Department of Computer Science, Maynooth University

Spatiotemporal Analytics in Professional League of Legends

Author
Stephen O' Farrell

Supervisor
Dr. Joseph Timoney

A dissertation submitted in partial fulfilment of the requirements for the MSc in Data Science & Analytics

August 10, 2020

Spatiotemporal Analytics in Professional League of Legends

A study of the potential of spatiotemporal data in competitive League of Legends

Stephen O' Farrell

Abstract

League of Legends is the most popular competitive video game in the world, though advanced analytics are still a rarity in the scene. This paper researches the idea of spatiotemporal data, the in-game locations over time of professional players during a competitive game. To track this data, a program is created which is capable of automatically tracking and recording this data using computer vision. This data is also used to provide analysis on individual games of League of Legends, as well as being used in several proof-of-concept projects on a larger scale. The program is capable of quickly gathering and analysing data that was not readily accessible beforehand, providing a foundation for more advanced analytics in the future.

Contents

Chapter 1

Introduction

1.1 League of Legends

League of Legends, often stylised as League or LoL, is a Multiplayer Online Battle Arena (MOBA) released for the PC in 2009 by Riot Games. The game pits two teams of 5 players against each other, each player controlling one of over 100 different playable characters, known as *champions*. The aim of the game is to destroy the enemy team's defensive *towers* and reach the *nexus* in their base, with the *blue side* playing on the left-hand side of the map against their counterparts, the *red side*. The map consists of three lanes where each team's nexus will send out waves of AI-controlled *minions* in an effort to reach the other team's base. These minions can be killed for gold and experience points, so each team will try to accompany their minions up the lanes while simultaneously taking out the enemy team's minions. Games usually last for around 35 minutes each.

There are five main roles in the game; The player who plays in the northernmost lane is known as the *Top* or *Toplaner* with the player who plays in the middle lane being referred to as the *Mid* or *Midlaner*. The southernmost lane traditionally has two players; an *Attack Damage Carry* (ADC) who will take all the resources of the lane while their *Support* is dedicated to assisting

(a) Regions of the map (b) Standard areas for each role

Figure 1.1: The minimap

the team throughout the game. In between the three lanes is the *jungle*, an area where neutral minions will spawn that can be killed for resources. Each team has a dedicated *Jungler* who will roam around this area, taking the neutral monsters and providing support to all three lanes. The minimap is further detailed in Figure 1.1.

The game quickly grew in popularity as the basic graphics allowed for almost anybody to play the game, no matter the power of their computer. In 2011, League was reported to have 11.5 million players every month. This grew to 67 million per month by 2014 and then broke 100 million by 2017 [1]. The size of this playerbase allowed for a healthy competitive scene to form.

1.2 League of Legends Esports

League of Legends has boasted one of the largest esports scenes since the Season 1 Championship took place in Sweden in 2011. In 2013 the global domestic leagues were introduced, meaning that several different regions around the world played domestically in a league format with large international tournaments at regular intervals, culminating in the annual World Championship

every October.

Since the scene's inception in 2011, it has grown dramatically. The most recent World Championship finals boasted a record of 44 million peak concurrent viewers [2]. Several major leagues have also franchised in recent years, with spots in the bigger leagues being sold for approx. $10m each [3]. This level of investment has led to a big push from esports organisations to improve their infrastructure.

One such improvement is the introdution of data analytics. Tech companies such as Microsoft [4] and SAP [5] have formed partnerships with large esports organisations in an effort to provide analytical support for their League of Legends teams. Though many teams have set up internal data science departments with the aim of gaining a competitive edge through data. Much like in more traditional sports, data science has a large part to play in the future of professional League of Legends.

1.3 Motivation

Data analytics has been revolutionising sports for decades. From the Oakland Athletic's success in baseball as documented in *Moneyball* [6], to Liverpool FC's resurgence in football under the influence of Ian Graham's data-driven model [7]. One of the many types of data used in traditional sports is spatiotemporal data, i.e. the locations of players over time. This is especially prevalent in free-flowing sports such as soccer as opposed to more 'turn-based' sports such as baseball, a sentiment that is mirrored in the real-time nature of a League of Legends game.

Advanced analytics are only just beginning to have an impact on the League of Legends scene, so any innovations will be highly valuable to professional teams. The idea behind this project is to specifically explore this spatiotemporal data in professional League of Legends. Every player and team plays the game differently, so any new technology that can analyse

movement throughout the game could give a strong advantage to any who employ it. Even though League of Legends is a computer game, the data surrounding player's locations is not freely available. Thus, any way to make this data more widely accessible could be a huge milestone for the industry. This is the key focus of the project.

1.4 Objectives

There are three main objectives to this project:

- Build a program to gather the data. There is currently no easy way to gather spatiotemporal data in League of Legends, so a program will have to be made. Such a program should be capable of automating as much of the process as possible, while also being efficient so as to quickly gather large quantities of data.
- The program should be able to provide analysis on any one game the program gathers data from. This will allow for immediate analysis on individual games and can give insights on a smaller scale. Such a feature would prove useful for any teams who wish to utilise the program.
- Perform large-scale analysis on the data. This will be an exploration of the possible uses of this data, attempting to use modern analytics methods to demonstrate several ways of finding patterns in the data. These will serve as proofs-of-concept, showcasing the potential of the field.

