Finding "Memphis"

a rating-based player comparison visualisation

Wang, Y.

Award date:
2016
Finding ”Memphis” - a rating-based player comparison visualization

Master Thesis

Yujie Wang

Supervisors:
dr.ir. Huub van de Wetering

Eindhoven, October 2016
Abstract

With the development of sensor and camera tracking systems, it becomes easier to track player events in a football match. With these match events, post-match analysis on players can be conducted to compare the performance of players.

Comparing football players is not an easy task, there are many attributes to be covered. Typical attributes are player positions and statistics. For instance, common statistics are number of goals, number of assists, etcetera. There are many tools that can compare football players by listing all the statistics side by side for all players. Imagine we want to compare 10 different statistics for over 500 players simultaneously, and we also want to find out the 10 best players. Or sometimes, the user may want to find players that are similar to a target player, for instance, Memphis Depay. Simply listing the statistics is not the solution to such problems. To solve such problems, we designed a prototype visualization tool, called PureSoccerFX.

In our tool, we give the user the flexibility to define the key attributes of interest by introducing a so called interactive weighted attribute tree. This tree allows the user to create abstraction levels from real statistics and interactively build their own rating system by manipulating the hierarchical weighted attribute tree. With this tree, each player will get a rating and the user can simply compare this rating to find top players. We also apply a modified strip treemap to visualize the rating of these players based on the weighted attribute tree, which provides the user with an overview of all players in the league. To help the user find similar players, We designed a player similarity graph that can compare player similarity and rating simultaneously. At last, we designed a player comparison view for the user to easily compare players from different attributes and make the final conclusion on who are the most wanted players.
Acknowledgements

First of all, I would like to thank my supervisor, dr.ir Huub van de Wetering for providing me the opportunity to do a football related thesis project. His insightful comments have helped me all the way through the project, I would not be able to finish this thesis project without his guidance.

I would also like to thank my thesis committee members: prof.dr.ir. Jack van Wijk and dr. Natalia Sidorova. Without their help, I would not be able to make the defense in time.

I want to thank my family and all my friends who always encourage and support me. I enjoy the moments sharing joy and happiness with you.

Last but not least, special credits to football! Football is my passion.

"Football, bloody hell."

- Sir Alex Ferguson
# Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Motivation and goal ................................................. 1
1.2 Problem formulation .............................................. 2
1.2.1 Visualize comparison ........................................ 2
1.2.2 Related questions ............................................ 3
1.3 Methodology and contribution ..................................... 4
1.4 Organization of this thesis ....................................... 4

## 2 Background and related work

2.1 Football data analysis ............................................. 5
2.1.1 Statistic ....................................................... 5
2.1.2 Rating ......................................................... 7
2.2 Football specific visualization techniques ....................... 9
2.2.1 Formation ..................................................... 9
2.2.2 Attacking sides ............................................. 10
2.2.3 Event frequency ............................................. 11
2.3 High-dimensional data visualization techniques ................. 12
2.3.1 Non-hierarchical data ....................................... 12
2.3.2 Hierarchical data ........................................... 14
2.4 Comparison specific visualization techniques .................. 17
2.4.1 Juxtaposition ............................................... 17
2.4.2 Superposition ............................................... 18
2.4.3 Explicit encoding ........................................... 18

## 3 Data processing

3.1 Data description .................................................. 19
3.1.1 Feature description ......................................... 19
3.1.2 Identify statistics from events ............................... 19
3.2 Data model ....................................................... 21
3.3 Generate statistic ................................................ 23
3.3.1 Accumulated statistic .................................... 23
3.3.2 Averaged statistic ......................................... 23
3.4 Generate single statistic rating ................................ 24
3.5 Weighted attribute tree .......................................... 25
3.5.1 Compute rating with a weighted attribute tree .......... 26
3.5.2 Generate player specific weights ........................... 27
## CONTENTS

4 System design  
  4.1 Dashboard  
  4.2 Select player  
  4.3 Interactive weighted attribute tree  
    4.3.1 Tree view - detail control  
    4.3.2 Icicle plot - weight control  
  4.4 Player performance treemap  
    4.4.1 Interaction  
    4.4.2 Color encoding  
    4.4.3 Compare players in treemap  
  4.5 Player similarity graph  
    4.5.1 Visual design of player similarity graph  
    4.5.2 Interaction with player similarity graph  
  4.6 Player detail view  
  4.7 Player comparison view  

5 Results and evaluation  
  5.1 Find best players  
  5.2 Find similar players to a target player  
  5.3 Compare Players  
    5.3.1 Compare players in player performance treemap  
    5.3.2 Compare players in player comparison view  

6 Conclusions  
  6.1 Main conclusion  
  6.2 Limitation  
  6.3 Future work  

Bibliography  

Appendix  

A Details on filter  
  A.1 Filter  
    A.1.1 Range slider  
    A.1.2 Scatter plot matrix  
    A.1.3 Position zone filter  

B Player statistics table  
  B.1 Player statistics table  

C Compute stacked bar width  

D Heatmap design  
  D.0.1 Event heatmap vs average position graph
# List of Figures

1.1 MLB team salaries [27] ................................................................. 2  
1.2 Process graph on visualizing football data ............................... 3  
2.1 Shooting statistics shown with visualization [23] ....................... 5  
2.2 Football statistics shown textually [20] .................................. 6  
2.5 Initial formation for match Manchester United vs Crystal Palace (numbers inside  the circles are player ratings, which are not related to formation) [23] ............... 10  
2.6 Average position for match Manchester United vs Crystal Palace [23] .......... 10  
2.7 Attacking sides for match Manchester United vs Crystal Palace [23] .......... 10  
2.8 Heat map example .................................................................. 11  
2.9 Color map example ................................................................. 11  
2.10 Example ball occupancy map over a match half for a team attacking left to right [2] 12  
2.11 Percentage of the population living in urban areas in the world and in different  continents [13] ................................................................. 12  
2.12 An example of a stacked bar chart .......................................... 13  
2.13 An example of a pie chart ....................................................... 13  
2.14 An example of visualizing player attributes with parallel coordinates [9] ............ 13  
2.15 An example of scatter plot matrices [21] .................................. 14  
2.16 An example of tree diagram[4] ................................................ 15  
2.17 An example of icicle plot [14] .................................................. 16  
2.18 An example of treemap[14] ..................................................... 16  
2.19 Different visualization techniques for comparison [18] .................... 17  
2.20 Comparison between two players [1] ...................................... 18  
2.21 Small multiples showing similar event chains [16] ..................... 18  
2.22 Spain 1 - 5 Netherlands (June 13, 2014; World Cup group B) [8] .............. 18  
3.1 Conceptual data model. The rectangles with text inside are the entities in the  data model. The links with arrow show the composition relation between entities.  The properties of an entity are either listed inside the rectangles or placed on the  out-going links ................................................................. 22  
3.2 Concept drawing of a weighted attribute tree. All the leaf nodes are the real stat-  istics and all other nodes belong to abstraction levels .......................... 25  
3.3 We can hide the children nodes of Attack attribute to get a new detail level of the  weighted attribute tree .......................................................... 26  
4.1 PureSoccerFX dashboard. (1) Player navigator (2) Filter (3) Player statistic table  (4) Player detail (5) Player comparison view (6) Player performance treemap (7)  Player similarity graph ........................................................... 29  
4.2 Player navigator supports browsing teams and players in Eredivisie ............... 30  
4.3 Example of searching a player .................................................. 30

Finding "Memphis" - a rating-based player comparison visualization ix
4.4 Overview of supported filters. There are five categories of different filters, namely, basic filter, statistic filter, rating filter, position filter and scatter plot matrix. 31
4.5 Player statistics table can be used to browse player statistics and select interesting players. The table can be ordered by the selected column. Here, the player statistics table is ordered by Dribble. 31
4.6 Interactive weighted attribute tree control panel. A tree view is placed on the left side for building trees and controlling detail level. An icicle plot is placed on the right side for weight adjustments. 32
4.7 The small triangle in front of each attribute node in the tree view can be used to show or hide children nodes. A color picker can be used to choose a color for the attribute. The "+" in the toggle button indicates the attribute is positive or "+" is shown for negative. A bar chart is then followed to show the relative weight of the attribute and at the end the name of the attribute is shown. 33
4.8 Bar charts showing absolute weights. Key Pass and Assist are children nodes of Passing, however, it is difficult to check whether their weights add up to the weight of Passing from this visualization. 33
4.9 Bar charts without offsets showing relative weights. Key Pass and Assist are children nodes of Passing, it is convenient to compare the weights of children nodes. 33
4.10 Bar charts with offsets showing relative weights. Key Pass and Assist are children nodes of Passing and their relative weights add up to 1. 33
4.11 Tree view and icicle plot visualizing Passing node. 34
4.12 The user dragged the border of node rectangle, the width of the rectangle changes and so does the weight. We just assume the weight of node changed with ∆w, the weights of its parent and children nodes should update as well. 34
4.13 Conceptual treemap structure. 35
4.14 Overview of treemap showing 516 players in Eredivisie. When the mouse is over a player block, a tool tip of rating details is shown. 36
4.15 The big block is a player block. The small blocks inside are attribute blocks. 37
4.16 Three different status for player blocks in a treemap. 37
4.17 Weighted attribute tree that is used to compute player rating. Here, Attack and Defense attributes are visible from player blocks in treemap. 37
4.18 Player block visualizing ratings of Attack and Defense attributes. 37
4.19 Tool tip showing rating and color encoding of each attribute. 38
4.20 It is difficult to compare player1 with player3 from player block size but if all the blocks are ordered from the highest rating to the lowest rating. And the blocks are placed from top left to bottom right. Then it is clear that player1 is better than player3. 38
4.21 Player blocks colored by Dribble attribute. The player with more dribbles will get a higher opacity value for its player block that results in a darker color. The color can be chosen by the user from a color picker. 40
4.22 Player rating vector visualized with stacked bar chart. 41
4.23 Overview of player similarity graph. 41
4.24 The players are ordered by similarity in horizontal direction and by rating in vertical direction. The player with the highest cosine similarity value and rating will be in the top left corner. 42
4.25 Player performance treemap and player similarity graph share the same status. 43
4.26 Scatter plot view of player similarity graph. The advantage of a scatter plot is that it can be used to locate clusters and order on two attributes. 44
4.27 Stacked bar chart view of player similarity graph. The advantage of a stacked bar chart is that it can show the part-to-whole relationship. 44
4.28 Indicator bar is on the right side of the player similarity graph, which can be used to select players or to find selected players. 44
4.29 Player similarity graph supports selection on players. 45
4.30 Overview of player detail. 46
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.31</td>
<td>Overview of player comparison view. Here we compare three players.</td>
<td>47</td>
</tr>
<tr>
<td>4.32</td>
<td>Illustration on how to draw a difference bar chart.</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>The weighted attribute tree we use to generate the rating. We choose 10 common statistics for the weighted attribute tree and assign an equal weight to each statistic.</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>When we apply equal weights for all statistics in tree, p11402 ranks first. Here, we show all the leaf node ratings in the treemap.</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>We can hide the rating details. Here, we only show Attack and Defense ratings in the treemap.</td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>Player detail view of p11702. Here, we apply the default equally weighted attribute tree with 10 statistics.</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>Player detail view of p11702 after generating a specific rating system.</td>
<td>51</td>
</tr>
<tr>
<td>5.6</td>
<td>Player similarity graph. The yellow line shows p11702, indicating he is on top of the graph. The numbers indicate the scales of the graph.</td>
<td>52</td>
</tr>
<tr>
<td>5.7</td>
<td>We can drag the mouse to make a selection of players in the top left corner. We try to keep the selection box as left as possible because the players more to the left are more similar to p11702.</td>
<td>53</td>
</tr>
<tr>
<td>5.8</td>
<td>After selection, 7 players are left, including p11702 (with yellow border). The similarity graph automatically scales and the numbers indicate the new scales after auto-scaling.</td>
<td>53</td>
</tr>
<tr>
<td>5.9</td>
<td>We can change the width of the stacked bar charts inside the player similarity graph to make it full scale. Now we can see all the selected players are very similar in rating structure.</td>
<td>53</td>
</tr>
<tr>
<td>5.10</td>
<td>In a player performance treemap, the size, the order and the color of player blocks can be used to compare players. The user can easily select the way to compare from a choice box.</td>
<td>54</td>
</tr>
<tr>
<td>5.11</td>
<td>p11702 is very strong at shooting and dribbling, so we choose to order the player blocks by shooting and color the player blocks by dribbling. The darker color indicates better rating.</td>
<td>54</td>
</tr>
<tr>
<td>5.12</td>
<td>Seven selected players are compared in player comparison view. Rating of shooting and dribbling attributes are selected to compare these players. Here, p11702 is set to be the reference player.</td>
<td>55</td>
</tr>
<tr>
<td>5.13</td>
<td>Once we set the equal zone to be 2. The comparison bar chart of dribbling attribute for p10705 no longer has the green background, which indicates the difference with p11702 is within 2%.</td>
<td>56</td>
</tr>
<tr>
<td>5.14</td>
<td>We add Averaged Dribble and Averaged Goal statistics into comparison. We also set p10705 as the reference player to see if p10216 is a better choice than p10705.</td>
<td>57</td>
</tr>
<tr>
<td>5.15</td>
<td>We set the range filter with a range from round 13 to round 32 so that all these seven players roughly play the same amount of matches. Here is the new player comparison view. p10705 is the reference player.</td>
<td>57</td>
</tr>
<tr>
<td>A.1</td>
<td>Overview of supported filters. There are five categories of different filters, namely, basic filter, statistic filter, rating filter, position filter and scatter plot matrix.</td>
<td>63</td>
</tr>
<tr>
<td>A.2</td>
<td>Range slider with color encoding indicating ratio of selected players. Here around 60 - 75% of all players are selected, who have played matches during round 12 to 18.</td>
<td>64</td>
</tr>
<tr>
<td>A.3</td>
<td>Example of a scatter plot matrix. On the top right, there are three scatter plots, which are showing the averaged statistics, while on the bottom left, the three scatter plots are showing accumulated statistics.</td>
<td>64</td>
</tr>
<tr>
<td>A.4</td>
<td>Selection supported for each scatter plot.</td>
<td>65</td>
</tr>
<tr>
<td>A.5</td>
<td>Position filter consists of 3x3 blocks which divide the football field into nine zones with equal size.</td>
<td>65</td>
</tr>
<tr>
<td>B.1</td>
<td>A player statistics table ordered by Dribble statistic attribute.</td>
<td>66</td>
</tr>
</tbody>
</table>
C.1 Illustration of how to draw stacked bars in the canvas with dynamic width. . . . . 67
D.1 Average position graph of a player. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
D.2 Event heatmap of the same player. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
D.3 We divide a football field equally into nine zones. . . . . . . . . . . . . . . . . . . . 69
D.4 Zones in an event heatmap. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
List of Tables

