan'] == 1]
balltouches = balltouches[balltouches['is_Position_nan'] == 1]
```

A.3 Function *is_goal*

```
def is_goal(x):
    if 'Goal' in x['Tags']:
        return True
    else:
```

```
        return False
```

A.4 Function *is_consequential*

```
def is_consequential(x):
    if not np.isnan(x['next_index']):
        if int(x['next_index']) - int(x['index_col']) > 1:
            return False
        else:
            return True
    else:
        return True
```

A.5 States assignment

```
states = []
for i, row in team_A.iterrows():
    if row['is_goal']:
        states.append('goal')
        states.append('lost')
    elif not row['is_ball_in_possession']:
        states.append('lost')
    else:
        states.append([row['X'], row['Y']])
```

A.6 Function *find_states*

```
def find_states(json_file, team):
    """
    States assignment to events
    :param json_file: marks file containing all match events
    :param team: ID of the team to which the states are assigned
    :return: list containing the state for each event and numpy
        vector containing the ID of the left team for each event
    """

    marks = pd.read_json(json_file)

    [...]

    team_A = balltouches[balltouches['Team'] == team]

    [...]

    left_team = team_A['LeftTeam']
    states = []
    for i, row in team_A.iterrows():
        if row['is_goal']:
```

```
            states.append('goal')
            states.append('lost')
        elif not row['is_ball_in_possession']:
            states.append('lost')
        else:
            states.append([row['X'], row['Y']])

    return states, left_team
```

A.7 If condition

```
    if team == int(left_team[i]):

    [...]

    else:
```

A.8 Function *field_ statistics*

```
def field_statistics(m, n, states, team, left_team):
    """
    Statistics about field areas
    :param m: areas of the long side of the field
    :param n: areas of the short side of the field
    :param states: list with position, lost and goals of match
    team events
    :param team: integer with the ID of the team
    :param left_team: integer with the ID of the first team
    playing from left to right
    :return: numpy vectors with the field area for each event
    and with the count of events for each field area (0 when
    the field is not identified, probably due to not available
    coordinates)
    """
    # subdivision of the field in m parts on the x, n parts on the y
    x = 2 / m
    y = 2 / n
    eps = 0.2
    # assignment of each ball touch to a field area
    field_zone = np.zeros(len(states))
    count = np.zeros(m * n + 2)
    for i in left_team.index:
        if states[i] == 'goal':
            field_zone[i] = m * n + 1
            count[m * n + 1] += 1
        elif states[i] == 'lost':
            field_zone[i] = m * n
            count[m * n] += 1
        else:
            if team == int(left_team[i]):
                for j in range(m):
```

```
                    for h in range(n):
                        if -1 + j * x < states[i][0] <= -1 +
                            (j + 1) * x and -1 + h * y <
                            states[i][1] <= -1 + (h + 1) * y:

                            field_zone[i] = j * n + h
                            count[j * n + h] += 1
                        if j == 0:
                            if -1 - eps < states[i][0] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif j == m - 1:
                            if 1 <= states[i][0] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        if h == 0:
                            if -1 - eps < states[i][1] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif h == n - 1:
                            if 1 <= states[i][1] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
            else:
                for j in range(m):
                    for h in range(n):
                        if -1 + j * x < -states[i][0] <= -1 +
                            (j + 1) * x and -1 + h * y <
                            -states[i][1] <= -1 + (h + 1) * y:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        if j == 0:
                            if -1 - eps < -states[i][0] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif j == m - 1:
                            if 1 <= -states[i][0] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        if h == 0:
                            if -1 - eps < -states[i][1] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif h == n - 1:
                            if 1 <= -states[i][1] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1

    return field_zone, count
```

A.9 Function *plot_statistics_on_field*

```
def plot_statistics_on_field(m, n, count, team):
```

```
    """
    Plot of statistics extracted by field statistics functions
    :param m: areas of the long side of the field
    :param n: areas of the short side of the field
    :param count: vector returned by field_statistics
    with count of events for each field zone
    :param team: ID of the team for the plot title
    :return: plots
    """
    # create and fill a grid to plot frequencies in
    # different positions
    grid = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            grid[i, j] = count[i * n + j]
    plt.xticks(ticks=np.arange(m))
    plt.yticks(ticks=np.arange(n))

    # blurred heatmap
    plt.imshow(grid, cmap='Blues', interpolation="spline16")
    if team is not None:
        plt.title(team)
    plt.show()
    plt.savefig('Blurred%s' % team)