1.5 Results

Overall, the project was a resounding success. The program is capable of quickly gathering data and provides an interesting analysis of each individual game. The data is clean and can be used in a variety of ways.

The program for gathering the data is able to efficiently collect it from a basic Youtube playlist. It can be easily installed from the Github repository and has been tested on both Windows and Linux systems. The data itself does not take up much space, but the setup of the output directory means that some manual manipulation is needed to gather large amounts of data into a single database (Though a script to do this is included in the repository). While testing it was found that the program takes between 60 and 120 seconds to process a single video from start to finish, with an average video length of around 35 minutes. Once the playlist and configuration are set up, the process is entirely automatic. As such, the program can gather large amounts of data with ease.

Updating the database of champions or leagues is rather simple too, allowing for the program to support future updates to the champion pool or different overlay dimensions. Several Python scripts are also included in the repository which describe how to update the project in line with any possible changes that Riot Games may make to the game.

The single game analysis is also a very helpful tool to quickly look at what happened in a game. It tracks some key metrics that are not currently widely available for analysis, such as proximity and the regions that certain players tend to play in. It does not provide a comprehensive statistical analysis and does not take any stats other than champion played & locations into account, but that was not the goal of this feature. Being able to quickly analyse key aspects of the game can be invaluable for teams looking to improve and learn from their games.

While the large scale analysis only scraped the surface of what is possible with this data, the projects proved to be excellent proofs of concept. They showcased the potential of gathering this data, rather than a thorough, in-depth exploration of the use cases.

Chapter 2

Background

2.1 Literature Review

Although League of Legends is currently the most popular esport by a significant margin, the analytics scene for the esport is still in its infancy. It proves difficult to find academic literature pertaining to the field. This is compounded by the fact that spatiotemporal analytics is a particularly niche aspect of the industry.

Philip Z. Maymin has done similar work in 2018, in a paper titled *An Open-Sourced Optical Tracking and Advanced eSports Analytics Platform for League of Legends* [8]. This paper describes an analytics platform which tracks an extensive range of data. Statistics such as kills, deaths, gold, experience points and much more are gathered along with the locations of players, but it relies entirely on the spectator client that comes with the game itself. This means that you need access to a digital replay file to gather data. It's also unable to gather data from replays that were saved before the most recent League of Legends update. Riot Games updates the game every couple of weeks, meaning replays quickly become outdated. There has also been no update on this program in the past couple of years.

The project that we create here gathers less extensive data as it only

focuses on the locations of players over time, but it can be easily modified to work with different regions as well as games from any time period. The focus on spatial data means that we're able to track data from any given YouTube video, a flexibility that isn't allowed by programs that use the official spectator client. This allows for teams to scout opponents from other regions with ease, and for data to be collected without relying on any third parties. It also means that teams will be able to study games from previous years.

Looking towards more traditional sports, we can see the impact that spatiotemporal analytics has on the likes of basketball or association football. For example, in *CourtVision: New Visual and Spatial Analytics for the NBA* by Goldsberry [9], spatial data is used to quantify how accurate professional basketball players are at shooting the ball and scoring. In League, the way teams move around key objectives can be a source of useful analysis. Enhancing statistics with spatial context can be a powerful tool.

Dr Goldsberry writes: *"As the NBA becomes more analytically driven, there is an emerging need for spatially informed evaluations that can be effectively communicated to a diverse set of audiences"*. The same can be said for League of Legends. As the professional scene continues to grow, teams will begin to put more focus on collecting and analysing data. It's likely that spatial data will play a key role in this data revolution.

While data analytics has maintained a significant presence in baseball and basketball in the past few decades, they both have oft-occuring and quantifiable measures to base analysis on (runs and points respectively). A sport which has more parallels with League of Legends would be association football, where several smaller actions such as passes, tackles and shots culminate in a larger and rarer reward: goals. League is similar in the sense that several smaller actions such as kills, towers destroyed and gold collected culminate in larger objectives which can lead to a victory. It's difficult to quantify how important each of these is to a victory, so enhancing them with

spatial context can lead to better insights.

Luckily, spatial analytics play a key role in football, especially in recent years. Brooks, Kerr & Guttag [10] used the locations of passes to identify patterns in professional Spanish football teams while in *Spatial and Spatiotemporal Analysis of Soccer* [11], Kim, Kwon & Li track the trajectories of all 22 players and the ball in an effort to find useful information when analysing the tactics of a given team. In this analysis it was found to be possible to use spatiotemporal data in football to quantify the influence a certain attacker has on the opposition's defensive formations, which is the sort of abstract result that would be highly useful if mirrored using League of Legends data. These are but a small sample of the multiple possible use cases of spatial and spatiotemporal data in sports. Bringing this sort of data to the League of Legends scene will make it much easier for similar studies to be run, providing a foundation for further innovations and insights to be gained.

2.2 Software Resources

The project was done entirely in Python, employing the use of several external libraries. Standard data science libraries such as Numpy and Pandas were used for their efficiency in data handling. OpenCV was used for computer vision, as it provided a simple way of dealing with the template matching needed to track images as needed. Plotly was used for the majority of the graphs as the added interactivity is very useful, though Seaborn and Matplotlib featured heavily as well. Youtube_dl and Pafy were also used to make it so that the program could work on a playlist of Youtube videos, rather than a locally stored file. For the proofs-of-concept, Scikit-learn was used to build a Logistic Regression model.