2.1 Lists of key performance indicators [19] .................................................. 7
2.2 Lists of top 20 attributes for attackers and defenders [15] ....................... 8
2.3 Comparison of non-hierarchical data visualization techniques. PCP stands for parallel coordinates plot and SPLOM stands for scatter plot matrix. .................... 15
2.4 Comparison of hierarchical data visualization techniques. ........................ 17

3.1 Feature definitions. The feature names are in Dutch and we make an English translation for each of them ................................................................. 20
3.2 List of all items for feature category, attribute and definition. .................... 21
3.3 Entity Event ............................................................................................... 23
3.4 Player ratings of accumulated and averaged statistics ............................... 24

5.1 Top 10 players in the whole season rated with equally weighted attribute tree. ... 50
5.2 Seven players selected from player similarity graph that are most similar to p11702 and have relatively high ratings. .................................................. 52
Chapter 1

Introduction

1.1 Motivation and goal

Football is one of the most popular sports in the world. In recent years, with the development of small wireless tracking devices, more and more emphasis has been put on football data analysis. During the match, player positions and game events are tracked by camera systems and football data companies like OPTA have been collecting and providing raw match data for several years. Even for team training, the football clubs have been applying wearable trackers to get additional training data like heart rate of their players to track the health condition and physical performance of the players. More and more professional clubs have formed their own football data analysis team to perform all kinds of analysis, from match performance to player comparison, based on the football data they have already collected. In the World Cup 2014, the German national team cooperated with SAP to analyze the match data to improve their performance [5], maybe this was also the secret weapon behind their winning the world cup that summer.

One of the benefits for football data analysis is helping team managers to decide which player is most suitable and still affordable to their own clubs. It is not always the case that buying star players can guarantee the championship. In the movie "Moneyball" [27], the general manager of the baseball team Oakland Athletics, Billy Beane, built a competitive baseball team in 2002 by doing intensive data analysis on undervalued baseball players. They won the first place in American League West with the third-lowest team payroll in the whole league (see Figure 1.1). Another good example is Leicester City, which is a football club in England Premier League. In the season 2014-2015, they barely avoided relegation and in the season 2015-2016, they became the new Champion of the league. The wage bill for Leicester City was only 48.2 million pounds [22], which was the fourth-lowest in the league, while on the contrary, Chelsea, the top in the league, paid 215.6 million pounds but just finished 10th after the whole season.

Memphis Depay (in the following work, we also refer to him as Memphis) was the key player for PSV in the Eredivisie season 2014-2015. He scored 22 goals for PSV in the league and was the best scorer. In the summer of 2015, he made his next step to Manchester United and making PSV fans wonder who will take his place in the club to score goals. Usually when a team sells its best player, the team may buy other players as well, so it raises the question who will be the best replacement.

In this thesis, we would like to use visualization techniques to deal with such tasks of which the main part is comparison so that we can find the most suitable players based on user interest. One of the biggest challenges is how to provide the user with the best flexibility to capture his very own definition of best player. Quite a lot of work has been done on visualizing football data, however, most papers [6, 16, 8] focus only on single match analysis. On the other hand,
CHAPTER 1. INTRODUCTION

Figure 1.1: MLB team salaries [27]

some papers [12] and websites like WhoScored.com introduce ranking systems for football players. However these ranking systems are fixed so the user is not able to modify the attributes that contribute to compute the ranking of players based on user’s interest. As a result, the user can not influence the ranking of players. In this thesis, we concentrate on players themselves rather than a single match performance and we would like to build a system in which users can have their influence on ranking players and thus comparing them.

1.2 Problem formulation

In this thesis project, the major problem is comparison. We would like to compare the performance of football players. It is relatively convenient to compare just two players, however, in our project, we would like to find an effective way to compare more than two players simultaneously. The target users will be the football enthusiasts or team managers who want to know more about the performance of a player. In the following sections, we will illustrate how to visualize these comparisons.

1.2.1 Visualize comparison

In Figure 1.2, we show the process by which we turn our original football data into a visualization. The original data we have is the spatio-temporal event based records from real Eredivisie matches from the season 2014-2015 and partly from the season 2015-2016. The detail of the data will be discussed later in Chapter 3.

1. The first thing we need to do is to extract the key attributes to determine the performance of the players. Examples of key attributes could be the number of shots or the number of goals scored. We refer to related research to identify the key attributes and apply necessary data processing to extract these key attributes correctly.
2. The next step is filtering. The key idea is to set limitations on different attributes to narrow the range of the subjects that we want to compare and output a selection of data for visualization.

3. The last step is applying suitable visual comparison techniques like juxtaposition, superposition or explicit encoding [18] for the previous selection of data to show the comparison between subjects. The details will be discussed in the following chapters.

4. Interaction is applied to help change the visualization and repeat Step 2 and Step 3 until the result is satisfying.

1.2.2 Related questions

In order to answer the question of "Finding Memphis", we derive the following questions that we would like to answer with our visualization tool.

1. We would like to have the flexibility to choose attributes for the ranking system based on user interest. For example, there are two coaches in the same team, they might have different opinions on what are the key attributes of Memphis. So, we want our tool to support such different opinions to rank players in a more flexible way.

2. We would like to find out the best players in Eredivisie. For example, in the season 2014-2015, Memphis was the MVP due to his brilliant performance in attacking but there could be other players that were also playing fantastic football regardless of their positions, maybe a great defender, we want to find these players as well.

3. We would like to find similar players. For example, PSV lost Memphis, who could replace him in the team?

4. We would like to compare the same player’s performance over time. For example, Memphis was the best attacking player of the season 2014-2015, but was he playing consistently well over the season?
CHAPTER 1. INTRODUCTION

1.3 Methodology and contribution

We apply fast prototyping for this thesis project. Fast prototyping is an iterative development approach which contains a three-step process: prototype, evaluation and refine.

- **Prototype:** the prototype step in this project is to design visualization according to the requirements and implement the design of the corresponding visualization.

- **Evaluation:** The evaluation step is to check the visualization implementation with the requirements to see whether all requirements are fulfilled as well as to find out whether there are new ideas needed.

- **Refine:** If we found some new ideas in review step, we need to derive the new requirements and refine the design so that all requirements can be met.

Overall, the major contributions of this thesis project are:

1. Introduce an interactive weighted attribute tree to support run-time modification on rating model.

2. Visualize rating and statistic of football player attributes for users to do further analysis to find out the most wanted players.

1.4 Organization of this thesis

The following chapters of this thesis are organized in the following way. In Chapter 2, related works on football data analysis and visualization are discussed. In Chapter 3, the football data set of this thesis project and related data processing are discussed. In Chapter 4, we discuss how we design this visualization tool including the visualization techniques applied as well as the reasons behind them. In Chapter 5, we show what this visualization tool can do and try to answer all the related questions mentioned in section 1.2.2. In Chapter 6, we draw a conclusion and point out the limitations and possible future works on this thesis project.
Chapter 2

Background and related work

In this chapter, we present background and related work on football data analysis and visualization. Since our project has a focus on comparison, we also cover comparison specific visualization techniques in this chapter.

2.1 Football data analysis

One of the most common reasons for football data analysis is to compare football players. In order to do so, the performance metrics for football players are needed first. In the following sections, the most popular metrics for football data analysis are discussed.

2.1.1 Statistic

Statistic is a very basic metric to compare football players. A football statistic is usually an accumulated or averaged number of a certain football event in a football match. The most seen statistics are goals, shots on/off target, assists, key passes, fouls, yellow/red cards, etcetera. During a football match TV broadcast, football statistics are usually shown with visualization (Figure 2.1), though sometimes they can be shown textually instead (Figure 2.2).

Accumulated statistic

Accumulated statistic is usually used to compare overall performance of players over a certain amount of time. A well-known example could be the ranking on goal scorer over a whole season (Figure 2.3). There is one slight problem if we just apply accumulated statistic to compare players...
which can be shown from Figure 2.3. There are several players that score exactly the same amount of goals like Michael de Leeuw and Michiel Kramer, they all score 17 goals, so how to decide which one is better? That is why in addition to accumulated statistic, averaged statistic also plays a role in football data analysis.

### Averaged statistic

Let us answer the previous question first. Michael de Leeuw and Michiel Kramer both score 17 goals, who is a better scorer? From Figure 2.3, we notice Michael de Leeuw has 28 appearances, which is fewer compared to Michiel Kramer. If we compute the averaged statistic goals per appearance, we can easily reach a conclusion that Michael de Leeuw is better. Since each appearance is counted whenever a player has played a match regardless of how many minutes he has played, a per 90 minutes averaged statistic is usually applied to solve the problem of different playing minutes. Can averaged statistic solve all problems? Actually no, if we have two players A and B who score exactly same amount of goals in exactly the same amount of playing time, who is better? That is why sometimes we need ratio statistic.

### Ratio statistic

Let us come back to the previous question. Player A and player B score exactly same amount of goals in exactly the same amount of playing time, which results in the same amount of goals per
90 minutes. Suppose player A has made 80 shots and player B has made 100, then we can use ratio of number of goals to number of shots to compare these two players. We can easily find out player A has a higher goals per shot indicating player A uses fewer shots to score more goals than player B. Now, we can say player A is a better scorer than player B.

2.1.2 Rating

Rating systems are widely used in sports to rank teams or players [12]. In general, to build a rating system, the key attributes that contribute to the rating must be identified first. Then, with a proper rating model, a rating can be computed. With a rating, it is relatively easy to rank players or teams. In the following sections, the basic approach to build a rating system is discussed.

Identify key attributes

Generally, there are two common ways to identify key attributes, which are consulting football experts [19] or applying machine learning [15]. In [19], Michael Hughes suggests a set of key performance indicators (attributes) for goal keepers and other generic players including strikers, midfielders and defenders, which is shown in Table 2.1.

<table>
<thead>
<tr>
<th>Performance Indicators</th>
<th>Goal keeper</th>
<th>Generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physiological</td>
<td>Height</td>
<td>Height</td>
</tr>
<tr>
<td></td>
<td>Strength</td>
<td>Strength</td>
</tr>
<tr>
<td></td>
<td>Power</td>
<td>Speed</td>
</tr>
<tr>
<td></td>
<td>Agility</td>
<td>Power</td>
</tr>
<tr>
<td></td>
<td>Coordination</td>
<td>Stamina</td>
</tr>
<tr>
<td></td>
<td>Reaction Time</td>
<td>Agility</td>
</tr>
<tr>
<td>Tactical</td>
<td>Vision</td>
<td>Support play</td>
</tr>
<tr>
<td></td>
<td>Organisation</td>
<td>When to cross</td>
</tr>
<tr>
<td></td>
<td>Communication</td>
<td>Passing</td>
</tr>
<tr>
<td></td>
<td>Distribution</td>
<td>Running off the ball</td>
</tr>
<tr>
<td>Technical-Defense</td>
<td>Shot stopping</td>
<td>Tackle</td>
</tr>
<tr>
<td></td>
<td>Coordination</td>
<td>Pressing opposition</td>
</tr>
<tr>
<td></td>
<td>Recovery speed</td>
<td>Interception-anticipation</td>
</tr>
<tr>
<td></td>
<td>Save</td>
<td>Clearance</td>
</tr>
<tr>
<td></td>
<td>Punch</td>
<td>Defensive header</td>
</tr>
<tr>
<td>Technical-Attack</td>
<td>Passing</td>
<td>Shooting</td>
</tr>
<tr>
<td></td>
<td>Throw</td>
<td>Heading</td>
</tr>
<tr>
<td></td>
<td>Ball control with feet</td>
<td>Reception</td>
</tr>
<tr>
<td></td>
<td>Kick</td>
<td>Passing</td>
</tr>
<tr>
<td></td>
<td>Tackle</td>
<td>Dribbling</td>
</tr>
<tr>
<td>Psychological</td>
<td>Concentration</td>
<td>Shooting</td>
</tr>
<tr>
<td></td>
<td>Motivation</td>
<td>Heading</td>
</tr>
<tr>
<td></td>
<td>Attitude</td>
<td>Reception</td>
</tr>
<tr>
<td></td>
<td>Body language</td>
<td>Passing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dribbling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Running with the ball</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Support play</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Crossing</td>
</tr>
</tbody>
</table>

Table 2.1: Lists of key performance indicators [19]
In Table 2.1, the list covers almost every aspect of a football player, however, usually only technical key performance indicators are covered in rating systems because only technical indicators can be easily measured in numbers.

In [15], Gunjan Kumar introduces the way to get key attributes with the help of machine learning. He retrieves both the ratings of football players given by experts and a full set of player attributes. Then he applies machine learning algorithms to determine the key attributes that influence rating decision. Table 2.2 shows the top 20 attributes for attackers and defenders that he found most important with machine learning. However, the attributes found by machine learning are not always reasonable. For example, in the top 20 defender attributes, we can see attribute Goals Open Play, Goals from Outside Box and Goals and it is clear that Goals already contains Goals Open Play and Goals from Outside Box, which makes people wonder why these goals has to be counted multiple times in the rating system, however, the machine learning algorithm will not be able to give reasonable explanations.