    # defined square heatmap
    f = plt.figure()
    f.set_figwidth(30)
    f.set_figheight(60)

    pitch = VerticalPitch(pitch_color='grass',
        line_color='white', stripe=True)

    fig, ax = pitch.draw()

    hm = plt.imshow(grid, cmap='Blues', interpolation="nearest",
        extent=[0, 80, 0, 120])

    plt.colorbar(hm)
    centers = np.zeros((m * n, 2))
    for i in range(m):
        for j in range(n):
            centers[i * n + j, 0] = i
            centers[i * n + j, 1] = j

            plt.text(10 + 25*(n-1-j), 5 + 20*(m-1-i),
                str(i * n + j), color="red", fontsize=20)

    plt.show()
    plt.savefig('DefinedSquare%s' % team)
```

A.10 All tournament matches

```
path = os.path.dirname(r'C:[...]')
```

```
path_list = glob.glob(path + "/**/*%s*.Marks.json" % team,
recursive=True, )
    for match_path in path_list:
        [states, left_team] = find_states(match_path, team_id)

        [field_zone_v, count_v] = field_statistics(m, n, states,
            team_id, left_team)

        field_zone = [*field_zone, *field_zone_v]
        count += count_v
plot_statistics_on_field(m, n, count, 'team A')
```

A.11 Transition matrix

```
def transit_matrix(m, n, field_zone, team):
    tm = np.zeros((m*n+2, m*n+2))
    for index in range(len(field_zone)-1):
        i = int(field_zone[index])
        j = int(field_zone[index+1])
        tm[i, j] += 1
    pn_transition = pd.DataFrame(tm[:, :])

    coordinates = pd.ExcelWriter('Transition matrix %s.xlsx'
        % team, engine='xlsxwriter')

    pn_transition.to_excel(coordinates, 'Transition matrix %s'
        % team, startcol=0, startrow=0)

    coordinates.close()

    return tm
```

A.12 Function *prob_ origin_ matrix*

```
def prob_origin_matrix(transition_matrix, team):
    m = transition_matrix.shape[0]
    n = transition_matrix.shape[1]
    ptm = np.zeros((m, n))
    s = np.sum(transition_matrix, axis=1)
    for i in range(m):
        for j in range(n):
            if s[i] != 0:
                ptm[i, j] = transition_matrix[i, j] / s[i]
            else:
                ptm[i, j] = 0
    pn_transition = pd.DataFrame(ptm[:, :])

    coordinates = pd.ExcelWriter('ProbOrigMatrix%s.xlsx'
        % team, engine='xlsxwriter')

    pn_transition.to_excel(coordinates, 'ProbOrigMatrix%s'
```

```
        % team, startcol=0, startrow=0)

    coordinates.close()

    return ptm
```

A.13 Total tournament edition

```
team_list = [...]
team_code = [...]
team_dict = dict(zip(team_list, team_code))
field_zone = []
count = np.zeros(m*n+2)
path = os.path.dirname(r'C:[...]')

for team, team_id in team_dict.items():

    path_list = glob.glob(path + "/**/*%s*.Marks.json" % team,
        recursive=True, )

    for match_path in path_list:
        [states, left_team] = find_states(match_path, team_id)

        [field_zone_v, count_v] = field_statistics(m, n,
            states, team_id, left_team)

        field_zone = [*field_zone, *field_zone_v]
        count += count_v
matrix_team = transit_matrix(m, n, field_zone, 'Tournament')
prob_team = prob_origin_matrix(matrix_team, 'Tournament')
plot_statistics_on_field(m, n, count, 'Tournament')
```

A.14 Stationary distribution

```
import_transition_matrix = pd.read_excel('C:[...]')
transition_matrix = import_transition_matrix.iloc[0:, 0:]
dim = np.shape(transition_matrix)[1]
# limit method
iteration_matrix = np.linalg.matrix_power(transition_matrix, k)
init_distr = np.repeat(1/dim, dim)
stationary_limit = np.matmul(iteration_matrix, init_distr)
# analytic method
S, U = eig(transition_matrix)

stationary = np.array(U[:, np.where(np.abs(S - 1.) <
    1e-8)[0][0]].flat)

stationary = np.array(stationary / np.sum(stationary))
stationary_limit = stationary.real
# empiric method
count = pd.read_excel('C:[...]')
```

```
count = pd.DataFrame.to_numpy(count.iloc[1:, 1:])
stationary_count = count / np.sum(count)
```

A.15 Confidence intervals

