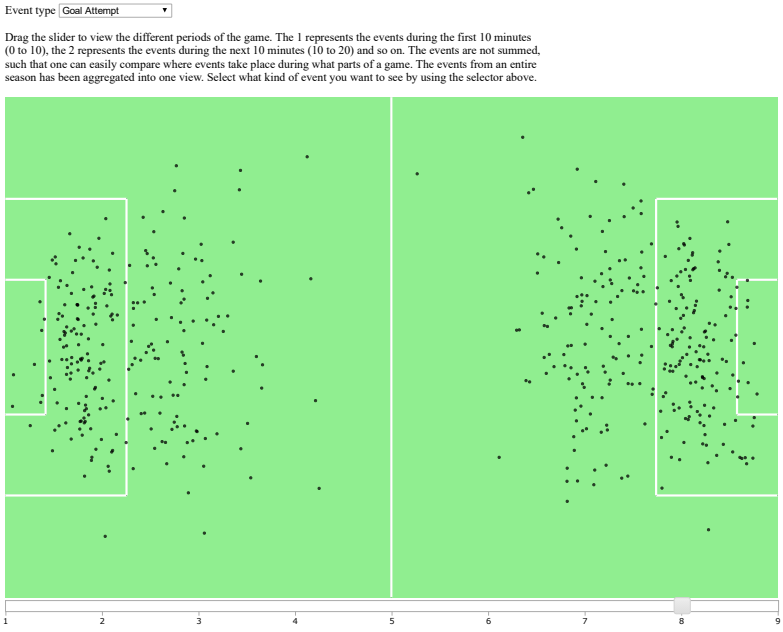

(a) ksdfh

(b) ksdfh

Figure 1: Visualization of event data in intervals during a match. Good passes during the 40 to 50 minutes compared to 50 to 60

0.0.1 What-why-how

What. The data being presented in the visualization is event data from a football season. Event data is things such as a good pass to another player or a goal attempt. From the API we can request events from a match. The event data include the type of event, the time stamp and the coordinates. What is being shown in the visualization is then all of the event data from an entire football season aggregated such that when a user selects good passes and the time interval 0 to 10 minutes, the user will be presented with all of the good passes that has happened in the season during the first 10 minutes.

Why. This visualization could be categorized as being in the exploratory data analysis category, as it allows a user to browse through the data in a very intuitive way, such that the user can get a feeling of how common each event is, and where on the field it is often carried out. But the visualization can also be used if the user have a specific hypothesis that they want to test. The visualization was first build because we had a hypothesis about the goal attempts, where the hypothesis was about how football players might take more risky attempts at goals during the late part of a game, as they perhaps are more desperate to score. To test this specific hypothesis, a user can select goal attempts and then use the slider to see how the goal attempts evolve throughout a match. The result of the hypothesis test is described later in the report.

How. As the data to be shown is spatial and discrete data, it is being presented in a field (which happens to be a football field). The type of mark being used is a point and the visual channel used to encode the mark is spatial position in both vertical and horizontal direction. To control what kind of event a point represents, the user is presented with a drop down menu, where the user can select the type of event. An alternative to this option could be to use the visual channel color, such that goal attempts was blue, good passes was yellow and so on. However, it was decided that only one type of event should be shown at once, as multiple types of events at once could quickly become noisy. Alternatively, a user could select two types and choose the color of them.

Because the resolution of the coordinates is not very high (coordinates are seperated by 1 meter), a lot of points are going to overlap because a lot of events happen at the same place. To avoid that this is going to disturb the meaning of the data, two things has been done. The first thing is that the opacity of each point is lowered a little, meaning that points are actually a bit transparent. This means that if two points are on top of eachother, the point will appear more opaque. The other thing that is done is that there is added or subtracted a small random amount from each point's x and y coordinates. This approach is called "jigger" and is often used when overlapping data are to be visually presented without overlapping. However, there might still be overlaps, as it is actually not checked whether or not the new x and y coordinates of a point collides with another point. One could make an implementation where this is checked, but even if a collision check is done by an array look-up in O(1) time, an implementation could easily result in a worst case time complexity of O(n) where n is the number of events, if the events are laid out in an unfortunate way (for example if they all have the same x and y coordinates, and they are centered around those coordinates, and the algorithm checks for a place to place a point in a circular fashion centered around the original coordinates). The time complexity of the algorithm behind the visualization is O(n), where n is the amount of data points, as the data is read from disk, then the data is binded to the points, and the points' coordinates are updated. Finally, the old data points are removed. This time complexity assumes that the D3 library data binding operation is done in O(n) time. Note

that this is not that good of a time complexity for a dynamic visualization, and sometimes it can be slow depending on how fast the data is loaded from disk or whether it is already cached or not. Alternatively the different events and periods could be preprocessed even further and saved as images that is displayed instead, but this would still require loading the images from the disk, but would make each change of visualization O(1) instead of (n). This approach would however limit the options of what can be done with the data in D3, if one wishes to develop the visualization further.

The data is prepared in R before it is used by D3. The pseudocode for the R code can be seen in the code section. The API that we get our data from can only serve event data from one match at a time. This means that we have to automatically query the API for the event data from all of the different events that is in a season, and then aggregate all of the data into one data frame. From that data frame, the data is sorted into different data frames depending on their event type. The data frames that only contain one event type is then split into 9 different data frames depending on their time in the match. The events during minute 0 to 10 is split into one data frame, the events during 10 to 20 is split into another and so on. These data frames are then saved to seperate files on the disk, where the name of the file is (type of event) + (period in the game) + (.json). 171 different files are produced from this process, which shows that it is important to automate procceses as this. These files are then placed in a directory where the javascript of the visualization can read them. Because we label the event types the same way as the API does, we have a global naming scheme, meaning that the javascript can simply reconstruct the file names and request them from the OS. Another benefit of doing this automatically is that the visualization can quickly be adopted to different kinds of data, and show different seasons etc.

0.0.2 Code

The visualization is made using the D3 library, but the visualization itself is coded entirely from scratch, and is not just a modified example (except for the slider).

The pseudocode for the data processing in R is as follow. Where eventID is the id of a match and touch means an event within the game, for example a good pass.

```
getTouchesAndWriteToFiles() {
  teams = getDataFrame(teams)

  eventIDs = new DataFrame()

  for(teamID in teams.teamID)
    events = getDataFrame(events + teamID)
    eventIDs = combine(eventIDs, events)

  eventIDs = distinct(eventIDs)

  allTouches = new DataFrame()

  for(eventID in eventIDs.IDs)
    touches = getDataFrame(touches + eventID)
    allTouches = combine(allTouches, touches)

  splittedTouches = split(allTouches, touchID)

  for(touchDataFrame in splittedTouches)
```

```
21      for(i = 0 to 8)
22        currentTouches = select(touchDataFrame, minutes >= i*10 AND minutes < (i +
              1)*10)
23        toWrite = toJSON(currentTouches)
24        write(toWrite, touchDataFrame.name + i + ".json")
25  }
```