### Table 2.2: Lists of top 20 attributes for attackers and defenders [15].

<table>
<thead>
<tr>
<th>id</th>
<th>Attacker attribute</th>
<th>Defender attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red Cards</td>
<td>Red Cards</td>
</tr>
<tr>
<td>2</td>
<td>Error leading to Goal</td>
<td>Error leading to Goal</td>
</tr>
<tr>
<td>3</td>
<td>Last Man Tackle</td>
<td>Own Goal</td>
</tr>
<tr>
<td>4</td>
<td>Assists</td>
<td>Penalties Conceded</td>
</tr>
<tr>
<td>5</td>
<td>Penalties Not Scored</td>
<td>Shots Cleared off Line Outside</td>
</tr>
<tr>
<td>6</td>
<td>Goals from Throws</td>
<td>Goals from Direct Free Kick</td>
</tr>
<tr>
<td>7</td>
<td>Penalties Conceded</td>
<td>Clearances Off the Line</td>
</tr>
<tr>
<td>8</td>
<td>Successful Crosses Corners in the air</td>
<td>Left Foot Goals</td>
</tr>
<tr>
<td>9</td>
<td>Goals from Set Play</td>
<td>Unsuccessful Corners Left</td>
</tr>
<tr>
<td>10</td>
<td>Own Goal</td>
<td>Goals Open Play</td>
</tr>
<tr>
<td>11</td>
<td>Error leading to Attempt</td>
<td>Error leading to Attempt</td>
</tr>
<tr>
<td>12</td>
<td>Foul Won Penalty</td>
<td>Penalty Goals</td>
</tr>
<tr>
<td>13</td>
<td>Successful crosses in the air</td>
<td>Goals from penalties</td>
</tr>
<tr>
<td>14</td>
<td>Clearances Off the Line</td>
<td>Unsuccessful Corners Right</td>
</tr>
<tr>
<td>15</td>
<td>Goals Open Play</td>
<td>Goals from Outside Box</td>
</tr>
<tr>
<td>16</td>
<td>Goals from Corners</td>
<td>Goals from Set Play</td>
</tr>
<tr>
<td>17</td>
<td>Key Throw In</td>
<td>Goals</td>
</tr>
<tr>
<td>18</td>
<td>Goals from Outside Box</td>
<td>Foul Won Penalty</td>
</tr>
<tr>
<td>19</td>
<td>Foul Won Penalty excluding handballs penalties</td>
<td>Last Man Tackle</td>
</tr>
<tr>
<td>20</td>
<td>Successful Crosses Left</td>
<td>Assist</td>
</tr>
</tbody>
</table>

**Build rating model**

For most rating systems, a weighted sum rating model is applied to generate the rating. In [12], Ian, Philip and David introduce a rating model called EA Sports Player Performance Index that contains six sub-indexes that cover different aspects from match events. Each sub-index is assigned a fixed weight and then generate a weighted sum value as the rating for a player. Using this rating system, they make a ranking of players from English Premiere League for the season 2008-2009, the result is shown in Figure 2.4.
CHAPTER 2. BACKGROUND AND RELATED WORK

2.2 Football specific visualization techniques

In this section, we will focus on the visualization techniques that are widely used in football areas. We will illustrate different visualization techniques based on different information they try to visualize. The techniques might be from web media or research papers, but they all provides the intuition on how to visualize the football related data.

2.2.1 Formation

Formation indicates how the team organizes its players on the field and is one of the key aspects to determine the playing style of the team. A formation somehow indicates the effort that a team wants to put in attacking. The popular formations are 4-4-2, 4-3-3, 4-2-3-1, 5-3-2 and 3-5-2. The first number usually indicates the number of defenders and the number(s) in the middle usually indicates the number of midfielders and the last number indicates the number of strikers. Usually, the formation with more defenders are more defensive, on the other hand, the formation with more strikers are more attacking and in between if the formation has more midfielders, then the team will probably play more balanced. In Figure 2.5, we can see the initial formation for both teams in the match Manchester United against Crystal Palace. The two teams applied quite different formations for this game. On the left side, the orange circles indicates the Manchester United players and we can see the team was using 4-1-4-1 formation with only one defensive midfielder. On the other hand, Crystal Palace applied 4-2-3-1, which had two defensive midfielder. The initial formation only shows how the coach intention of how he wants their players to keep position but it is quite common that in the match, the players are not always on their position because of the opponents’ movement.

To have a better understanding of how the players keep their positions in a real game, a second way of showing the average position of the players is shown in Figure 2.6. Average position is calculated by averaging all the positions the player has been during the entire match and only shows the average position on the field. The figure is also from the game Manchester United against Crystal Palace. If we compare Figure 2.6 to Figure 2.5, we may find out in the average position graph that the average positions of the full backs for both teams end up almost in the midfielders’ position, which might indicate the full backs were actively involved in the attacking actions. In addition, the average positions for the left and right midfielders of Crystal Palace were quite defensive as their positions end up almost the same area covered by the two defensive midfielders. Also, for Manchester United, 7 players had their average positions in the opponents half while for Crystal
Palace, they only had 3, which might indicate Manchester United was a stronger team compared to Crystal Palace and Crystal Palace had to spend much effort on defense. Another interesting fact is that the left midfielder of Manchester United, number 9 Martial likes to cut inside from the wing since we can see his average position ended up more close to the middle of the field rather than the left side of the field.

2.2.2 Attacking sides

Attacking sides are used to tell which part of the field the team used more often when they attack. In Figure 2.7, we can see the attacking sides for the match Manchester United against Crystal Palace. The orange arrows indicate Manchester United and the blue arrows indicate Crystal Palace. We can see both teams used the right side to attack more often than the other two sides. However, Manchester United used the three sides more balanced compared to Crystal Palace.
2.2.3 Event frequency

Sometimes, we are more interested in how frequent a certain event happens in a certain area, in this case, we can apply heat map to visualize such information. A heat map is a graphical representation of data where the individual values contained in a matrix are represented as colors [29]. It’s quite common in football data visualization to use heat map to show either the active area of the players or the active area of different kind of match events like passes, shots, fouls and so on.

Figure 2.8: Heat map example

Figure 2.8 shows an example of heat map and we can know the basic elements that build the heat map from the figure. The first thing we need to have is the matrix. We can think of the matrix as a collection of blocks just like what is shown in the figure. The second thing we need is the value for each block. The last thing we need to do is to map a color to the value in the block. Sometimes there is a one-to-one mapping between color and block value indicating one block value only has one corresponding color and vice versa. But usually, we just map a range of block values to a corresponding color, for instance, we can assign color white to block value from 0 to 10. Usually, we need to define a color map before we apply the heat map, a color map is just a mapping from value to color. Figure 2.9 shows an example of color map.

In [2], heat map is applied to show the ball occupancy on the field, which is shown in Figure 2.10

Figure 2.9: Color map example

and we can find out the team preferred to pass the ball the left side of the field than the right side. In [6], heat map is applied to show the goal probability from different positions on football field.
2.3 High-dimensional data visualization techniques

In this thesis, we convert an event based data set to a high dimensional statistic based data set and visualize this data to compare football players. In this section, we focus on presenting related visualization techniques for high dimensional data.

2.3.1 Non-hierarchical data

If we want to compare number of goals, assists and key passes of two players, we just compare these attributes one by one thus each attribute is treated equally. In this scenario, there is no hierarchical structure for these attributes. For these kind of attributes, if we just have one or two attributes to compare, we can just compare the numbers directly. However, for most cases, we may have more than two attributes and we can use visualization techniques to compare these attributes. In the following sections, we introduce such visualization techniques and give a general comparison on them as well.

**Bar chart**

Bar chart is a very basic visualization technique, it is very useful in showing a single attribute. If we want to show multiple attributes simultaneously, we can simply use additional bars to visualize these attributes. In Figure 2.11, we can see an example [13] of a bar chart showing the percentage...
of the population living in urban areas in the world and in different continents from different year. Each different year is an attribute and the percentage of the population is encoded into the height of the bar. To compare the percentage, we can simply compare the height of the bars. In general, bar charts are helpful to look up and compare values.

**Stacked bar chart**

A stacked bar chart is a composition of several bar charts stacked vertically, which is shown in Figure 2.12. With a stacked bar chart, it is easy to compare the part-to-whole relationship of an attribute. In addition, it is still possible to look up the value for each attribute since the bar visualizes the absolute value of an attribute.

![Figure 2.12: An example of a stacked bar chart.](image)

**Pie chart**

A pie chart is used to show the part-to-whole relationship of an attribute, which is shown in Figure 2.13. Unlike a stacked bar chart, a pie chart only shows the relative contributions of parts to a whole. For example, here in Figure 2.13, the values for A, B, C and D are 10, 20, 30 and 40. If we modify the values to 1, 2, 3 and 4, we can still get the same pie chart.

![Figure 2.13: An example of a pie chart.](image)

**Parallel coordinates**

![Figure 2.14: An example of visualizing player attributes with parallel coordinates.](image)

A parallel coordinates plot is usually drawn with several vertical bars, which represent attributes. Usually, for each vertical bar, brushing is always supported for user to do some filtering on original data. The polygonal chain in parallel coordinates usually represents the entity that owns these attributes. In [9], the entity for parallel coordinates is a football player, which is shown in

Finding "Memphis" - a rating-based player comparison visualization 13
Figure 2.14. All the vertical bars in Figure 2.14 represent different attributes of a player. The parallel coordinates plot here is used to find correlations between attributes like if a higher value in attribute A always results a higher value in attribute B.

**Scatter plot matrix**

![Figure 2.15: An example of scatter plot matrices [21]](image)

A scatter plot is a very useful visualization for data exploration and finding correlations between attributes, however, for a scatter plot, there are only two axises, which could be inconvenient if we want to explore more attributes simultaneously. Scatter plot matrices are introduced to solve this problem. As is shown in Figure 2.15, one diagonal blocks of scatter plot matrices are filled with attributes, while for all other blocks, a scatter plot with attributes from horizontal and vertical directions is drawn. Once a user makes a selection in one scatter plot, all other scatter plots will also show the selection simultaneously. For instance, in [21], scatter plot matrices are used for multivariate data exploration to find flowers with *Sepal Length*, *Petal Length*, *Sepal Width* and *Petal Width* attributes, which is shown in Figure 2.15. One drawback for a scatter plot matrix is its scalability, since if there are too many attributes in the matrix, the matrix can be too complex to use and the individual scatter plots get to be too small.

**Conclusion for non-hierarchical data visualization techniques**

In Table 2.3, we list the tasks related to this thesis. We also compare all the non-hierarchical data visualization techniques that we have discussed to check whether these techniques can meet the requirement for each task.

### 2.3.2 Hierarchical data

Tree is a basic data structure widely used in computer science. For example, for football player rating, we may have an overall rating as root, which may consist of two children, attack rating and defense rating. For attack rating and defense rating, they may also have other children, which
Table 2.3: Comparison of non-hierarchical data visualization techniques. PCP stands for parallel coordinates plot and SPLOM stands for scatter plot matrix.

<table>
<thead>
<tr>
<th>Task</th>
<th>Technique</th>
<th>Bar</th>
<th>Stacked bar</th>
<th>Pie</th>
<th>PCP</th>
<th>SPLOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>look up value</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>compare value</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>part-to-whole relationship</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>find correlation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>find outliers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>locate cluster</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

form a football player rating tree. The advantage of hierarchical structure is its efficiency in query and the ability to add abstraction levels to hide the details of lower levels. In order to visualize such hierarchical data, we introduce some commonly used techniques in this section.

**Tree diagram**

![Figure 2.16: An example of tree diagram][4]

In Figure 2.16, a very basic tree visualization is shown. Each node in the tree is represented as a small circle with some text. The relation between parent and children is visualize with a link to connect parent node with children node. Interactions like expanding a node is usually supported for the user to see the detail of the tree. One disadvantage of a tree diagram is that although it shows the structure clearly, it can not visually show the part-to-whole relationship of a child node to its parent node. Another disadvantage of a tree diagram is its scalability, if we need to draw more tree structures simultaneously, it is quite difficult for a tree diagram to scale.

**Icicle plot**

As is introduced in [14], an icicle plot is also a way to visualize hierarchical data. Unlike a tree diagram, an icicle plot is a space filling technique to visualize a tree structure. An example of an icicle plot is shown in Figure 2.17. Each node in a tree is represented as a rectangle. A parent node is usually visualized above a child node. The top most rectangle is the root. Since icicle plots apply space filling, each node is required to have a size. The size of a parent node is the sum of all sizes of children nodes. For example, in our football player rating example, if a player's attack rating is 80 and his defense rating is 60, we could say the size of the attack rating node is 80 and 60 for the defense node. Suppose we do not have weights in this rating system, the size for overall
rating node is then 140 since the overall rating node is the parent for attack and defense nodes. With an icicle plot, we can now visually see the part-to-whole relationship of a child node to its parent node. However, the overall size of a node in an icicle plot is the sum size of the node itself and its children nodes, the scalability of an icicle plot still could be a problem if we want to draw a large number of icicle plots simultaneously.

**Treemap**

Treemaps are also introduced in [14]. Just like icicle plots, treemaps also apply space filling. The difference is the way to arrange children nodes. Icicle plots put children nodes beneath their parent nodes while treemaps put them inside parent nodes, which also solves the scalability problem. An example of a treemap is shown in Figure 2.18. There are several different algorithms to generate different styles of tree maps. Widely used algorithms are slice and dice [3], squarified [17] and strip [3].
Conclusion for hierarchical data visualization techniques

In Table 2.4, we list the tasks related to this thesis project. We also compare all the hierarchical data visualization techniques that we have discussed to check whether these techniques can meet the requirement for each task.

Table 2.4: Comparison of hierarchical data visualization techniques.

<table>
<thead>
<tr>
<th>Task</th>
<th>Tree diagram</th>
<th>Icicle</th>
<th>Treemap</th>
</tr>
</thead>
<tbody>
<tr>
<td>show hierarchical structure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>look up value for each level</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>part-to-whole relationship</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>scalability to large number</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

2.4 Comparison specific visualization techniques

In our project, one major goal is to compare players based on football data, we also need to find suitable visualization techniques to meet this requirement. In this section, we will focus on possible visualization techniques for comparison.