```
def multinomial_proportions_confint(counts, alpha=0.05,
    method='goodman'):

    if alpha <= 0 or alpha >= 1:

        raise ValueError('alpha must be in (0,1),
            bounds excluded')

    counts = np.array(counts, dtype=float)
    if (counts < 0).any():
        raise ValueError('counts must be >= 0')
    n = counts.sum()
    k = len(counts)
    proportions = counts / n
    chi2 = stats.chi2.ppf(1 - alpha / k, 1)

    delta = chi2 ** 2 + (4 * n * proportions * chi2 * (1 -
        proportions))

    region = ((2 * n * proportions + chi2 + np.array([-
        np.sqrt(delta), np.sqrt(delta)]))/(2*(chi2 +
        n))).T

    return region
```

A.16 Matrix of confidence intervals amplitude

```
import_transition_matrix = pd.read_csv('C:[...]', header=None)
dim = np.shape(import_transition_matrix)[1]
transition_matrix = import_transition_matrix.to_numpy()

prob_trans_matrix = prob_origin_matrix(transition_matrix,
    'Tournament')

region = np.zeros((dim, dim, 2))
for i in range(dim):

    region_v = multinomial_proportions_confint
        (transition_matrix[i, :])

    region[i, :, :] = region_v

amplitude_confint_matrix = np.zeros((dim, dim))
for i in range(dim):
    for j in range(dim):
```

```
        amplitude_confint_matrix[i, j] = region[i, j, 1] -
            region[i, j, 0]
```

A.17 Matrix of relative amplitudes

```
rel_ampl_matrix = np.zeros((dim, dim))
max_elem = 0
for i in range(dim):
    for j in range(dim):
        if transition_matrix[i, j] > 5:

            rel_ampl_matrix[i, j] = amplitude_confint_matrix[i, j]/
                prob_trans_matrix[i, j]

            if rel_ampl_matrix[i, j] > max_elem:
                max_elem = rel_ampl_matrix[i, j]

for i in range(dim):
    for j in range(dim):
        if rel_ampl_matrix[i, j] == 0:
            rel_ampl_matrix[i, j] = 2 * max_elem
```

A.18 Absolute and relative confidence intervals amplitude

```
dim = len(count)
region_stat = multinomial_proportions_confint(count)

amplitude_confint_count = np.zeros(dim)
for i in range(dim):

    amplitude_confint_count[i] = region_stat[0, i, 1]-
        region_stat[0, i, 0]

plot_statistics_on_field(6, 3, amplitude_confint_count,
    'StationaryDistribution')

rel_ampl_confint_count = np.zeros(dim)
for i in range(dim):

    rel_ampl_confint_count[i] = amplitude_confint_count[i]/
        stationary_count[i]

plot_statistics_on_field(6, 3, rel_ampl_confint_count,
    'StationaryDistribution')
```

A.19 Function *exp_ thr*

```
def exp_thr(n):
    import_transition_matrix = pd.read_csv('C:[...]')

    prob_trans_matrix = prob_origin_matrix(pd.DataFrame.to_numpy
        (import_transition_matrix), 'Tournament')

    import_count = pd.read_excel('C:[...]')
    count = import_count.iloc[0:, 1:]
    dim = int(np.shape(transition_matrix)[1])
    xT = np.zeros((dim, n))
    np_transition_matrix = prob_trans_matrix
    xT[:, 0] = np_transition_matrix[:, -1]
    for i in range(n-1):
        xT[:, i+1] = np.dot(np_transition_matrix, xT[:, i])
    exp_thr_matrix = pd.DataFrame(xT[:, :])

    coordinates = pd.ExcelWriter('ExpectedThreat%sIterations.xlsx'
        % n, engine='xlsxwriter')

    exp_thr_matrix.to_excel(coordinates, 'ExpectedThreat%sIterations'
        % n, startcol=0, startrow=0)

    coordinates.close()
    return xT, count
```

A.20 Function *find_states_match*

```
def find_states_match(json_file):

    marks = pd.read_json(json_file)

    [...]

    states = []
    team = []
    left_team = []
    for i, row in balltouches.iterrows():
        if row['is_goal']:
            states.append('goal')
            team.append(row['Team'])
            left_team.append(row['LeftTeam'])
        else:
            states.append([row['X'], row['Y']])
            team.append(row['Team'])
            left_team.append(row['LeftTeam'])

    return states, team, left_team
```

A.21 If condition (event by event)