Chapter 3

Design Solution and Overview

3.1 Conceptual Design

3.1.1 Gathering the Data

Any League of Legends broadcast will have a standard layout that we can use to our advantage.

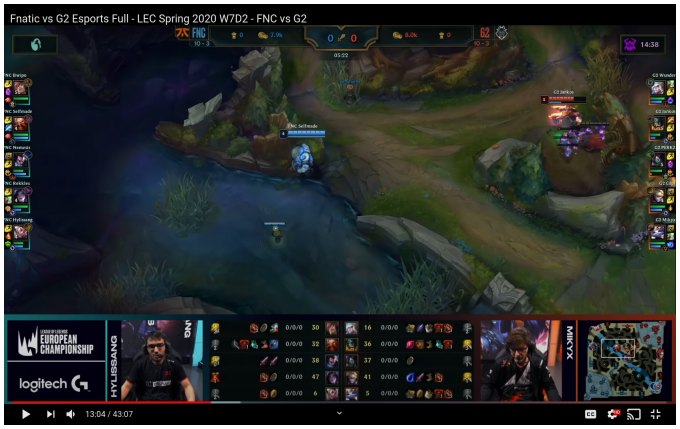


Figure 3.1: In-game screenshot

In the bottom-right corner of the screen is the 'minimap'. This is a birds-

eye view of the entire playable area, showing each champion at all times, as shown in Figure 3.4. We'll use this to track the players throughout the game. We can also see all 10 champions split between the two sidebars, one for the blue side and one for the red side. Each portrait corresponds to a given position, with the topmost portrait showing the Toplaner, followed respectively by the Jungler, Midlaner, AD Carry and then Support. At the start of the game, this can be used to accurately identify which champions are being played as well as which position they're playing as, due to the relatively large dimensions of these portraits. The top-right corner also shows the countdown before an epic monster spawns in the game, which will always happen at the 20 minute mark. This countdown can be used to track the in-game timer more effectively than the traditional timer, once again due to the increased size. Finally, there is a small icon at the very top of the screen which is only present in the spectator overlay, this will only show when the game is live and in-game. This means every frame that we analyse will be live and in-game. These key areas are highlighted in Figure 3.2

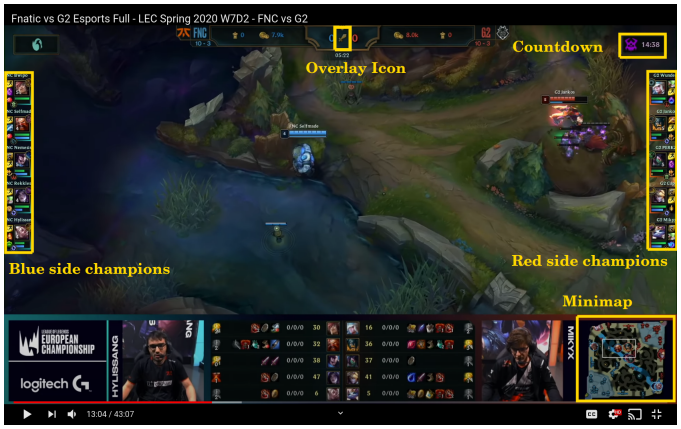


Figure 3.2: In-game screenshot with key elements highlighted

Identifying champions

We'll use OpenCV's template matching to identify these elements of the overlay. Template matching is a very common computer vision technique where the algorithm attempts to spot a provided template in an image. For example, we save a picture of *Akali's* portrait in the sidebar. We can then use this image as a template and see if it can be found in a frame of our target video. If the program can find Akali's portrait in the frame, then it determines that Akali is playing the position corresponding to that particular position in the sidebar. This sidebar is shown in Figure 3.3 below.

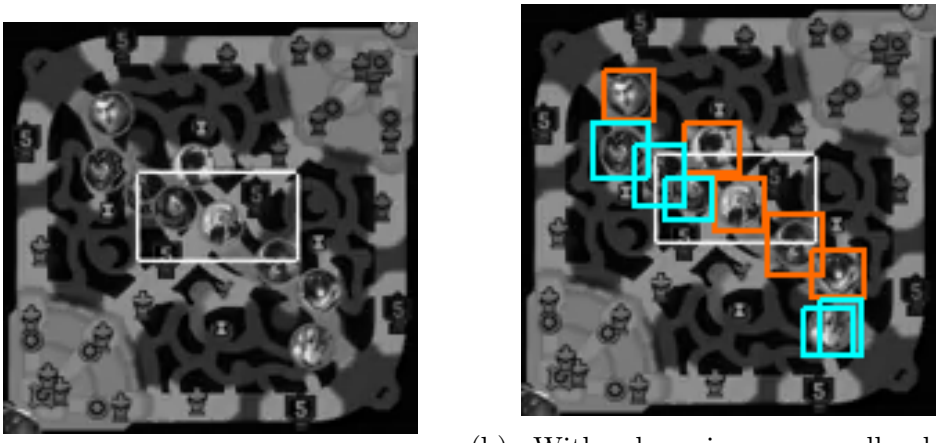
Figure 3.3: The blue side sidebar, with the champions and positions manually labelled

Identifying live footage

As you would expect in a professional broadcast, not every frame of the video shows live footage of the game. There will always be significant sections of the video that we cannot use, such as analyst desk segments, action replays, in-game pauses, etc. Naturally, these frames should not be used to track

positions. The easiest way to do this is to look at the overlay and use template matching to detect whether the small sword icon is showing. This part of the overlay is generally hidden when the frame doesn't show live footage, meaning we can easily use that to check whether we should consider this frame or not.