Figure 2.19: Different visualization techniques for comparison [18].

2.4.1 Juxtaposition

Juxtaposition designs place objects next to each other and rely on the viewer’s memory to make the connection between objects [18], that’s why juxtaposition is also known as separation. Figure 2.19 (a) shows an example of juxtaposition and we can see two separate plots showing the data of sensor X and sensor Y are placed side by side for the viewer to compare. Small multiples [28] is also a type of juxtaposition since several small plots are put together making it possible to make comparison.

In Figure 2.20, we can see an example from [1], which compares two players side by side and the different parts of body with color encoding indicating different categories. For example, head represents head ball and in this case, the player on the right has better skills in head ball than the one on the left.

In [9] [11] [16], small multiples have been used to show similar event chains. An example of such clusters from [16] is shown in Figure 2.21. The color of the chains in the small multiples indicates the different type of events.
CHAPTER 2. BACKGROUND AND RELATED WORK

2.4.2 Superposition

Superposition designs overlay multiple objects to present them at the same time and same place [18]. An example can be found in Figure 2.19 (b). We can see the data of sensor X and sensor Y are placed in the same plot resulting in better comparability.

In [9], superposition is applied to show different players’ features, which is already shown in Figure 2.14. We can see parallel coordinates [10] is applied and each line in the graph represents a single player so that we can compare all the players (lines) at the same time.

In [8], Marijn Grootjans introduces the "wave" to visualize the momentum during a football match. The performance of both teams are shown in the same figure, which is shown in Figure 2.22. The red area in the figure indicates the performance of the Spanish team and the blue area indicates the performance of the Dutch team.

2.4.3 Explicit encoding

Just as is shown in Figure 2.22, the white line itself is a type of explicit encoding which shows the performance difference between two teams. In general, explicit encodings compute relationships between objects and provide visual encoding of the relationship directly [18]. In Figure 2.19 (c), we can see the difference of data value between sensor X and sensor Y is directly plotted. The positive value indicates sensor X has larger value than sensor Y and the negative value indicates sensor X has smaller value. The advantage of this technique is if we can find a relation between objects, the visualization for the relation could be simple. However, it might take some effort to find a suitable relation.
Chapter 3

Data processing

The original data we have is the spatio-temporal match data of Eredivisie season 2014-2015 and 2015-2016. Since the data for season 2015-2016 only contains 13 rounds of overall 34 rounds, we apply the data from season 2014-2015 for most of our research. The original data contains all the events that occurred in each match. For each record, there are several features and among them one is timestamp for each record indicating when the event occurs and the other is coordinate of X and Y indicating where the event occurs on the field. Overall, for Eredivisie season 2014-2015, we have data for 34 rounds, 306 matches, 18 teams, 516 players and 457414 events. Because of the confidential agreement, all the team and player names have been anonymized.

In the following sections, we discuss how we turn original data into player statistics and ratings in our project.

3.1 Data description

3.1.1 Feature description

In the original data set, each row is an event record and each column is a feature. There are 12 columns for each event record so we have 12 features all together. Since the original description is in Dutch, we make a English translation for each feature and in the following sections, we use the English translation instead of the original Dutch word. With all these features, we can then identify all kinds of match statistics for each player.

3.1.2 Identify statistics from events

First of all, let us define what a statistic is. From the 12 features listed in Table 3.1, the combination of category, attribute and definition can be used to identify a specific statistic. For each statistic, there can be only one category, but for attribute and definition there can be multiple items.

If we denote the set of category with $\mathbb{C}$, the set of attribute with $\mathbb{A}$ and the set of definition with $\mathbb{D}$. We can define a statistic $(c, a, d)$ with the following equation. For a statistic, $c$ is an arbitrary category in $\mathbb{C}$, $a \in P(\mathbb{A})$ and $d \in P(\mathbb{D})$. $P(\mathbb{A})$ and $P(\mathbb{D})$ are the power set of attributes and the power set of definition.

$$(c, a, d) \in \mathbb{C} \times P(\mathbb{A}) \times P(\mathbb{D}) \quad (3.1)$$

All possible elements for the set of category, attribute and definition are shown in Table 3.2.

For example, if we want to identify successful long pass with right foot statistic, for category we choose pass, for attribute we choose right foot and for definition we choose isPassLong and isPassCompleted. If we use Equation 3.1 to formulate successful long pass with right foot statistic,
Table 3.1: Feature definitions. The feature names are in Dutch and we make an English translation for each of them.

<table>
<thead>
<tr>
<th>Feature</th>
<th>English Translation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tijd</td>
<td>Time</td>
<td>It is used to tell when the event takes place. The unit for this feature is ms.</td>
</tr>
<tr>
<td>Helft</td>
<td>Half</td>
<td>In a football match, there are two halves, the first and second half. This feature is used to indicate in which half the event takes place. The value should be either 1 or 2.</td>
</tr>
<tr>
<td>Effectiviteit</td>
<td>Effectivity</td>
<td>It is used to indicate the importance of the event.</td>
</tr>
<tr>
<td>Categorie</td>
<td>Category</td>
<td>It is used to identify the type of the event.</td>
</tr>
<tr>
<td>Speler</td>
<td>Player</td>
<td>It is used to identify the player that the event belongs to.</td>
</tr>
<tr>
<td>Team</td>
<td>Team</td>
<td>Team is used to identify the team that the event belongs to.</td>
</tr>
<tr>
<td>Attribuut</td>
<td>Attribute</td>
<td>It is used to describe the property of the event.</td>
</tr>
<tr>
<td>Definitie</td>
<td>Definition</td>
<td>It is used to describe the outcome of the event.</td>
</tr>
<tr>
<td>Wedstrijd</td>
<td>Match</td>
<td>It is used to identify which teams are playing the match.</td>
</tr>
<tr>
<td>Ronde</td>
<td>Round</td>
<td>It is used to identify in which round of the season the match is playing.</td>
</tr>
<tr>
<td>LocationX</td>
<td>LocationX</td>
<td>LocationX is used to indicate the X(length) coordinate of the event. The value is between 0 to 100.</td>
</tr>
<tr>
<td>LocationY</td>
<td>LocationY</td>
<td>LocationY is used to indicate the Y(width) coordinate of the event. The value is between 0 to 100.</td>
</tr>
</tbody>
</table>

we can write the following.

\[(\text{pass}, \{\text{right foot}\}, \{\text{isPassLong, isPassCompleted}\})\] (3.2)

Now with the definition of statistic, given a set of events $E$ and a target statistic $s$, for each event $e$ in $E$, we can easily check if this event $e$ matches the statistic $s$. We also denote category with $c$, attribute with $a$ and definition with $d$. We can define what this match function does with the following equation.

\[
\text{match}(e, s) = \begin{cases} 
1, & \text{if } s.c = e.c \text{ and } s.a \subseteq e.a \text{ and } s.d \subseteq e.d \\
0, & \text{otherwise}
\end{cases} \quad (3.3)
\]

Whenever the event $e$ matches the statistic $s$, this match function returns 1, otherwise returns 0.
Table 3.2: List of all items for feature category, attribute and definition.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>pass, defending action, attacking action, throw in, reception, goal attempt, goal kick, foul, indirect free kick, player out of play, player back in play, interception, corner, dribble, offside, save on goal attempt, direct free kick, yellow card, goal, touch, penalty, red card, own goal for, own goal</td>
</tr>
<tr>
<td>Attribute</td>
<td>right foot, direct, left foot, duel touched, head, duel untouched, body, blocked, clearance, cross pass, over, positioning, on cross pass, curved out, high, saved by the keeper, caught, ground, throw keeper, through, on deep pass, in the air, punched, sliding, overtaking opponent right, fake pass, off target, overtaking opponent left, fair play, low, curved in, left leg, goal, time delay, right leg, hands, protest, volley, straight, on the left post, right corner, on the right post, on the crossbar, saves a goal, left corner, through the center, simulation, goal attempt opponent, on the post</td>
</tr>
<tr>
<td>Definition</td>
<td>isPassCompleted, isPassForward, isPassShort, isPassBackward, isPassWide, isPassLong, isDuelPart, isDuelWonByDefender, isDuelAir, isAerial, isPossessionLoss, isDuelStanding, isDuelWonByAttacker, isPossessionGain, isPossessionGainDuel, isPossessionGainInterception, isClearance, isPassIntoOpponentBox, isKeyAction, isShotOffTarget, isCornerCompleted, isCornerIntoOpponentBox, isShotOnTarget, isDuelGround, isCornerShort, isAssist, isGoal, isOwnGoal</td>
</tr>
</tbody>
</table>

3.2 Data model

The first step for our visualization tool to visualize data is actually reading the original data and store those data in a data model built in our tool. In this section, we introduce the data model used by our tool.

The original data set contains all match event data for the Eredivisie season 2014-2015. In the Eredivisie, there are 18 teams competing for the title and each team will play 1 home match and 1 away match against each of the other teams. As a result, each team plays 34 matches, which is 34 rounds. Our data model actually follows this general knowledge of the Eredivisie and the concept diagram of our data model is shown in Figure 3.1. The rectangles with text inside are the entities in our data model. The links with arrow show the composition relation between entities.

From Figure 3.1, we can see there are six entities in our data model, namely, the Eredivisie season entity, the Round entity, the Match entity, the Team entity, the Player entity and the Event entity. The Event entity is the most important entity in this data model because all the statistics are generated based on the properties held in Event entities. The properties of Event entities are shown in Table 3.3. Among these properties, category, attribute and definition are used to identify the statistics, as discussed in section 3.1.2.

We can also notice both Match and Player entity contain a list of Event entities. This does not mean the Event entities are stored twice. Actually, they are stored only once, and the reference to each Event entity will be used in the data model. The reason to keep a list of Event entities for both Match and Player entity is because we want to support two different query paths for a certain type of match event. One path is time based and the other is player based. An example for a time based event query is finding all shooting events that take place during round 1 to 10.
CHAPTER 3. DATA PROCESSING

On the other hand, finding all shooting events of Memphis is a player based query. Sometimes, we even can make a joint query like finding all shooting events of Memphis during round 1 to 10 in our data model.

The advantage of this data model is that we can achieve more efficiency when we make a query on a certain event compared to making a query in the original data set. For the original data set, whenever we make an arbitrary query, we have to go through all the 457414 records for all these queries. On the other hand, with this data model, we have to go through all the records only for the worst case that we want to find a certain event for all players and for a whole season.

Figure 3.1: Conceptual data model. The rectangles with text inside are the entities in the data model. The links with arrow show the composition relation between entities. The properties of an entity are either listed inside the rectangles or placed on the out-going links.
Table 3.3: Entity Event

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id is an identifier for the event. Id is an integer number starting from 0.</td>
</tr>
<tr>
<td>time</td>
<td>time indicates when the event takes place in the match.</td>
</tr>
<tr>
<td>player</td>
<td>player is used to find which player the event belongs to.</td>
</tr>
<tr>
<td>team</td>
<td>team is used to find which team the event belongs to.</td>
</tr>
<tr>
<td>match</td>
<td>match indicates in which match the event takes place.</td>
</tr>
<tr>
<td>round</td>
<td>round indicates in which round the event takes place.</td>
</tr>
<tr>
<td>coord</td>
<td>coord is a coordinate that consists of two numbers of X and Y indicating the location of the event.</td>
</tr>
<tr>
<td>category</td>
<td>category is used to identify the type of the event.</td>
</tr>
<tr>
<td>attribute</td>
<td>attribute is a list used to describe the property of the event.</td>
</tr>
<tr>
<td>definition</td>
<td>definition is a list used to describe the outcome of the event.</td>
</tr>
</tbody>
</table>

3.3 Generate statistic

In this section, we introduce how we can query events from our data model and how to turn events to actual statistics. As is introduced in section 3.1.2, we can use Category, Attribute and Definition features from Event entity to identify a statistic. If an event meets all the requirements for Category, Attribute and Definition of a statistic, we just say this event matches this statistic.

3.3.1 Accumulated statistic

In our data model, each player has a list of events. To generate an accumulated statistic, we just check through the whole list of events and find out all the events that match our target statistic. We can simply use Equation 3.4 to generate an accumulated statistic for a player \( p \) of a statistic \( s \). From the data model shown in Figure 3.1, we know \( p.events \) is a list of Event entities for player \( p \).

\[
\text{accumulated}(p, s) = \sum_{e \in p.events} \text{match}(e, s) \tag{3.4}
\]

3.3.2 Averaged statistic

Here, the definition for average is per match because in our data set, we do not have the data for how many minutes a player has played. We only know how many matches a player has attended. To compute an averaged statistic, we need to know the accumulated statistic and the appearance for a player. First, we need to define what appearance is. If we want to compute the appearance for a player \( p \), we can use the following equation. From Figure 3.1, we can see round and player are the properties of an Event entity. The general idea of this equation is to count the number of distinct round property values in the event list of player \( p \).

\[
\text{appearance}(p) = \#\{ e.round | e.player = p \land e \in p.events \} \tag{3.5}
\]

Now, we can simply use Equation 3.6 to compute an averaged statistic for a player \( p \) of a statistic \( s \).

\[
\text{averaged}(p, s) = \frac{\text{accumulated}(p, s)}{\text{appearance}(p)} \tag{3.6}
\]
3.4 Generate single statistic rating

In a football match, it is common that a player makes 30 passes, but only makes 5 shots. Although the number of passes is apparently much higher than the number of shots, we cannot say this player is better at passing than shooting. Because the best player in that match could make 60 passes, 30 passes in a match seem to be not really good. On the other hand, 5 shots in a match could already be the highest number. That is why we want to generate a meaningful number to set unified metric for all statistics. We call this meaningful number rating.

There are two important requirements for the rating, which are listed below.

1. The rating of all statistics should be bounded with unified minimum and maximum value.
2. The rating of a statistic should be able to show how well the player is compared to the best player with this statistic.