```
    if int(team[i]) == int(left_team[i]):

    [...]

    else:
```

A.22 Function *field_statistics_match*

```
def field_statistics_match(m, n, states, team, left_team):
    """
    Statistics about field zones
    :param m: subdivisions of the long side of the field
    :param n: subdivisions of the short side of the field
    :param states: list with states
    :param team: list with team
    :param left_team: list with left team
    :return: numpy vectors with the field area for each
    event and with the count of events for each field area
    """
    # subdivision of the field in m parts on the x,
    n parts on the y
    x = 2 / m
    y = 2 / n
    eps = 0.2
    len_states = len(states)

    # assignment of each ball touch to a field area
    field_zone = np.zeros(len_states)
    count = np.zeros(m * n + 2)
    for i in range(len_states):
        if states[i] == 'goal':
            field_zone[i] = m * n + 1
            count[m * n + 1] += 1
        else:
            if int(team[i]) == int(left_team[i]):
                for j in range(m):
                    for h in range(n):

                        if -1 + j * x < states[i][0] <= -1 +
                            (j + 1) * x and -1 + h * y <
                            states[i][1] <= -1 + (h + 1) * y:

                            field_zone[i] = j * n + h
                            count[j * n + h] += 1
                        if j == 0:
                            if -1 - eps < states[i][0] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif j == m - 1:
                            if 1 <= states[i][0] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
```

```
                        if h == 0:
                            if -1 - eps < states[i][1] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif h == n - 1:
                            if 1 <= states[i][1] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
            else:
                for j in range(m):
                    for h in range(n):

                        if -1 + j * x < -states[i][0] <= -1 +
                            (j + 1) * x and -1 + h * y <
                            -states[i][1] <= -1 + (h + 1) * y:

                            field_zone[i] = j * n + h
                            count[j * n + h] += 1
                        if j == 0:
                            if -1 - eps < -states[i][0] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif j == m - 1:
                            if 1 <= -states[i][0] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        if h == 0:
                            if -1 - eps < -states[i][1] <= -1:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1
                        elif h == n - 1:
                            if 1 <= -states[i][1] < 1 + eps:
                                field_zone[i] = j * n + h
                                count[j * n + h] += 1

    return field_zone
```

A.23 xT computing: initial idea

```
# initial idea
xT = np.zeros(len(field_zone))
for i in range(1, len(field_zone)):
    if team[i] == team[0]:
        xT[i] = tixT[int(field_zone[i])]
    else:
        xT[i] = -tixT[int(field_zone[i])]
```

A.24 xT computing: weighted sum idea

```
xT = np.zeros(len(field_zone))
for i in range(1, 3):
```

```
    if team[i] == team[0]:
        xT[i] = tixT[int(field_zone[i])]
    else:
        xT[i] = -tixT[int(field_zone[i])]
for i in range(3, len(field_zone)):
    if team[i] == team[0]:

        xT[i] = tixT[int(field_zone[i])]*8/15 +
            xT[i-1]*4/15 + xT[i-2]*2/15 + xT[i-3]*1/15

    else:

        xT[i] = -tixT[int(field_zone[i])]*8/15 +
            xT[i-1]*4/15 + xT[i-2]*2/15 + xT[i-3]*1/15
```

A.25 xT computing: time segments idea

```
max_time = max(time_stamp)
minute_range = k
millisec_range = minute_range*60*1000  # 1000 because time_stamp
                                       # is measured in milliseconds
time_segments = int(max_time//millisec_range)

xT = np.zeros(time_segments+1)
for i in range(1, time_segments+1):
    for j in range(len(time_stamp)):
        if i*millisec_range <= time_stamp[j] <= (i+1)*millisec_range:
            if team[j] == team[0]:
                xT[i] = xT[i] + tixT[int(field_zone[j])]
            else:
                xT[i] = xT[i] - tixT[int(field_zone[j])]
```

A.26 Function *act_ dur*