Tracking positions



(a) No highlighting

(b) With champions manually highlighted

Figure 3.4: Tracking champions from the in-game minimap

Once the program knows which champions to follow, it can then begin following each champion. The minimap will show each champion throughout the game, as in Figure 3.4. By template matching against pictures of each champion in the game, their locations over time can be tracked and saved.

Syncing the times

While tracking the locations is the main goal of the project, the data is almost useless if there is no corresponding timestamp for each datapoint. Simply tracking each frame as a second would be highly inaccurate as any interruption of the broadcast would desync the time. This is a side effect of

the program only considering live footage. The game is not paused during a replay, so ignoring these frames and failing to account for them in the timer will lead to the timer being incorrect for any frames after the replay. Instead, the program reads the time for every frame it analyses.

Every broadcast will have the in-game timer as part of the overlay. This could be used to synchronise the datapoints with their times, but it is rather small. Fortunately, there's a better option in the 'Baron Nashor Countdown' timer located in the top right of the overlay, as shown in Figures 3.2 and 3.5. This counts down to the 20 minute mark, when the epic monster known as *Baron Nashor* spawns. This timer can be read for each frame and use this to track the in-game time.

Figure 3.5: The countdown timer showing 19 minutes and 21 seconds until the Baron Nashor spawns, which corresponds to an in-game timer of 39 seconds

The only issue is that this won't track any data past the first 20 minutes. For the purposes of this project, however, the first 20 minutes are the most important as this is where each role is most distinct from the rest. After the 20 minute mark, the game becomes much more fluid and teams move around in large groups more often. As such, we'll only take these first 20 minutes into consideration for the project. Any who wish to extend the range of data should be able to change the program to read from the in-game timer as opposed to the countdown timer described here.

Input

Each game input to the program will be a standard video, the easiest way to quickly go about this is to use a Youtube playlist as input. There are already thousands of broadcast replays on Youtube, so it makes sense to take advantage of this. Using the Pafy and Youtube_dl Python libraries in conjunction with OpenCV make this trivial.

The main issue is the difference in overlay between different leagues. For example, the European League (LEC) has slightly different dimensions in their overlay to the North American League (LCS), so the program will have to account for this disparity across multiple leagues.

Interpolation

As can be seen in figure 3.4, there are times when champions are directly on top of each other on the minimap. The template matching will not be able to spot every champion at every point of the game, so some datapoints will inevitably be missing. Naive interpolation methods wouldn't be appropriate here, as it could lead to large, inaccurate paths being drawn between points.

As such, a custom interpolation will have to be done for the points; one which will account for the type of paths that players will generally walk. It will also have to account for the players around them, as being near another player will lead to missing data. Once these edge cases have been accounted for, then a more traditional interpolation can be used to predict the rest of the missing data.

Accounting for Different Leagues

Most professional broadcasts are run by separate teams, which often leads to creative differences in the presentation of the broadcast. One important difference is the spectator overlay, a crucial part of this program as it uses these dimensions often. For example, when tracking the champions the program

will focus on the minimap by cropping the image to the specific dimensions of this minimap. Any change to these dimensions will heavily reduce the accuracy, as the crop will lead to an incomplete map image. Unfortunately this means that every league the program supports will have to use custom dimensions that the program will call on. This consists of four values, representing the pixel locations of the four corners of the minimap. Before attempting to track the champion, the program will crop the image using these values. The resulting cropped image will then be the minimap for that league.

There is also the issue of a broadcaster completely changing the overlay. In Summer 2020, the North American league's overlay removed the sword icon needed to find live footage and replaced it with the league's official logo. To make matters worse, they changed the logo shown halfway through the tournament. This had to be circumvented by manually changing the program to check for either of these new logos rather than the sword icon anytime a game from this particular tournament is used.

3.1.2 Single Game Analysis

Level One

At the start of every game, there is the *level one*. This is the period of the game before any minions appear, where the players are free to run around the map without worrying about gathering resources. This lasts for the first 90 seconds of the game and is often the source of early advantages. It is the closest thing that League of Legends has to a 'dead ball situation', akin to a pitch in baseball or a corner kick in football. More often than not, teams will form a defensive formation in an attempt to thwart any offensive moves. However, it is not unusual for a team to aggressively push into enemy territory to get an advantage.

Seeing how teams line up for these first 90 seconds can be very useful as it can show patterns in play that can be abused. The best way to show this