With these requirements, we decide to apply data normalization to generate a rating. Because data normalization always generates a number between 0 and 1. We can see the way we apply data normalization for a player \( p \) of a statistic \( s \) in Equation 3.7. In the equation, \( f \) indicates a function to generate the statistic and it can be either accumulated or averaged in our case. \( \text{Players} \) is the set of all players in the data set. \( \min_f \) and \( \max_f \) are used to find the minimum and maximum value of the statistic computed with function \( f \) from all players in \( \text{Players} \).

\[
\text{normalized}_f(p, s) = \frac{f(p, s) - \min_f(s)}{\max_f(s) - \min_f(s)},
\]

\[
(\max_f(s) = \max\{f(\text{player}, s) | \text{player} \in \text{Players}\}, \\
\min_f(s) = \min\{f(\text{player}, s) | \text{player} \in \text{Players}\})
\]

Since we want a rating between 0 and 100, we add a factor of 100 to the normalized value to generate the rating. The equation is shown below.

\[
\text{rating}_f(p, s) = \text{normalized}_f(p, s) \times 100
\]

Now we are able to generate and compare ratings for players of either an accumulated or an averaged statistic. However, comparing the rating of an accumulated or an averaged statistic alone cannot guarantee that we can find a better player. Just think about the following situations shown in Table 3.4. If we just compare the accumulated ratings, we can make a conclusion that player A is as good as player B in scoring goals but better than player C. On the other hand, if we just compare the averaged ratings, we can make a quite different conclusion that player C is the best player in scoring goals and player A is better than player B. The problem is that we always miss part of important information that helps to compare players if we only compare the rating of accumulated statistics or the rating of averaged statistics.

Table 3.4: Player ratings of accumulated and averaged statistics

<table>
<thead>
<tr>
<th>Player</th>
<th>Statistic</th>
<th>Type</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>goal</td>
<td>accumulated</td>
<td>90</td>
</tr>
<tr>
<td>B</td>
<td>goal</td>
<td>accumulated</td>
<td>90</td>
</tr>
<tr>
<td>C</td>
<td>goal</td>
<td>accumulated</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>goal</td>
<td>averaged</td>
<td>80</td>
</tr>
<tr>
<td>B</td>
<td>goal</td>
<td>averaged</td>
<td>60</td>
</tr>
<tr>
<td>C</td>
<td>goal</td>
<td>averaged</td>
<td>100</td>
</tr>
</tbody>
</table>
That is why we decide to aggregate accumulated and averaged statistic rating to a single one so that we can easily compare one rating rather than two to find out which player is better. A solution is to assign weights to accumulated and averaged statistic ratings, and sum them up to generate the aggregate rating for a statistic. Generally speaking, accumulated statistic is more important if we want to compare player’s performance over a whole season, that is why we decide to assign more weights to accumulated statistic rating than averaged statistic rating. With this aggregated rating, we can still find out the more efficient player since if two players have same accumulated rating, the player with better averaged statistic rating will have a higher aggregated rating. We can now update the equation to compute the rating for a player $p$ of a statistic $s$. We use $\alpha$ to denote the weight of the accumulated statistic rating.

$$rating(p, s) = \alpha \cdot rating_{accumulated}(p, s) + (1 - \alpha) \cdot rating_{averaged}(p, s) \quad (3.9)$$

In our tool, the weight for $\alpha$ is set to 0.85.

Using Equation 3.9, the rating for a certain statistic is higher if the statistic itself is higher. However, there are cases that we want to get a lower rating if the statistic is higher. For instance, the Foul statistic, we may want a player with more fouls to have a lower rating. We call such statistics negative statistics. In order to generate a rating for a negative statistic, we can simply reverse the rating computed with Equation 3.9. In Equation 3.10, we show how to reverse the rating for a player $p$ of a negative statistic $s$.

$$negative(p, s) = 1 - rating(p, s) \quad (3.10)$$

### 3.5 Weighted attribute tree

![Weighted attribute tree](image)

Figure 3.2: Concept drawing of a weighted attribute tree. All the leaf nodes are the real statistics and all other nodes belong to abstraction levels.

In player comparison, we may need to compare quite a lot of statistic ratings simultaneously, comparing them one by one is just not efficient. We introduce a weighted attribute tree to generate higher abstraction rating levels to solve the problem. This tree is also the rating model in our tool. Some nice properties of this weighted attribute tree are listed below.

1. A rating can be computed for any attribute in a weighted attribute tree.
2. Detail level of the weighted attribute tree can be controlled, the detail is illustrated in Figure 3.3.
3. A new weighted attribute tree can be defined by sub-trees of existing trees.
Figure 3.3: We can hide the children nodes of *Attack* attribute to get a new detail level of the weighted attribute tree.

4. A new weighted attribute tree can be defined by combining existing trees.

An attribute in a weighted attribute tree is a tree node. There are two different tree nodes, which are statistic nodes or abstraction level nodes. In Figure 3.2, all the leaf nodes in this tree must be statistic nodes and all other nodes are abstraction level nodes. All abstraction level nodes are defined by the user and he can create as many abstraction levels as he wants. For each attribute, the user can also modify the weight, which gives the user the flexibility to address the importance of attributes in the rating system.

For each node in a weighted attribute tree, it always has a weight. The weight of the root node is 1. The weight of a parent node equals the sum of the weights of its child nodes.

We can use the following equation to define an attribute.

\[
\text{attribute} = \text{weight} \times \text{children}, \text{children is a set of children attributes.}
\]

If children is empty, this attribute is a statistic node. (3.11)

Now, we can rewrite the weighted attribute tree in Figure 3.2 with this equation. We notice, for attributes *Goal*, *Assist*, *Clearance* and *Interception*, their children set is empty, which indicates these attributes are actually leaf attributes.

\[
\begin{align*}
\text{Overall} &= (1, \{\text{Attack, Defense}\}), \\
\text{Attack} &= (0.75, \{\text{Goal, Assist}\}), \\
\text{Goal} &= (0.5, \emptyset), \\
\text{Assist} &= (0.25, \emptyset), \\
\text{Defense} &= (0.25, \{\text{Clearance, Interception}\}), \\
\text{Clearance} &= (0.125, \emptyset), \\
\text{Interception} &= (0.125, \emptyset)
\end{align*}
\] (3.12)

### 3.5.1 Compute rating with a weighted attribute tree

To compute the rating with a weighted attribute tree, for each attribute node in the tree, we need to check if this node is leaf node. For leaf attribute nodes, since they are just normal statistic nodes, we can simply compute the rating of this statistic. For other abstraction level nodes, we need to compute the weighted sum rating of its children attributes. Equation 3.13 is used to compute the rating for a player *p* of an attribute *a* in a weighted attribute tree.
CHAPTER 3. DATA PROCESSING

\[\text{rating}(p, a) = \begin{cases} 
    a\text{.weight} \cdot \text{rating}(p, s), & \text{if } a \text{ is leaf (} s \text{ is the actual statistic of attribute } a) \\
    \sum_{\alpha \in a\text{.children}} \text{rating}(p, \alpha), & \text{otherwise} 
\end{cases} \] (3.13)

We can use the weighted attribute tree in Figure 3.2 to show how to compute the Overall rating of a player \( p \) with Equation 3.13.

\[
\begin{align*}
\text{rating}(p, \text{Overall}) &= \text{rating}(p, \text{Attack}) + \text{rating}(p, \text{Defense}) \\
  &= \text{rating}(p, \text{Goal}) + \text{rating}(p, \text{Assist}) + \text{rating}(p, \text{Clearance}) + \text{rating}(p, \text{Interception}) \\
  &= \frac{1}{2} \cdot \text{rating}(p, \text{Goal}) + \frac{1}{4} \cdot \text{rating}(p, \text{Assist}) \\
  &\quad + \frac{1}{8} \cdot \text{rating}(p, \text{Clearance}) + \frac{1}{8} \cdot \text{rating}(p, \text{Interception})
\end{align*}
\] (3.14)

3.5.2 Generate player specific weights

In a weighted attribute tree, modifying the weights one by one could be a time consuming task. As a result, we want to automatically generate the weights for the user.

The key idea of assigning the weights is simple, when a player gets a higher rating in one attribute, we just assign a higher weight to this attribute since this is the strong point of the player. When we want to find the similar player, we are going to find the players with similar strong points. We can use Equation 3.15 to compute the new weight of a certain attribute \( a \) for a player \( p \). \textit{attributelist} is a list of attributes that are used in the current rating model of a player.

\[
a\text{.weight} = \frac{\text{rating}(p, a)}{\sum_{\alpha \in \text{attributelist}} \text{rating}(p, \alpha)}
\] (3.15)
Chapter 4

System design

In this chapter, we show the design of the whole system. For each part, we choose some specific visualization techniques, we also give justifications for these choices.

4.1 Dashboard

Figure 4.1: PureSoccerFX dashboard. (1) Player navigator (2) Filter (3) Player statistic table (4) Player detail (5) Player comparison view (6) Player performance treemap (7) Player similarity graph.

Figure 4.1 shows the dashboard of our visualization tool, PureSoccerFX. FX indicates this tool is implemented with JavaFX. Most of the visualizations implemented are visible from the dashboard. In the following sections, we introduce these visualizations one after another.

4.2 Select player

As a visualization tool to compare football players, the first thing needed to be done is to find the players to compare with. In our tool, we provide several different ways to find those players. Namely, by player navigator, by player search, by filter and by player statistics table.
Figure 4.2 shows player navigator, which is used to find players through teams. The user will first find a team and then find a player from this team. The color encoded bars before teams and players indicate the rating of current team or player. The rating of a team is the average rating of the top eleven players from that team since for a playing match, only eleven players can be on the field from the same team. The width and opacity are used simultaneously to indicate whether the current rating is good or not. A larger width and larger opacity indicate a better rating. Once the user finds an interesting player, he can just click the mouse and the selected player becomes highlighted in other visualizations.

Player search is very efficient if the user knows exactly which player he wants to find. What he needs to do is simply type the name in the search bar and all the players that meet this name will be shown in the list below. Player search also supports typing in partial names. This is shown in Figure 4.3, where an example of searching a player is given.

Figure 4.4 shows all supported filters in our tool. With a filter, the user can easily decrease the number of interesting players. The details of filters are in Appendix A.

Figure 4.5 shows the player statistics table. The user can easily check player statistics in the table. The table can be ordered by an attribute of the selected column, which provides the user a way to find interesting players. The details of the player statistics table can be found in Appendix B.
CHAPTER 4. SYSTEM DESIGN

Figure 4.4: Overview of supported filters. There are five categories of different filters, namely, basic filter, statistic filter, rating filter, position filter and scatter plot matrix.

Figure 4.5: Player statistics table can be used to browse player statistics and select interesting players. The table can be ordered by the selected column. Here, the player statistics table is ordered by Dribble.

4.3 Interactive weighted attribute tree

As is introduced in section 3.5, a weighted attribute tree is used as the rating model of our tool. If the user wants to change the rating model, he needs to modify the weights for different attributes, which sounds simple. However, if there are a large number of attributes in the rating system, modifying these weights one by one will be time consuming.

We introduce an interactive weighted attribute tree to help the user interact with the weighted attribute tree more easily. Here, interactive indicates the user has the flexibility to modify the rating system and tree indicates we will build a hierarchical tree rating model to handle all the attributes.

There are three major requirements for the interactive weighted attribute tree:

1. User should have the flexibility to build a weighted attribute tree easily.
2. User should be able to control the detail level of the weighted attribute tree.
3. Interaction of modifying weights in weighted attribute tree should be user friendly.
For the first and second requirements, a tree view can simply provide the solution, which can be seen from the left part of Figure 4.6, we just need to make a single mouse click on the small black triangle in front of each attribute and we can show or hide the detail of our weighted attribute tree. To build a tree with tree view, we just need to find the node where we want to put a child node and with easy mouse clicks to finish the task. However, a tree view is not really handy for the third requirement since all the attribute nodes are placed separately, which makes it difficult to have an overview on how weights are distributed over all nodes in the tree. On the other hand, icicle plot has the advantage for the third requirement as all nodes from the same level are placed horizontally one after another. While nodes from different levels are placed vertically one after another. If the width of the node equals to the weight of the node, an icicle plot automatically enforces the weight of the node equals to the sum of the weight of its children nodes. An example of an icicle plot is shown in the right part of Figure 4.6. When a user wants to change a weight in an icicle plot, he just needs to drag the border of an attribute node and all the weights of other nodes should change accordingly. The user can easily have an overview on how the weights change in the whole tree. The details on the design of the tree view and the icicle plot in our project are discussed in the following sections.

4.3.1 Tree view - detail control

The major task for a tree view is to easily build a tree and control the detail level of a weighted attribute tree. From Figure 4.7, we can see all children nodes of \textit{shooting} are hidden and all children nodes of \textit{Passing} attribute are shown in the tree view. In front of each node, there is a small triangle, which can be used to show or hide the children nodes. In this way, the user can control what attributes should be visible in a player performance treemap (discussed in section 4.4) and a player similarity graph (discussed in section 4.5).

Another important information of each attribute is weight. For each attribute node, we apply a small bar chart to visualize weight. In a weighted attribute tree, weights are stored for each node, which is discussed in section 3.5. However, for the bar chart visualization, we compare the difference between visualizations with absolute and relative weights and decide to visualize relative...
weights in the bar chart visualization. The comparison of these two different visualizations can be found from Figure 4.8 and Figure 4.9. In Figure 4.8, the bar chart visualizes absolute weights, the disadvantage is obvious that when the absolute weight is very small, it becomes difficult to compare the weights of children nodes since the bar width could be extremely small. On the other hand, it is also quite difficult to check if the weights of children nodes add up to the weight of their parent node. In Figure 4.9, we can see the bar chart that visualizes relative weights. Now, it seems easy to compare the weights of children nodes but still a little difficult to check the fact that the weights of children nodes should add up to 1. To solve this problem, we introduce the bar chart visualization which visualizes the relative weights with an offset. The new visualization is shown in Figure 4.10 and we can now see it becomes easier to confirm the weights of children nodes indeed add up to 1. However, it becomes a little difficult to compare the weights of children nodes if the value of these weights are relatively close. Since in our rating system, the weights do not necessarily need to be very precious, we decide to apply the bar chart visualizing relative weights with an offset.