```
def act_dur(vect_team, field_zone, tixT):
    count_action = []
    xT_action = []
    j = 0
    for i in range(len(vect_team)):
        if i == 0:
            count_action.append(0)
            count_action[j] += 1
            xT_action.append(0)
            xT_action[j] += tixT[int(field_zone[i])]

        elif vect_team[i-1] == vect_team[0] and vect_team[i]
            == vect_team[0]:

            count_action[j] += 1
            xT_action[j] += tixT[int(field_zone[i])]
```

```
        elif vect_team[i-1] != vect_team[0] and vect_team[i]
            != vect_team[0]:

            count_action[j] += 1
            xT_action[j] -= tixT[int(field_zone[i])]

        elif vect_team[i-1] == vect_team[0] and vect_team[i]
            != vect_team[0]:

            count_action.append(0)
            xT_action.append(0)
            j += 1
            count_action[j] += 1
            xT_action[j] -= tixT[int(field_zone[i])]

        elif vect_team[i-1] != vect_team[0] and vect_team[i]
        == vect_team[0]:

            count_action.append(0)
            xT_action.append(0)
            j += 1
            count_action[j] += 1
            xT_action[j] += tixT[int(field_zone[i])]

    team_A_action = []
    team_B_action = []
    for i in range(len(np_count_action)):
        if vect_team[i] == team[0]:
            team_A_action.append(np_count_action[i])
        else:
            team_B_action.append(np_count_action[i])
    mean_action_A = np.mean(team_A_action)
    mean_action_B = np.mean(team_B_action)

    return np_count_action, np_xT_action, mean_action_A,
        mean_action_B
```

A.27 Cumulative xT

```
    cum_xT_action_A = []
    cum_xT_action_B = []
    j = 0
    k = 0
    for i in range(len(np_xT_action)):
        if np_xT_action[i] > 0:
            if i == 0 or i == 1:
                cum_xT_action_A.append(np_xT_action[i])
                j += 1
            elif i > 1:

                cum_xT_action_A.append(np_xT_action[i] +
                cum_xT_action_A[j-1])

                j += 1
```

```
        elif np_xT_action[i] < 0:
            if i == 0 or i == 1:
                cum_xT_action_B.append(np_xT_action[i])
                k += 1
            elif i > 1:

                cum_xT_action_B.append(np_xT_action[i] +
                cum_xT_action_B[k-1])

                k += 1

    return np_count_action, np_xT_action, mean_action_A,
        mean_action_B, cum_xT_action_A, cum_xT_action_B
```

A.28 Expected threat player

```
def exp_thr_player(team, player, field_zone, tixT):
    player_list = []
    xT_player = []
    team_list = []
    for i in range(len(player)):
        exist_count = player_list.count(player[i])
        if exist_count > 0:
            j = player_list.index(player[i])
            xT_player[j] += tixT[int(field_zone[i])]
        else:
            player_list.append(player[i])
            team_list.append(team[i])
            xT_player.append(0)
            xT_player[-1] += tixT[int(field_zone[i])]
    return xT_player, player_list, team_list
```

A.29 Role extraction

```
def role_extraction(player_list, json_file):
    lineups = pd.read_json(json_file)
    teams = lineups['Teams']
    players_A = teams[0]['Players']
    players_B = teams[1]['Players']
    players_id_A = []
    role_A = []
    for i in range(len(players_A)):
        players_id_A.append(int(players_A[i]['PlayerID']))
        role_A.append(players_A[i]['Role'])
    players_id_B = []
    role_B = []
    for i in range(len(players_B)):
        players_id_B.append(int(players_B[i]['PlayerID']))
        role_B.append(players_B[i]['Role'])
    players_id = players_id_A + players_id_B
    role = role_A + role_B
```

```
    role_list = []
    count_g = 1
    count_d = 1
    count_m = 1
    count_f = 1
    for i in range(len(player_list)):
        j = players_id.index(player_list[i])
        if role[j] == 'Goalkeeper':
            role_list.append(str('GK')+str(count_g))
            count_g += 1
        if role[j] == 'Defender':
            role_list.append(str('DF')+str(count_d))
            count_d += 1
        if role[j] == 'Midfielder':
            role_list.append(str('MF')+str(count_m))
            count_m += 1
        if role[j] == 'Forward':
            role_list.append(str('FW')+str(count_f))
            count_f += 1

    return players_id, role, role_list
```

A.30 Bar chart match

```
fig, ax = plt.subplots()
y_pos = np.arange(len(xT_player))
ax.barh(index_A, xT_player_A, align='center', color='b')
ax.barh(index_B, xT_player_B, align='center', color='orange')
ax.set_yticks(index_A + index_B, labels=role_list_A + role_list_B)
ax.invert_yaxis()
ax.set_xlabel('xT value')
ax.set_title('xT created by players')
ax.legend(['Team A', 'Team B'], loc='lower right')
plt.show()
```

A.31 Expected threat player tournament