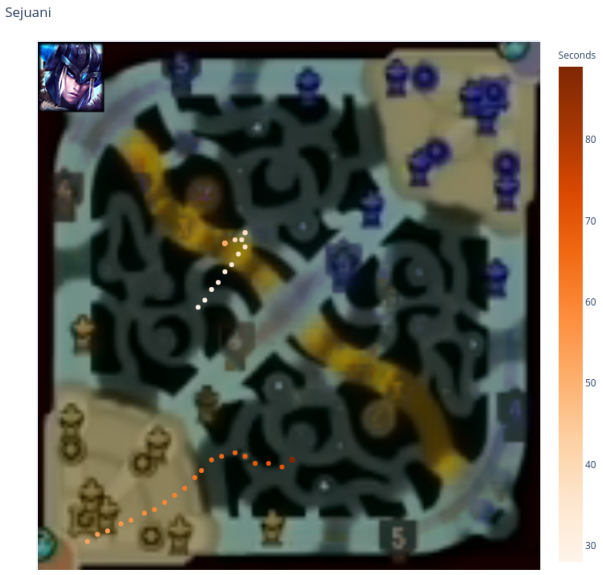
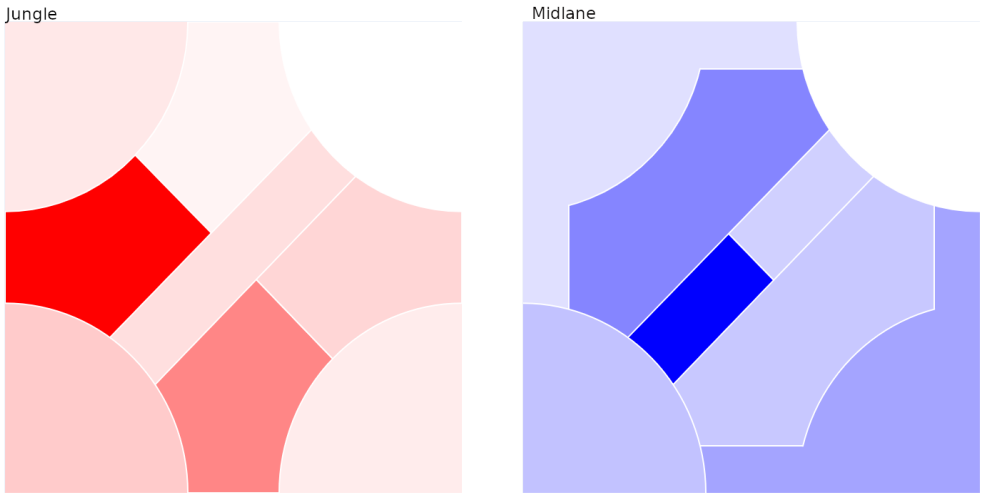


Figure 3.6: Level One path

for a single game is to simply graph the path that each player took. This will allow you to quickly see how each player moved around the map, as shown in Figure 3.6 by the orange and white lines. Note that the player begins at the white dotted line before teleporting back to base and continuing along the orange-red dotted line.

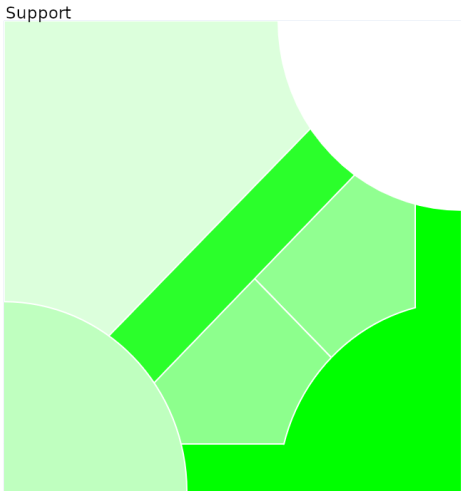
Role-based Maps

How players move around the map is a key focus of this project. To this end, it makes sense to be able to quickly gauge how certain players moved throughout the course of the game. A good way to convey this information is through a choropleth map using custom-made regions. Colouring these regions based on how often the subject is found in them will quickly show the areas they tend to play in. The darker the colour of a given section, the more times the target player was spotted in that area. Thus, it's very easy



(a) Jungle

(b) Midlane



(c) Support

Figure 3.7: Role-based Region maps for the first 8 minutes. Note that all three are for blue side champions, hence why they are never spotted in the red base (top right).

to look at a graph and quickly deduce the play patterns of that particular player in this particular game. Some examples are shown in Figure 3.7.

It's also important to note that tracking certain roles is more useful than others. The AD Carry, for example, generally doesn't have much impact on the flow of the game as they tend to gather resources and let the rest of the team dictate the pace. On the other hand, the jungler is free to roam around the map throughout the game, having a large impact on the first 20 minutes. With this in consideration, it's best to focus on the three roles whose movements have the most influence on the game; Jungler, Support and Midlaner.

However, each role in the game plays completely differently to the rest. This means that focusing on the same regions for every role won't be particularly useful. For example, the midlaner pushing past the halfway point of the midlane is important as it shows they were playing aggressively. On the other hand, it's not important which half of the lane a support was in when roaming to the midlane, as the more important information is that the support was in midlane at all. Splitting up the regions too much will give rise to the *Modifiable Areal Unit Problem* [12], a phenomenon wherein splitting up the regions of an area class map will distort the message the map is trying to send. By using custom regions focused on the important areas for each key role, the program can give a better picture of what happened during the game.

Another thing to consider is the different time periods of the game. Getting data for the entire first 20 minutes isn't particularly useful as it doesn't show how those trends change throughout the game. Luckily, there are significant breakpoints in the game that we can use. The *Rift Herald*, a major early objective, spawns at 8 minutes. Then at 14 minutes the *Tower Plates*, an early-game reward for damaging towers early, are no longer obtainable. Teams will often base their play around these objectives, thus it makes sense to split the maps based on these time periods. The final result will then be 3

maps for 3 roles; showing the Jungler, Support & Midlaner's positions from 0-8, 8-14 and 14-20 minutes.