To convert absolute weight to relative weight for each node, we can simply use the absolute weight of current node divided by the absolute weight of its parent. Equation 4.1 is used to
compute relative weight of node $n$.

$$weight_{relative}(n) = \frac{n.weight}{n.parent.weight} \quad (4.1)$$

4.3.2 Icicle plot - weight control

Figure 4.11: Tree view and icicle plot visualizing Passing node.

The major task for an icicle plot here is to provide an overview of how the weights are divided to each attribute node and provide an easy interaction of dragging to modify the weights.

An icicle plot is a space-filling visualization, it uses rectangles filled with color to represent the nodes in a tree. In our case, the rectangles in the icicle plot represents the weights of attribute nodes in a weighted attribute tree and the color for each rectangle is identical to the color chosen in the tree view. In Figure 4.11, we can see an icicle plot on the right, which visualizes exactly the same weighted attribute tree as the tree view on the left side. In this icicle plot, the big rectangle on top represents the weight of Passing attribute, which is the parent of Key Pass attribute and Assist attribute. On the bottom level, the smaller rectangle on the left represents the weight of Key Pass attribute since it has a smaller weight and the rectangle on the right represents the weight of Assist attribute. We can see a clear border between Key Pass rectangle and Assist rectangle, which can be dragged to modify the weights between these two attributes. The changes in weight of their parent nodes and children nodes should be updated simultaneously. In order to achieve this, once the weight of a attribute changes, the weight of its parent node and the weights of its children nodes should be updated as well. The situation is shown in Figure 4.12.

Figure 4.12: The user dragged the border of node rectangle, the width of the rectangle changes and so does the weight. We just assume the weight of node changed with $\Delta w$, the weights of its parent and children nodes should update as well.

When the weight of node changed with $\Delta w$, no matter what value $\Delta w$ is, the weight of its parent should be simply re-computed by adding all the weights of children nodes. Algorithm 1 shows the idea to update the weight of parent node from bottom to top recursively.
CHAPTER 4. SYSTEM DESIGN

Algorithm 1: Recursively update weight of parent node from bottom to top.

updateParent(node)

    parent = node.getParent()
    if parent == null then
        return
    end
    weight = 0
    foreach child in parent.getChildren() do
        weight += child.weight
    end
    updateParent(parent)

We also need to update the weights of children nodes, the value of changed weight $\Delta w$ should be distributed to the weight of every child node according to the relative weight of each child node. Algorithm 2 is used to recursively update weights of children nodes from top to bottom.

Algorithm 2: Recursively update weights of children nodes from top to bottom.

updateChildren(node, $\Delta w$)

    old_weight = node.weight - $\Delta w$
    foreach child in node.getChildren() do
        child_old_weight = child.weight
        child.weight = (node.weight / old_weight) * child.weight
        updateChildren(child, child.weight - child_old_weight)
    end

4.4 Player performance treemap

![Figure 4.13: Conceptual treemap structure.](image)

We introduced weighted attribute tree as the rating model to compute player ratings. The conceptual treemap structure for our project is shown in Figure 4.13. Since we need to compare 516 players simultaneously, other tree structure visualizations like icicle plot or sunburst chart simply could not scale to such a large number with a limited area size. A treemap also makes it possible to give a player with better rating a larger block so that we can visually see the outstanding players by block size.

From Figure 4.14, we can see the overview of a treemap showing 516 players in Eredivisie. Each big block in the treemap represents a player, which is referred as player block. Here, all the player blocks have different size. We can see the player blocks on top are larger than the player
blocks on the bottom which is because these player blocks are sized and ordered by player rating. A player with a higher rating will get a larger block size and will also be placed on top. Inside the player block, there are several small blocks which represent the ratings of all the expanded attribute nodes in the weighted attribute tree, which is explained in section 4.3.1. The small blocks are referred as attribute blocks. The details of the player block and the attribute block are shown in 4.15.

For a player performance treemap, we have the following requirements.

1. Users can browse and select multiple player blocks from the treemap.
2. Users can get information about a certain player from the treemap.
3. Users can compare players using the treemap.

In the following sections, we discuss the design of the player performance treemap.

4.4.1 Interaction

To meet the first requirement, we design three different statuses for player blocks, namely, last selected status, selected status and mouse over status. Since we want the treemap to support
Figure 4.15: The big block is a player block. The small blocks inside are attribute blocks.

multiple selection, we emphasize the last user selection with last selected status so that the user can visually see the difference between last selected and other previously selected player blocks. This is illustrated in Figure 4.16.

Figure 4.16: Three different status for player blocks in a treemap.

When a player block is in last selected status, a yellow border with width of 4 pixels will be added around the player block. A dark blue border will be added if the player block is in selected status and a light blue border for mouse over status. We also assign priorities to these three status to define which status should be visually seen by the user if the player block holds more than one status simultaneously. In our design, last selected status has the highest, mouse over status has the second highest and selected status has the least priority. The reason to assign a higher priority to mouse over status than selected status is that treemap is also linked with player similarity graph (discussed in section 4.5), which shares the same status encoding with the treemap. If we do not assign a higher priority to the mouse over status, when we have the mouse over a player block in the treemap, we can not find out which one is in mouse over status in the player similarity graph.

4.4.2 Color encoding

Figure 4.17: Weighted attribute tree that is used to compute player rating. Here, Attack and Defense attributes are visible from player blocks in treemap.

Figure 4.18: Player block visualizing ratings of Attack and Defense attributes.
The color of attribute blocks inside a player block are defined by colors in the interactive weighted attribute tree. This tree also defines which attribute blocks should be visible to users and which should be hidden (discussed in section 4.3.1). In Figure 4.17, we can see a weighted attribute tree and in Figure 4.18, we can see a player block with attribute blocks visualizing the rating using this weighted attribute tree. However, the weighted attribute tree in Figure 4.17 is not always visible in our tool, so we need to help the user understand the meaning of all the attribute blocks. We design a mouse over tool tip showing the rating of each attribute as well as the color encoding to help the user understand the attribute blocks better. In Figure 4.19, the tool tip is corresponding to the player block in Figure 4.18, now it is easy for user to check the tool tip for the meaning of each attribute block. For example, the color dark green is used to visualize attack attribute and we can see Memphis gets a rating 65 out of the possible highest rating 70.

4.4.3 Compare players in treemap

One reason to choose treemap to visualize the rating of 516 players is that the user is able to have an overview of which players are probably among the best players at a glance once we put player comparison into this overview. We design three methods to compare players in a treemap, namely, by size, by order or by color. Generally speaking, we can also combine the usage of these different methods to compare at most three different attributes simultaneously in a treemap. In the following sections, we discuss these methods one after another.

Compare players from block size

The block size mentioned here refers to the size of player blocks. The size of each player block is related to the player rating computed with the weighted attribute tree. A higher rating results in a larger size of a player block. One drawback of comparing players from player block size is that if two players have very close ratings, the block size will be close which will be difficult for the user to identify which player is actually better. An example of this drawback is shown in Figure 4.20. It is difficult to tell whether player1 is better than player3 by block size. That is why we introduce comparing players from block order since the player blocks that are placed in front will always be better than those are behind.

Figure 4.20: It is difficult to compare player1 with player3 from player block size but if all the blocks are ordered from the highest rating to the lowest rating. And the blocks are placed from top left to bottom right. Then it is clear that player1 is better than player3.
CHAPTER 4. SYSTEM DESIGN

Compare players from block order

In order to compare players from player block order, we first need to apply an ordered treemap to arrange all the blocks. The algorithm we apply for treemap is called strip treemap [3]. We modified the original strip algorithm a little bit to adapt to our application. In the original algorithm, it will always order the list passed into it, but in our case, we only need to order the player blocks by a certain attribute rating. Algorithm 3 shows our modified strip treemap algorithm. 

Algorithm 3:

<table>
<thead>
<tr>
<th>Algorithm 3: Modified algorithm to draw a strip treemap</th>
</tr>
</thead>
<tbody>
<tr>
<td>stripTreemap(List blocks, boolean needOrder)</td>
</tr>
<tr>
<td>if needOrder then</td>
</tr>
<tr>
<td>sort(blocks)</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>foreach b in blocks do</td>
</tr>
<tr>
<td>List current_strip;</td>
</tr>
<tr>
<td>if AvgAspectRatio(current_strip + b) &lt; AvgAspectRatio(current_strip) then</td>
</tr>
<tr>
<td>current_strip.add(b)</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>drawStrip(current_strip)</td>
</tr>
<tr>
<td>current_strip.clear()</td>
</tr>
<tr>
<td>current_strip.add(b)</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>foreach b in blocks do</td>
</tr>
<tr>
<td>stripTreemap(b.getChildren(), false)</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

With a strip treemap, we can order all player blocks by an arbitrary attribute. The player blocks from highest rating to lowest rating are placed from top left to bottom right. In Figure 4.20, suppose we have drawn this treemap with strip algorithm, the order of players will be player1, player2, player3, player4 and player5. In this way, we can easily identify player1 is the best player.

Compare players from block color

The last method to compare players is by block color. The idea is to draw a single colored layer with different opacity on each player block. The opacity is determined by the attribute rating, a higher rating will results in a higher opacity value. Equation 4.2 can be used to compute the opacity of a player block visualizing the rating of attribute $a$ for a player $p$. Players is a list of all players. From the equation, we can find out the opacity changes linearly with the rating of the selected attribute.

$$
\text{opacity}(p, a) = \frac{\text{rating}(p, a) - \min(a)}{\max(a) - \min(a)},
$$

$$
\max(a) = \max\{\text{rating}(\text{player}, a) | \text{player} \in \text{Players}\},
$$

$$
\min(a) = \min\{\text{rating}(\text{player}, a) | \text{player} \in \text{Players}\}.
$$

In Figure 4.21, we can see all player blocks are colored by Dribble attribute, the player with more dribbles will get a higher opacity for its player block that get a darker color. But just like
the method to compare players by size, if two players have similar rating, it becomes difficult to identify which one is better.

Figure 4.21: Player blocks colored by Dribble attribute. The player with more dribbles will get a higher opacity value for its player block that results in a darker color. The color can be chosen by the user from a color picker.

4.5 Player similarity graph

In this project, we would like to find out the similar players from Eredivisie. In order to do so, we need to clarify what is similarity in our project. Suppose each player need to compute ratings for \( n \) attributes, then we can use a vector \( V \) to denote these \( n \) attributes and we call this vector the attribute vector. After computing ratings of each attribute in \( V \), each player will have \( n \) ratings of these \( n \) attributes, then we can use a vector \( R \) containing these \( n \) ratings to represent a player’s profile. We call this vector \( R \) the rating vector of a player. The following equation is used to compute each component in the rating vector \( R \) for a player \( p \) given an attribute vector \( V \).

\[
R_i = \text{rating}(p, V_i), 1 \leq i \leq n
\]  

(4.3)

Now we can convert our problem of comparing the similarity of players to comparing the similarity of vectors. One way to compare the vectors is by computing Euclidean distance [26]. Another way is to compute cosine similarity [25], which measures the cosine of the angle between two non zero vectors. Since we are more interested in if the players have similar relative rating structure instead of the absolute value, we use cosine similarity for player similarity comparison. Suppose we want to compare player \( p_a \) and player \( p_b \) and we use \( A \) to denote the rating vector of player \( p_a \) and \( B \) for player \( p_b \), we can use Equation 4.4 to compute the similarity of these two players.

\[
\text{similarity}(p_a, p_b) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}
\]  

(4.4)

To visualize the rating structure of a player rating vector \( R \), we can simply use a stacked bar chart. One important thing to mention is that we are not going to visualize the absolute value of attribute ratings, instead, we want to visualize the relative ratio. We can use Equation 4.5 to compute the relative ratio for each component given the rating vector \( R \) of a player \( p \).
\[ \text{ratio}(p, i) = \frac{R_i}{\sum_{j=1}^{n} R_j}, 1 \leq i \leq n \]  

(4.5)

In Figure 4.22, we can see the visualization of a player rating vector \( R \).

\[ R = (r_1, r_2, r_3) \]

![Figure 4.22: Player rating vector visualized with stacked bar chart.](image)

Figure 4.23 shows the overview of the player similarity graph. However, Rome is not built in one day, it still takes several steps to reach the final design. In the following sections, we discuss our design on the player similarity graph.

![Figure 4.23: Overview of player similarity graph](image)

### 4.5.1 Visual design of player similarity graph

The requirements for a player similarity graph is to help the user to easily find players that are similar to but also better than the target player on certain attributes. In another word, we require the following.

1. Users can compare player similarity in a player similarity graph.
2. Users can compare player ratings of different attributes in a player similarity graph.
3. Users can see the rating structures of each player.

For the first two requirements, a scatter plot can do the job well since it supports sorting by two attributes. However, a scatter plot can not meet the third requirement. Because the points inside a scatter plot are too small to show the rating structure. That is why we think of the idea to put the
CHAPTER 4. SYSTEM DESIGN

Figure 4.24: The players are ordered by similarity in horizontal direction and by rating in vertical direction. The player with the highest cosine similarity value and rating will be in the top left corner.

stacked bar chart inside a scatter plot. A conceptual drawing of our design is shown in Figure 4.24.

In horizontal direction, the players are ordered by similarity, the player with higher cosine similarity value will be placed more to the left. In vertical direction, the players are ordered by rating, the player with higher rating will be placed more to the top. One tricky design for the vertical direction is that on this direction, it is not really an axis but an ordering which means it is impossible for any of the two players to overlap in the vertical direction. With this design, we just need to focus on the top left corner to find the most interesting players with both higher similarity value and rating.

4.5.2 Interaction with player similarity graph

To help the user easily find out the desired players, several interactions are supported by player similarity graph. These interactions are introduced one after another in the following sections.