```
for team, team_id in team_dict.items():

    path_list_marks = glob.glob(path + "/**/*%s*.Marks.json"
        % team, recursive=True, )

    path_list_phases = glob.glob(path + "/**/*%s*.Phases.json"
        % team, recursive=True, )

    path_list_lineups = glob.glob(path + "/**/*%s*.Lineups.json"
        % team, recursive=True, )

    for i in range(len(path_list_marks)):
        if i == 0:
```

```
        [states, team, player, left_team, time_stamp,
            first_enj, second_enj] = find_states_match
            (path_list_marks[i], path_list_phases[i])

        field_zone = field_statistics_match(m, n, states,
            team, left_team)

        same_team = []
        same_team_player = []
        same_team_field_zone = []
        for j in range(len(team)):
            if team[j] == team_id:
                same_team.append(team[j])
                same_team_player.append(player[j])
                same_team_field_zone.append(field_zone[j])

        [xT_player, player_list, team_list] =
            exp_thr_player(same_team, same_team_player,
            same_team_field_zone, tixT)

        [player_id, role, role_list] = role_extraction
            (player_list, path_list_lineups[i])

    if i > 0:
        [states, team, player, left_team, time_stamp,
            first_enj, second_enj] = find_states_match
           (path_list_marks[i], path_list_phases[i])

        field_zone = field_statistics_match(m, n, states,
            team, left_team)

        same_team = []
        same_team_player = []
        same_team_field_zone = []
        for j in range(len(team)):
            if team[j] == team_id:
                same_team.append(team[j])
                same_team_player.append(player[j])
                same_team_field_zone.append(field_zone[j])

        [xT_player_i, player_list_i, team_list_i] =
            exp_thr_player(same_team, same_team_player,
            same_team_field_zone, tixT)

        [player_id_i, role_i, role_list_i] =
            role_extraction(player_list_i,
            path_list_lineups[i])

        for k in range(len(player_list_i)):
            exist_count = player_list.count(player_list_i[k])
            if exist_count > 0:
                z = player_list.index(player_list_i[k])
                xT_player[z] += xT_player_i[k]
            else:
                player_list.append(player_list_i[k])
                team_list.append(team_list_i[k])
```

```
                    xT_player.append(0)
                    xT_player[-1] += xT_player_i[k]
```

A.32 Bar chart tournament

```
fig, ax = plt.subplots()
y_pos = np.arange(len(xT_player))
ax.barh(y_pos, xT_player, align='center', color='b')
ax.set_yticks(y_pos, labels=role_list)
ax.invert_yaxis()
ax.set_xlabel('xT value')
ax.set_title('xT created by players during the tournament')
plt.show()
```

A.33 Normalization by minutes on the match

```
match_dur = 90+int(first_enj)+int(second_enj)
min_on_field = np.zeros(len(player_list))
for i in range(len(player_list)):

    if not np.any(player_left == player_list[i]) and not
        np.any(player_entered == player_list[i]):

        min_on_field[i] = match_dur

    elif np.any(player_left == player_list[i]) and not
        np.any(player_entered == player_list[i]):

        j = np.where(player_left == player_list[i])
        min_on_field[i] = int(time_stamp_subst[j[1]])//(1000*60)

    elif not np.any(player_left == player_list[i]) and
        np.any(player_entered == player_list[i]):

        j = np.where(player_entered == player_list[i])

        min_on_field[i] = (match_dur*1000*60 -
            int(time_stamp_subst[j[1]]))//(1000*60)

    elif np.any(player_left == player_list[i]) and
        np.any(player_entered == player_list[i]):

        j = np.where(player_entered == player_list[i])
        k = np.where(player_left == player_list[i])

        min_on_field[i] = (int(time_stamp_subst[k[1]]) -
            int(time_stamp_subst[j[1]]))//(1000*60)

xT_player_norm_by_min = np.divide(xT_player, min_on_field)
```

A.34 Normalization by minutes on the tournament

```
for team, team_id in team_dict.items():

    path_list_marks = glob.glob(path + "/**/*%s*.Marks.json"
        % team, recursive=True, )

    path_list_phases = glob.glob(path + "/**/*%s*.Phases.json"
        % team, recursive=True, )

    path_list_lineups = glob.glob(path + "/**/*%s*.Lineups.json"
        % team, recursive=True, )

    for i in range(len(path_list_marks)):
        if i == 0:

            [player_left, player_entered, time_stamp_subst] =
                min_played(path_list_marks[i],
                path_list_lineups[i], path_list_phases[i])

            [states, team, player, left_team, time_stamp,
                first_enj, second_enj] = find_states_match
                (path_list_marks[i], path_list_phases[i])

            field_zone = field_statistics_match(m, n, states,
                team, left_team)

            same_team = []
            same_team_player = []
            same_team_field_zone = []
            for j in range(len(team)):
                if team[j] == team_id:
                    same_team.append(team[j])
                    same_team_player.append(player[j])
                    same_team_field_zone.append(field_zone[j])

            [xT_player, player_list, team_list] = exp_thr_player
                (same_team, same_team_player,
                same_team_field_zone, tixT)

            match_dur = 90+int(first_enj)+int(second_enj)
            min_on_field = []
            for h in range(len(player_list)):

                if not np.any(player_left == player_list[h]) and
                    not np.any(player_entered == player_list[h]):

                    min_on_field.append(match_dur)

                elif np.any(player_left == player_list[h]) and not
                    np.any(player_entered == player_list[h]):

                    j = np.where(player_left == player_list[h])
                    min_on_field.append(int(time_stamp_subst[j[1]])
                        //(1000*60))
```

```
            elif not np.any(player_left == player_list[h]) and
                np.any(player_entered == player_list[h]):

                j = np.where(player_entered == player_list[h])

                min_on_field.append((match_dur*1000*60-
                    int(time_stamp_subst[j[1]]))//(1000*60))

            elif np.any(player_left == player_list[h]) and
                np.any(player_entered == player_list[h]):

                j = np.where(player_entered == player_list[h])
                k = np.where(player_left == player_list[h])

                min_on_field.append((int(time_stamp_subst[k[1]])
                    -int(time_stamp_subst[j[1]]))//(1000*60))

        [player_id, role, role_list] = role_extraction
            (player_list, path_list_lineups[i])
    if i > 0:

        [player_left, player_entered, time_stamp_subst] =
            min_played(path_list_marks[i], path_list_lineups[i],
            path_list_phases[i])

        [states, team, player, left_team, time_stamp, first_enj,
            second_enj] = find_states_match(path_list_marks[i],
            path_list_phases[i])

        field_zone = field_statistics_match(m, n,
        states, team, left_team)

        same_team = []
        same_team_player = []
        same_team_field_zone = []
        for j in range(len(team)):
            if team[j] == team_id:
                same_team.append(team[j])
                same_team_player.append(player[j])
                same_team_field_zone.append(field_zone[j])
        [xT_player_i, player_list_i, team_list_i] =
            exp_thr_player(same_team, same_team_player,
            same_team_field_zone, tixT)
        match_dur = 90 + int(first_enj) + int(second_enj)
        min_on_field_i = []
        for h in range(len(player_list_i)):

            if not np.any(player_left == player_list_i[h]) and
                not np.any(player_entered == player_list_i[h]):

                min_on_field_i.append(match_dur)

            elif np.any(player_left == player_list_i[h]) and not
                np.any(player_entered == player_list_i[h]):
```

```
                    j = np.where(player_left == player_list_i[h])
                    min_on_field_i.append(int(time_stamp_subst[j[1]])
                        //(1000*60))

                elif not np.any(player_left == player_list_i[h]) and
                    np.any(player_entered == player_list_i[h]):

                    j = np.where(player_entered == player_list_i[h])
                    min_on_field_i.append((match_dur*1000*60-
                        int(time_stamp_subst[j[1]]))//(1000*60))

                elif np.any(player_left == player_list_i[h]) and
                    np.any(player_entered == player_list_i[h]):

                    j = np.where(player_entered == player_list_i[h])
                    k = np.where(player_left == player_list_i[h])
                    min_on_field_i.append((int(time_stamp_subst[k[1]])
                        -int(time_stamp_subst[j[1]]))//(1000*60))