Proximity

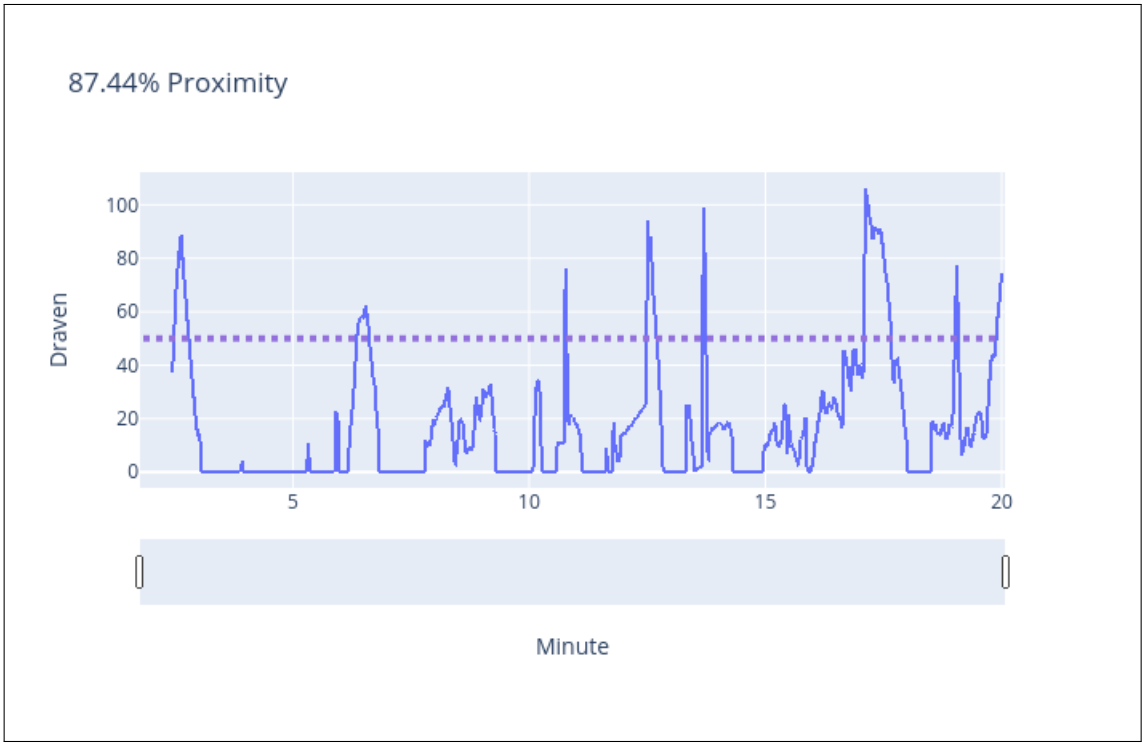


Figure 3.8: Proximity graph showing the distances between a Support player and their allied ADC (playing as 'Draven'). The dashed line shows the 'close proximity' threshold; Any time players are within 50 units of each other, they are considered to be in each other's proximity

While the locations of players throughout the game in isolation is undoubtedly useful, it's also important to consider how they move in relation to their teammates. Seeing when players are in close proximity to each other can convey a lot about playstyles. It's also useful to aggregate this data to a single number in order to get an idea of how much one player tends to play around another.

A good way to show this is to graph the distances between any two players over time, as shown in Figure 3.8. This will allow you to easily see when these two players stick together and when they separate. To get the aggregate number, an arbitrary distance is chosen as being in close proximity. The percentage of time that the two players spend within that distance of each other can be considered their proximity, with high proximity values indicating a large amount of time spent in each other's vicinity. It can also be split into different time periods, same as with the region maps.

3.1.3 Large-Scale Analysis

Level One Patterns

While the program is capable of showing the level one paths of any individual game, building a model that could analyse the general formation of a team over multiple games would be a valuable tool. Getting an early advantage can be highly useful as the pace of the game can be changed drastically by this.

To spot such a pattern, a model can be built that focuses on the idea of finding an enemy champion in isolation. That is to say; if a team was to make an aggressive foray into the enemy side of the map, where and when could they find a given player. This is useful as it allows you to plan around this information beforehand. Knowing where and when you're most likely to find a given player on their own allows you to target that specific player and make a play with a higher percentage chance of putting them at a disadvantage, graphs demonstrating some level one patterns can be seen in Figures 3.9 & 3.10.

To calculate this, a predictive model can be built with coordinates & time as input and role as output. The idea is to take a position and time as input and return the role most likely to be found there. The model would be trained on data collected for a given team and a prediction would be made

for every point on the map, keeping time constant. By colouring each pixel with the champion most likely to be found there, the map effectively portrays the formation of the team. By changing the time and creating a second map, the formation changes over time can be seen.