Link with player performance treemap

Player similarity graph shares the same statuses with player performance treemap (discussed in section 4.4), namely, last selected status, selected status and mouse over status. The detail about these status is discussed in section 4.4.1. The key idea of linking is that once a status of a player changes, the change in status will be visible simultaneously in both player performance treemap and player similarity graph. In Figure 4.25, we can see the linking between these two visualizations.

From scatter plot to stacked bar chart

The advantage of a scatter plot is that it can be used to locate clusters and order on two attributes. The advantage of a stacked bar chart is that it can show the part-to-whole relationship. We design our player similarity graph in such a way that we can keep both of these two advantages.

In our design, the player similarity graph uses both horizontal and vertical directions to order the stacked bar charts in it. If we make the width of these stacked bars small enough, then the player similarity graph will look like a scatter plot, which is shown in Figure 4.26. For comparison, in Figure 4.27, a normal view of stacked bar chart with same order of players is shown. In order to achieve this effect, we need to change the width of the stacked bar dynamically. The idea is to always keep the right end of the stacked bar with the least similar player always on the right end of the canvas. The detail of how to compute this width is in Appendix C.
CHAPTER 4. SYSTEM DESIGN

Figure 4.25: Player performance treemap and player similarity graph share the same status.

**Indicator bar**

On the right side of the player similarity graph, we design an indicator bar which also shows the status of each stacked bar, which is shown in Figure 4.28. When there are too many stacked bars shown in the player similarity graph, it might be difficult to identify the selected players from the stacked bars directly. With this indicator bar, the selected players can be easily accessed by moving the mouse over the indicator.

**Make selection**

In our design, the player similarity graph also supports selection interaction. The user can easily drag the mouse to make a selection on interesting players. All the players in the selection will be highlighted with color yellow. The indicator bar also highlights these players with color yellow. The overview of selection interaction is shown in Figure 4.29.
CHAPTER 4. SYSTEM DESIGN

Figure 4.26: Scatter plot view of player similarity graph. The advantage of a scatter plot is that it can be used to locate clusters and order on two attributes.

Figure 4.27: Stacked bar chart view of player similarity graph. The advantage of a stacked bar chart is that it can show the part-to-whole relationship.

Figure 4.28: Indicator bar is on the right side of the player similarity graph, which can be used to select players or to find selected players.
Figure 4.29: Player similarity graph supports selection on players.
4.6 Player detail view

Figure 4.30 shows the overview of player detail view. Basic information like player name, team name and appearance are shown in the left side of the detail view. In the middle, the player block from player performance treemap (discussed in 4.4) and the stacked bar from player similarity graph (discussed in 4.5) are shown. For player block, it is placed inside a dotted rectangle, whose size is used to indicate the rating of the best player. The user can have an overview on how good this player is compared to the best in Eredivisie. In the right, the composition of rating is shown in detail, where user can know how the rating of the player is computed. In addition, an event heatmap is used to show the player’s favourite area in the football field. At last, we can see a button called “Generate rating system”, which is used to generate the weights for the current weighted attribute tree to best fit the profile of the player shown in this detail view. The details of heatmap are discussed in Appendix D and the rating generation is discussed in section 3.5.2.

4.7 Player comparison view

Player comparison view is used as the final conclusion for selected players. The user can select the interesting attributes to generate attribute comparison bar charts to compare the performance of selected players. We can see an overview of the player comparison view in Figure 4.31. As the final conclusion of comparing players, we can see almost all the visualizations we used to compare players are placed in this view. In this way, the user can easily compare players in this view instead of going to other views for comparison.

The new visualization in this view is the attribute comparison bar chart. For each attribute comparison bar chart, there are three different statuses, which are equal status, better status and worse status. The status is visualized with a background color, which is shown in Figure 4.32. Better status is indicated by color green, equal status is indicated by color white and worse status is indicated by color red. From the color, we can not tell how much better or worse the player is compared to the reference player. That is why there is a difference bar visualizing the difference inside each colored block. The difference is represented by percentage, indicating the percentage that the player is better or worse than the reference player, which is computed with Equation 4.6. We also define a equal zone for attribute comparison bar chart. If the difference is within the range of the equal zone, equal status will be applied.

\[
difference(p, p_{ref}, a) = \frac{\text{rating}(p, a) - \text{rating}(p_{ref}, a)}{\max\{\text{rating}(\text{player}, a) | \text{player} \in \text{Players}\}}
\] (4.6)
Figure 4.31: Overview of player comparison view. Here we compare three players.

Figure 4.32: Illustration on how to draw a difference bar chart.
Chapter 5

Results and evaluation

In section 1.2.2, we introduced the related questions we want to answer with our visualization tool. In Chapter 4, we introduced the designs to help answer these questions. Here, we want to discuss the results of our design.

5.1 Find best players

The very first question we want to answer with our visualization tool is "Who are the best players in a whole season". The answer to this question is shown in the player performance treemap (section 4.4). A player performance treemap visualizes the rating of players based on a weighted attribute tree. Before we check the results for top 10 players, we first take a look at the weighted attribute tree we use to generate the rating of a player.

![Weighted Attribute Tree](image)

Figure 5.1: The weighted attribute tree we use to generate the rating. We choose 10 common statistics for the weighted attribute tree and assign an equal weight to each statistic.

The weighted attribute tree we use is shown in Figure 5.1. For simplicity, we choose 10 statistics and assign an equal weight to each statistic, which means the weight for each statistic is 0.1. From the icicle plot in Figure 5.1, we can see the leaf nodes have equal width, which confirms the weight for each statistic is the same. we can see 3 of the statistics are defense statistics and 7 of them are attack statistics, which makes this rating system tend to give a higher rating to attacking players.

In Figure 5.2, the player blocks in the treemap are placed from top left to bottom right
Figure 5.2: When we apply equal weights for all statistics in tree, p11402 ranks first. Here, we show all the leaf node ratings in the treemap.

Figure 5.3: We can hide the rating details. Here, we only show Attack and Defense ratings in the treemap.

according to the rating. We can see p11402 ranks first here. Since we expand all the attributes in the weighted attribute tree, there are ten attribute blocks inside each player block, which makes it difficult to see if the rating system tends to give a higher rating to attacking players. In Figure 5.3, we only show two abstraction level attributes Attack and Defense instead of showing all leaf attributes and we can see the green areas visualize the Attack rating and the orange areas visualize the Defense area. Just focusing on the player blocks in first row, it is clear most of the player blocks have larger green areas than orange areas, which indicates the rating system indeed tends to give a higher rating to attacking players. The top 10 players in this rating system are listed in Table 5.1. We can see only two defenders are in the list.

Table 5.1: Top 10 players in the whole season rated with equally weighted attribute tree.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Player</th>
<th>Rating</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p11402</td>
<td>73</td>
<td>Midfielder</td>
</tr>
<tr>
<td>2</td>
<td>p11702</td>
<td>72</td>
<td>Attacker</td>
</tr>
<tr>
<td>3</td>
<td>p10418</td>
<td>71</td>
<td>Midfielder</td>
</tr>
<tr>
<td>4</td>
<td>p10705</td>
<td>69</td>
<td>Attacker</td>
</tr>
<tr>
<td>5</td>
<td>p11205</td>
<td>69</td>
<td>Defender</td>
</tr>
<tr>
<td>6</td>
<td>p11717</td>
<td>69</td>
<td>Defender</td>
</tr>
<tr>
<td>7</td>
<td>p11201</td>
<td>68</td>
<td>Midfielder</td>
</tr>
<tr>
<td>8</td>
<td>p11705</td>
<td>68</td>
<td>Midfielder</td>
</tr>
<tr>
<td>9</td>
<td>p10209</td>
<td>68</td>
<td>Attacker</td>
</tr>
<tr>
<td>10</td>
<td>p11502</td>
<td>67</td>
<td>Midfielder</td>
</tr>
</tbody>
</table>

5.2 Find similar players to a target player

The second question we want to answer is "which players have similar playing style like a target player". We can use player similarity graph to answer this question. Before we find similar players to a target player, we first modify the rating system a little bit to make the weights in the system more related to the strong points of the target player. Here, we choose the target player to be
p11702. We can check the rating detail of p11702 from player detail view, which is shown in Figure 5.4.

From the rating details, we can see p11702 is good at attacking attributes but poor at all defensive attributes. To find a similar player to p11702, we want to have a rating system that assigns more weights to attacking attributes and less to defense. We can see a button called “Generate rating system”, which is used to automatically generate a rating system based on player’s profile. We can click the button and check how the weights change in Figure 5.5.

We can see with the new rating system, the weights for attacking attributes increase and the weights for defensive attributes decrease, which is just what we want. For the overall rating, there is a ten-point increase. Before generating the specific rating system for p11702, the rating is 72, but now it is 82. Now we can find similar players to p11702. To do so, we need to check player similarity graph, which is shown in Figure 5.6. We can see p11702 is already in the top left corner of the graph. As we know, for the vertical direction of the player similarity graph, the more to the top indicates better rating and for the horizontal direction, the more to the left indicates more similar. So, p11702 is already the best player in this specific rating system, but we can still find similar players by dragging the mouse to make a selection of the top left corner to select players more similar to p11702 and have a relatively high rating. The selection interaction is shown in Figure 5.7, all the yellow highlighted bars indicate the selected players.

After selection, only seven players are left, which can be seen in Figure 5.8. The list of selected players is shown in Table 5.2. Although these players are very similar, due to the auto-scale of the similarity graph, the horizontal distance between p11702 and other players becomes more, which seems that the selected players are not similar enough to p11702. In order to have a clear overview on the rating structure of these players, we can drag the slider in the bottom left to adjust the width of the stacked bar inside the player similarity graph, Figure 5.9 shows the result when we
set the bar width to maximum. We can see the rating structures of these players are quite similar, which shows they are all typical attacking players. Now, we have found similar players to p11702, we still need to compare these players to find a best replacement. In the next section, we introduce how to compare players in our visualization tool.

Table 5.2: Seven players selected from player similarity graph that are most similar to p11702 and have relatively high ratings.

<table>
<thead>
<tr>
<th>Player</th>
<th>Rating</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>p11702</td>
<td>82</td>
<td>100%</td>
</tr>
<tr>
<td>p10705</td>
<td>77</td>
<td>99.48%</td>
</tr>
<tr>
<td>p11705</td>
<td>74</td>
<td>99.58%</td>
</tr>
<tr>
<td>p11512</td>
<td>72</td>
<td>99.61%</td>
</tr>
<tr>
<td>p11409</td>
<td>72</td>
<td>99.53%</td>
</tr>
<tr>
<td>p10216</td>
<td>70</td>
<td>99.69%</td>
</tr>
<tr>
<td>p11400</td>
<td>66</td>
<td>99.58%</td>
</tr>
</tbody>
</table>

Figure 5.6: Player similarity graph. The yellow line shows p11702, indicating he is on top of the graph. The numbers indicate the scales of the graph.
CHAPTER 5. RESULTS AND EVALUATION

Figure 5.7: We can drag the mouse to make a selection of players in the top left corner. We try to keep the selection box as left as possible because the players more to the left are more similar to p11702.

Figure 5.8: After selection, 7 players are left, including p11702 (with yellow border). The similarity graph automatically scales and the numbers indicate the new scales after auto-scaling.

Figure 5.9: We can change the width of the stacked bar charts inside the player similarity graph to make it full scale. Now we can see all the selected players are very similar in rating structure.
CHAPTER 5. RESULTS AND EVALUATION

5.3 Compare Players

In our visualization tool, comparison is the core. We embedded comparison elements in player performance treemap. We also implement a separate player comparison view to help user make a conclusion on which player is the best choice. In this section, we will continue to compare the seven players listed in Table 5.2 and try to find the best replacement for p11702.

5.3.1 Compare players in player performance treemap

In a player performance treemap, there are three methods to compare players, namely, by player block size, by player block order or by player block color. In Figure 5.10, the default setting is shown. The player blocks are sized and ordered by the same Overall rating. The order of players is p11702, p10705, p11705, p11512, p11409, p10216 and p11400. Since p11702 is very strong at shooting and dribbling, so we choose to order the player blocks by shooting and color the player blocks by dribbling. We can see the results in Figure 5.11. Now the order is determined by rating of shooting attribute and the new order of players is p11702, p11512, p11705, p10705, p11409, p11400 and p10216. If we only compare shooting, then p11512 seems to be a good replacement for p11702. However, we still need to compare dribbling. The color indicates the rating of dribbling attribute, the darker color indicates better rating. We can notice p10705 has almost the same color as p11702, which means their dribbling ratings are similar. He also has reasonable shooting rating, which makes him a possible replacement for p11702.

It is nice that we can compare players from player performance treemap, however, if we want to compare more than three attributes simultaneously, it will be difficult to just use a player performance treemap. That is why we design the player comparison view, which is discussed in the following section.

5.3.2 Compare players in player comparison view

In Figure 5.12, we can see the player comparison view showing the selected seven players. We select rating of shooting and dribbling to compare these players. We can see different visualizations are all placed here to compare including the player blocks from player performance treemap, the stacked bar from player similarity graph, the event heatmap from player detail view and the attribute comparison bar chart. In an attribute comparison bar chart, a reference player is selected.
CHAPTER 5. RESULTS AND EVALUATION

Figure 5.12: Seven selected players are compared in player comparison view. Rating of shooting and dribbling attributes are selected to compare these players. Here, p11702 is set to be the reference player.

Here, p11702 is the reference player. The background color of an attribute comparison bar chart indicates whether current player is better than the reference player. A green background indicates better status, a white background indicates equal status and a red background indicates worse status. The order of players is determined by the selected attributes, a player with better sum rating on these selected attributes is placed more to the left. Here, after selecting the rating of shooting and dribbling attributes, the rank of players is p11702, p10705, p11512, p11705, p11409, p10216 and p11400. We can also notice only p10705 has one rating better than p11702, which is dribbling rating. While for all other players, they all have worse ratings than p11702.

Sometimes, we may want to ignore the small difference between players so that the background color will only show green or red if the difference is larger than a certain value. We can realize this equal zone by dragging the slider in the top left corner of the player comparison view. In Figure 5.12, we can see the equal zone is set to be 0. In Figure 5.13, we can see a different setting of equal zone, which is set to be 2 indicating the difference within 2% will be ignored. Once we set the equal zone to be 2. The comparison bar chart of dribbling attribute for p10705 no longer has the green background, which indicates the difference with p11702 is within 2%.