            [player_id_i, role_i, role_list_i] = role_extraction
                (player_list_i, path_list_lineups[i])

            for k in range(len(player_list_i)):
                exist_count = player_list.count(player_list_i[k])
                if exist_count > 0:
                    z = player_list.index(player_list_i[k])
                    xT_player[z] += xT_player_i[k]
                    min_on_field[z] += min_on_field_i[k]
                else:
                    player_list.append(player_list_i[k])
                    team_list.append(team_list_i[k])
                    role_list.append(role_list_i[k])
                    xT_player.append(0)
                    xT_player[-1] += xT_player_i[k]
                    min_on_field.append(min_on_field_i[k])
xT_player_norm_by_min = np.divide(xT_player, min_on_field)
fig, ax = plt.subplots()
y_pos = np.arange(len(xT_player))
ax.barh(y_pos, xT_player_norm_by_min, align='center', color='b')
ax.set_yticks(y_pos, labels=role_list)
ax.invert_yaxis()
ax.set_xlabel('xT value')
ax.set_title('xT created by players of team A during the
    tournament normalized by minutes')
plt.show()
```

A.35 Function *gained_exp_thr_player*

```
def gained_exp_thr_player(team, player, field_zone, tixT):
    diff_xT = np.zeros([len(tixT), len(tixT)])
    for i in range(len(tixT)):
        for j in range(len(tixT)):
            diff_xT[i, j] = tixT[j] - tixT[i]
```

```
    player_list = []
    xT_player = []
    team_list = []
    for i in range(len(player)-1):
        exist_count = player_list.count(player[i])
        if exist_count > 0:
            j = player_list.index(player[i])

            if field_zone[i] != m*n+1 and
                field_zone[i+1] != m*n+1:

                xT_player[j] += diff_xT[int(field_zone[i]),
                    int(field_zone[i+1])]

        else:
            player_list.append(player[i])
            team_list.append(team[i])
            xT_player.append(0)

            if field_zone[i] != m*n+1 and
                field_zone[i+1] != m*n+1:

                xT_player[-1] += diff_xT[int(field_zone[i]),
                    int(field_zone[i+1])]

    return xT_player, player_list, team_list, diff_xT
```

A.36 Expected threat contribution

```
def contribution(team, player, field_zone, tixT):
    diff_xT = np.zeros([len(tixT), len(tixT)])
    for i in range(len(tixT)):
        for j in range(len(tixT)):
            diff_xT[i, j] = tixT[j] - tixT[i]
    player_list = []
    xT_player = []
    team_list = []
    for i in range(len(player)-1):
        exist_count = player_list.count(player[i])
        if exist_count > 0:
            j = player_list.index(player[i])
            if field_zone[i] != m*n+1 and field_zone[i+1] !=
            m*n+1:
                if team[i] == team[i+1]:
                    xT_player[j] += diff_xT[int(field_zone[i]),
                    int(field_zone[i+1])]
                else:
                    xT_player[j] += -tixT[int(field_zone[i+1])]
                if i > 0:
                    if team[i-1] != team[i]:
                        xT_player[j] += tixT[int(field_zone[i-1])]
            else:
                print('goal')
        else:
```

```
            player_list.append(player[i])
            team_list.append(team[i])
            xT_player.append(0)
            if field_zone[i] != m*n+1 and field_zone[i+1] !=
            m*n+1:
                xT_player[-1] += diff_xT[int(field_zone[i]),
                int(field_zone[i+1])]
            else:
                print('goal')
    return xT_player, player_list, team_list, diff_xT
```

Appendix B

Images

Figure B.1: Folder graph

	X	Y
0	-0,02381	-0,143824
1	-0,119238	0,035882
2	0,120952	0,014118
3	0,141143	0,046176
4	0,114476	0,339706
5	0,059429	0,443235
6	0,013333	0,827059
7	0,026667	0,812941
8	-0,021524	0,611176
9	-0,042857	0,633529
10	-0,206286	0,863824
11	-0,321524	0,862059
12	-0,281714	-0,005294
13	-0,03219	0,785882
14	-0,425333	-0,140294
15	-0,51581	0,086176
16	-0,11619	0,405882
17	-0,211048	0,369412
18	0,081905	-0,093235
19	-0,201905	0,745294
20	0,032571	-0,095882
21	-0,758476	0,646471
22	-0,772381	0,630294
23	-0,832952	-0,069118
24	-0,832381	0,168824
25	-0,875619	0,065882
26	-0,827619	1,010588
27	-0,711238	0,967353

Figure B.2: Ball coordinates

	X	Y
122	-1,03048	0,412059
123	-1,02781	0,507941
1311	-1,01029	-0,53235
117	-1,00381	0,295294
1071	-0,41752	1,070882
1169	1,010286	1,07
56	0,11219	1,057941
26	-0,82762	1,010588
1878	-0,62743	1,006176
1949	0,283619	1,005
1299	-0,82781	1,003529
369	0,561905	1,000882

Figure B.3: Out of ranges positions