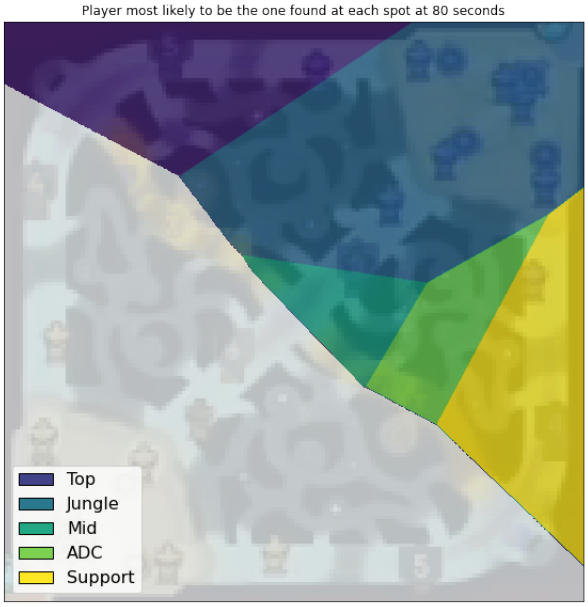
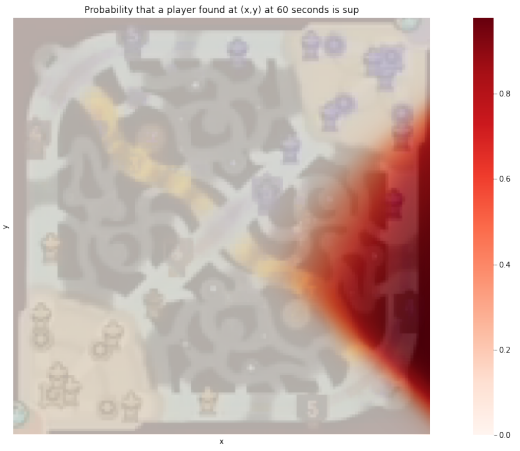
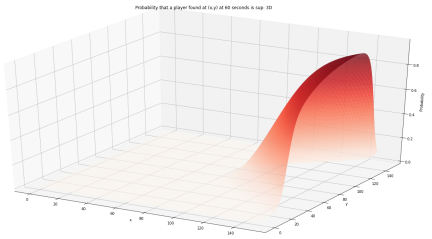


Figure 3.9: Graph showing the red side level one tendencies of G2 Esports at 80 seconds for the 2020 Spring season, as explained in Section 4.2.

While this will show the player most likely to be found at a given position at a given time, it doesn't fully capture how likely they are to be on their own. If there are two players that are near 50% probability, then you are unlikely to find them on their own. Whereas if the model is 90% sure about a prediction, you have a much better chance of finding that person on their own, making it much easier to force an advantageous fight. Thus, the individual probabilities should also be graphed. This will allow you to see the places where you are most likely to find a player on their own.



(a) 2D



(b) 3D

Figure 3.10: Red Side probability graphs of G2's Mihael 'Mikyx' Mehle for the 2020 Spring Season. The more intense the colour of a point is, the more likely Mikyx is to be the one found at this point at the 60 second mark

Proximity Patterns

As mentioned in the Section 3.1.2(c), the distance between players is a powerful statistic as it captures the patterns of players in relation to their teammates. It could prove very useful to gather a significant amount of proximity data and analyse it to find the playstyles of certain teams.

To achieve this, a large sample size of location data can be gathered and the proximities for each team calculated from this. Naturally, this is a relatively large computing task so it's better to focus only on a select few roles. For the purposes of this study we'll concern ourselves only with the supports & junglers of a team, specifically how these two interact with their respective teammates. We're essentially trying to find out how often two players stand near each other throughout the game.

A potentially useful source of information lies in the correlation between jungler-midlaner proximity and winrate of teams. These two roles often play in tandem, so we'll attempt to quantify how much focus successful teams put

on this relationship. That is to say, are teams that have a jungler who plays heavily around midlane more successful? We can also check jungle-support proximity against winrate, in an effort to see if either strategy is better than the other.