From Figure 5.13, we can see p10705 is the best replacement for p11702 so far. He also likes to play in the front left of the football field just like p11702. He has equal dribbling rating as p11702 and slightly lower rating in shooting, but compared to others, he is the one to choose.

However, when we check the appearance of these players, one player draws our attention again, which is p10216. He also likes to play in the front left of the football field. However, he just played 21 matches while others played around 30 matches, which is a big difference. Since the accumulated statistic contributes a lot to the final rating, it is possible that p10216 gets a lower rating because he did not play as many matches as others. In order to find out whether p10216 could be a replacement for p11702, we add two averaged statistics into comparison, which is shown in Figure 5.14.

We add Averaged Dribble and Averaged Goal statistics into comparison. We also set p10705 as the reference player to see if p10216 is a better choice than p10705. We notice p10216 is very
strong at *Dribble*, because his averaged number is higher than p10705. For *Averaged Goal*, he has a slightly lower number. Besides p10216, p11512 also draws our attention. We can see he is very good at scoring goals. He has a better *shooting* rating and better averaged goals than p10705. However, p11512 likes to play in the front right rather than front left, which is quite different from p11702. If we do not care about positions, then he could be a good replacement. But here, we still consider position as an important attribute, so we will go back to check more about p10216. We still want to know if p10216 has played the same amount of matches, will he get a better rating?

In our tool, there is a very useful round filter that makes it possible to compare player ratings in a range of certain rounds. Our goal is to find a range of rounds in such a way that all these seven players play roughly the same amount of matches. If we can find such a range, we can then check if p10216 can have a better rating as expected. The range we find for rounds is from round 13 to round 32. The new player comparison view is shown in Figure 5.15. p10705 is still set to be the reference player. We notice that the overall rating of p10216 increases a lot and becomes the second best in these seven players. For all the selected attributes, p10216 has same *shooting* and *dribbling* rating as p10705, but better averaged goals and averaged dribbles. From this player comparison view, we can confirm if p10216 could play as many matches as other players, he is actually among the top players.

In conclusion, from the full season’s point of view, p10705 plays more matches and contributes more. He also has similar *shooting* and *dribbling* ratings to p11702 and also plays in the front left just like p11702, which makes him a possible replacement of p11702. On the other hand, p10216 players fewer matches, but he proves he can be as good as one of the best players if he could play as many matches as others. If we want to find a promising player to replace p11702, then p10216 will be a better choice.
Figure 5.14: We add *Averaged Dribble* and *Averaged Goal* statistics into comparison. We also set p10705 as the reference player to see if p10216 is a better choice than p10705.

Figure 5.15: We set the range filter with a range from round 13 to round 32 so that all these seven players roughly play the same amount of matches. Here is the new player comparison view. p10705 is the reference player.
Chapter 6

Conclusions

In this chapter, we first draw the main conclusion and then discuss the limitations of our visualization tool and the possible future work.

6.1 Main conclusion

We designed and implemented a prototype visualization tool to compare the performance of football players. We introduced the interactive weighted attribute tree to make abstraction levels attribute that can hide the detail of the rating system and give the user the flexibility to build or modify their own rating system at run-time. To find similar players, we implemented automatic weight generation for a certain player to build a specific rating system to find similar players. In addition, we designed a player similarity graph, which combines the advantage of scatter plots and stacked bar charts. The user can easily locate a group of similar players and make selections for further comparison. For player comparison, we designed a player performance treemap that can compare players with player block size, order or color. We also designed a player comparison view to help the user compare all the visualizations appeared in the tool to make the final conclusion on which players are the better ones.

We made some sample analysis of questions like "Who are the best players in the whole season", "Which players have similar playing style like a target player" and "Who is the best replacement of a target player" to show our tool can solve all the problems mentioned in 1.2.2. We also showed that our tool can provide a complete work-flow from browsing players to getting the final conclusion.

6.2 Limitation

Limitations of this study are listed as follows.

- Only event positions are included in the original data set. All the position based conclusion in this study is not accurate since we do not have the positions of a player when he is running without a ball.

- The coordinates in the original data set are not correct. We fixed most of the coordinates, however, some of the coordinates are still not correct, which may cause some inaccuracy in identifying player positions.

- Averaged statistic is per match instead of per minute. Due to lack of playing minutes for players, we can not generate the more accurate per playing minute statistics. This may cause some unfair comparisons between players.
There is no user study conducted during this research. It is possible that the functions provided in this tool are not the most wanted ones by the user.

6.3 Future work

From data processing aspect:

- Our tool now supports accumulated and averaged statistics and we can generate ratings based on these statistics. However, the ratio statistics are not supported yet. Many ratio statistics are widely used in football statistic analysis right now like goal conversion rate and pass success rate. It is possible to make special ratio statistics for scoring or passing, but we want to make it as flexible as possible, so that the user can define their own ratio statistics in the future.

- In our tool, the rating of a statistic equals to the weighted sum of the accumulated statistic rating and the averaged statistic rating. The weights for these two ratings are now fixed in our tool. However, it will give user more flexibility if the user can choose the weights and that is also one of the possible improvements of our tool.

From visualization aspect:

- For the main dashboard, it would be great to upgrade to a docking system. Right now, the layout is a slight problem for user experience as the user may need to drag the border of a view to frequently modify the size. With a docking system, the user can simply maximize or minimize the view and set the layouts of their own will, which will be more user friendly.

- For the interactive weighted attribute tree, right now, the user can only build a weighted tree from scratch, which will be time consuming. In the future, our tool should support importing an existing tree or combining existing trees into a new tree. Or from an existing tree to save a subtree as a new rating system.

From performance aspect:

- We developed our tool with model-view-controller design pattern. Due to the large usage of event listeners, some of the listeners have conflicts with each other, which makes the response time become longer. In the future, we should check and remove all the redundant listeners to provide better system response time.
Bibliography


[2] Peter Carr Yisong Yue Sridha Sridharan Alina Bialkowski, Patrick Lucey and Iain Matthews. Identifying team style in soccer using formations learned from spatiotemporal tracking data. 2014. ixix, 11, 12


BIBLIOGRAPHY


Appendix A

Details on filter

A.1 Filter

Filters are helpful to decrease the number of interesting players. In Figure A.1, an overview of supported filters are shown. There are five categories of different filters, namely, basic filter, statistic filter, rating filter, position filter and scatter plot matrix. A basic filter is used to filter on rounds of interest, which makes comparing players over time possible. It is also possible to compare appearances of a player in a whole season. A statistic filter is used to filter on player statistics, a tick box is used to distinguish between accumulated statistic and averaged statistic. A position zone filter is used to filter on player average position on a football field divided into nine zones. A scatter Plot matrix has the same function as a statistic filter, however, a scatter plot matrix can visually show the distribution of all players over the selected attributes, that is why we keep both in our tool. The result of filtering is shown on top right of this filter dashboard, where the number of selected players is shown as a fraction of the total.

Figure A.1: Overview of supported filters. There are five categories of different filters, namely, basic filter, statistic filter, rating filter, position filter and scatter plot matrix.
APPENDIX A. DETAILS ON FILTER

A.1.1 Range slider

![Range slider with color encoding indicating ratio of selected players. Here around 60 - 75% of all players are selected, who have played matches during round 12 to 18.](image)

Most of the filters mentioned before apply a range slider to set the bounds. The original range slider is from ControlsFX [7] and we make some modification based on the original one. Each slider bar contains two control points. The user can move these control points over the slider to set the bounds. The values for these control points are shown at the left end and the right end of the slider. The left end shows the lower bound and the right end shows the upper bound. The selected range is color encoded and the color is used to indicate the ratio of the number of selected players. The details of color encoding are shown in Figure A.2.

A.1.2 Scatter plot matrix

![Example of a scatter plot matrix. On the top right, there are three scatter plots, which are showing the averaged statistics, while on the bottom left, the three scatter plots are showing accumulated statistics.](image)

A scatter plot matrix is helpful for multivariate analysis, namely, analyze multiple football statistic attributes in our case. Figure A.3 shows a scatter plot matrix, with three statistic
attributes, Key Pass, Dribble and Open Play Shot. Since for football statistics, both accumulated and averaged statistics need to be considered when comparing players, we show both statistics in our matrix. On the top right above the diagonal, there are three scatter plots, which are showing the averaged statistics, while on the bottom left, the three scatter plots are showing accumulated statistics. Each point in the scatter plot represents a football player in the Eredivisie. For these points, there are four different status, namely, filtered out status, selected status, clicked status and mouse over status. Filtered out status indicates the point is not in the selection made by the user. Selection is an interaction supported by each scatter plot, which is used to select a number of interesting points to highlight them for further analysis. An example of a selection interaction is shown in Figure A.4. The points in the blue rectangle are in the selection. Selected status indicates the point is in the selection made by the user. Clicked status indicates the point is in the selection and has been clicked by the user. Mouse over status indicates the user is browsing the point for the time being.

Figure A.4: Selection supported for each scatter plot.

Figure A.5: Position filter consists of 3x3 blocks which divide the football field into nine zones with equal size.

A.1.3 Position zone filter

Position is very important in football. During each game, a player is usually assigned a specific position. For different positions there are always different roles. We may be interested for players from only a few positions, as a result, we need to find a way to filter out players on positions. In Figure A.5, the position filter is shown. Each block in the position filter indicates a specific zone on the football field. The details on zones are discussed in D.0.1. Since for football player roles, there are forwards, midfielders and defenders from front to back. Horizontally, there are left, middle and right. We decide to divide the football field into nine equally divided zones and if a player’s average position falls into a certain zone, this player will be selected if that zone is selected by the user. We give each zone a notation to indicate the position. LW means left winger, CF means center forward, RW means right winger, LM means left midfielder, CM means center midfielder, RM means right midfielder, LB means left back, CB means center back and RB means right back. Here in Figure A.5, LW and CF are selected.
Appendix B

Player statistics table

B.1 Player statistics table

![Player statistics table](image)

Figure B.1: A player statistics table ordered by Dribble statistic attribute.

In order to have an overview on the statistics of all players, it is convenient to show information in a table. That is why we have a player statistics table in our tool. One thing to mention is that only the players that meet the requirements of all the filters mentioned in section A.1 are visible in the player statistics table. An example of a player statistics table is shown in Figure B.1. Each row in the table is a statistics profile for a player while each column indicates a statistic attribute. The players in the table can be ordered by column. In Figure B.1, the table is ordered by Goal statistic. We can see p10505 ranks first and he is highlighted with a blue border, which indicates the mouse is now over this player.
Appendix C

Compute stacked bar width

In Figure C.1, an illustration of how to draw stacked bars in the canvas with dynamic width. Suppose we want to put $n$ stacked bars in one row and we know the width is the width of the canvas. We also want to place the stacked bar with minimum similarity in such a way that its right end will be just on the right end of the canvas. In order to draw all the stacked bars in the canvas, we need to know the coordinates for all the bars. We use the top left corner of a stacked bar as the coordinate and we denote this coordinate with $(x,y)$. Suppose we have a player list list ordered by player rating. Equation C.1 is used to compute the x coordinate of the $i_{th}$ player in the list and Equation C.2 is used to compute the y coordinate. $MAX\_WIDTH$ represents the width from similarity value 0 to 1 and the value $MAX\_WIDTH$ also changes dynamically with the change of $n$. We can use Equation C.3 to compute this value.

$$x_i = (1 - similarity(p_i, p_{\text{target}})) \cdot MAX\_WIDTH$$  \hspace{1cm} (C.1)$$

$$y_i = (i - 1) \cdot \frac{canvas.height}{list.size}$$  \hspace{1cm} (C.2)$$

Figure C.1: Illustration of how to draw stacked bars in the canvas with dynamic width.
APPENDIX C. COMPUTE STACKED BAR WIDTH

\[
MAX\_WIDTH = \frac{(n - 1) \cdot canvas.width}{n \cdot (1 - \text{min\_similarity})}
\]  \hspace{1cm} (C.3)

In the player similarity graph, we use a slider to dynamically change the number of \( n \) to change the width of the stacked bar thus making the change from scatter plot to stacked bar chart possible.
Appendix D

Heatmap design

D.0.1 Event heatmap vs average position graph

In our final design, we use an event heatmap, however, at first, we applied an average position graph instead. Figure D.1 and Figure D.2 shows the difference between an average position graph and an event heatmap. Both graphs are visualizing the same position information of a same player. We can clearly find out we can get more information from an event heatmap since we can see this player takes a lot of actions in the front and the left of the football field and he has more actions in the front than in the back while from an average position graph, we can only know this player likes to stay in the left of the field. As a result, we apply an event heatmap in our tool.

Figure D.3: We divide a football field equally into nine zones.

Figure D.4: Zones in an event heatmap.
To compute this event heatmap, we first need to divide the football field into several zones and then compute the frequency of events that occurs in those zones. In our case, we divide the field equally into nine zones. In football data analysis, in attacking direction, the field is usually divided into three zones, offensive third, middle third and defensive third. The football field can also be divided into left area, center area and right area. The zones are illustrated in Figure D.3. The small numbers along the side line of the football field are the coordinates and the big number inside the field is used to identify the different zones. For example, zone 1 is in the most left and most front of the football field. In Figure D.4, we can see how these nine zones are mapped into the event heatmap. Now, we need to compute the frequency of events occurs in each zone. We can use Equation D.1 to compute this frequency for a player $p$ in a zone. $\text{zone list}$ is a list of all the zones shown in Figure D.4.

$$
frequency(p, \text{zone}) = \frac{\#\{e | e \in p.events \land e.coordinate \in \text{zone}\}}{\sum_{z \in \text{zone list}} \#\{e | e \in p.events \land e.coordinate \in z\}}$$  \hspace{1cm} (D.1)

Then we can set the opacity for each zone to generate the heatmap visualization, the value of opacity is identical to the frequency value computed with Equation D.1.