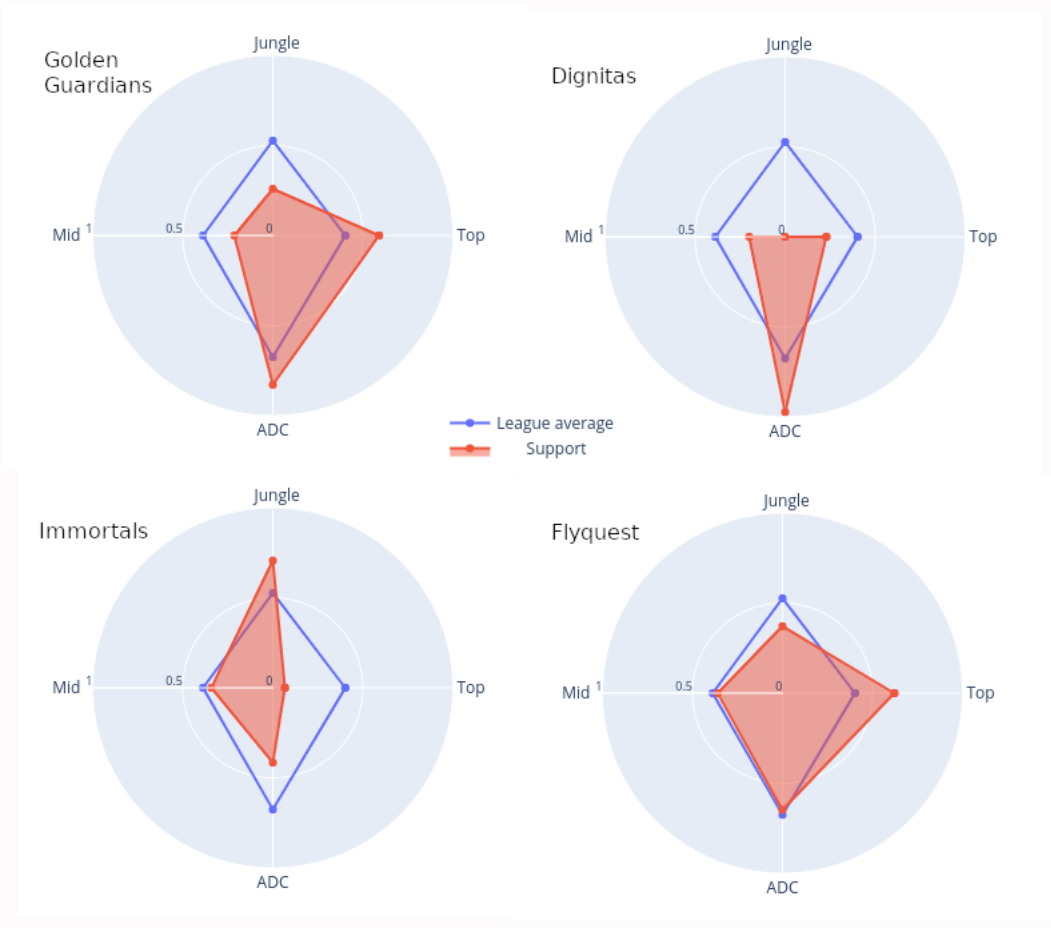


Figure 3.11: Radar graphs showing how 4 different teams' support proximities compares to the NA League of Legends Championship Series (NA LCS) average

Proximities can also be used to get an idea of a certain player's playstyle. It can be used to see how one player's tendencies compare to the rest of the league and can effectively portray the focus they put on each of their

teammates. A good way to show this is with a radar/star plot, which we'll employ here as shown in Figure 3.11. By getting the average proximities for each support in a given league and comparing them with their counterparts, we can plot these values on a radar graph. The process can then be repeated for junglers. This will allow for quick analysis of playstyles.

Pressure Patterns

While each team has their own side of the map, it is common for players to venture into the opposing half to make an aggressive play. Passive players will often sit back and play safer, but others are not so willing. It is especially important for junglers to figure out when to push forward and apply pressure, as they are hidden in their jungle for a significant part of the game. Figuring out how teams play around these aggressive forays can be highly beneficial for punishing reckless aggression.

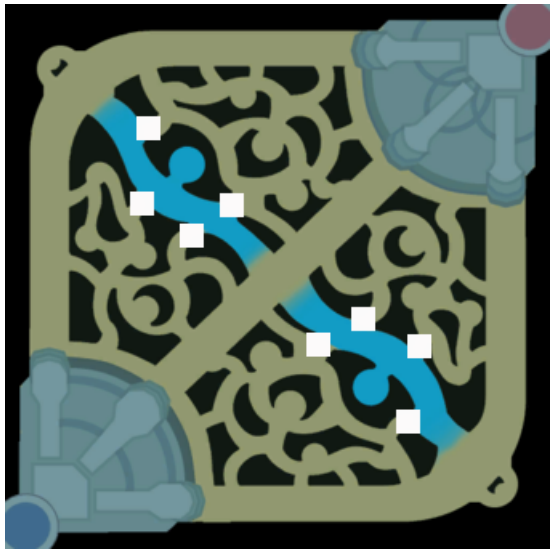


Figure 3.12: The 8 exits of the jungle, marked in white

There are four main exits from the jungle on each side, along with the three lanes, as highlighted in Figure 3.12. Anytime the jungler passes through

these exits, they exert pressure. Figuring out which exits they tend to take, as well as how their team plays around these times, can be used to gain valuable insights on an opposing team's playstyle.

For this demonstration, we'll gather a large amount of location data and iterate through the jungler's locations in each game. Anytime they move through one of these exits, we'll save the locations of all 5 players on that team. When they're all saved, we can plot these using a density plot. The idea behind this is to analyse the states of each lane anytime the jungler leaves their side of the map. If it can be shown that players will sit back and play passively while their jungler pushes forward, it means that there will be a slight delay in lanes assisting their jungler due to this desync of playstyles. This is a weakness that can be exploited.

3.2 Actual Implementation

3.2.1 Gathering the Data

Input

Inputting the videos is rather simple. A Youtube playlist url is needed, which the Pafy library can use to grab the video IDs. OpenCV can then use this video ID to stream the video and analyse the frames from there.

To avoid the issue of different leagues having different overlays, the desired league is also needed as input. They can also decide how many videos to skip. This input format is described in Listing 3.1

```
# Change this to get different leagues
league = "lec"

# Change this url to get different videos
playlist_url = "https://www.youtube.com/playlist?list=..."
```

```
# Change this to skip the first n videos of the playlist
videos_to_skip = 0
```

Listing 3.1: Customising the input

Identifying Champions

As mentioned, OpenCV's template matching feature will be used to identify the champions at the start of the game. The sidebars of the in-game overlay show the ten champions being played, so the idea is to focus on each individual portrait in the frame and match against a database of all known portraits.

Once we get a frame at the start of the game, each role can be identified by cropping this frame ten times; one for each role. This is important as template matching on a smaller image will be significantly faster than on a large image. Cropping an image before template matching will be done wherever possible by the program. On top of this, the images will be converted to grayscale. This will further optimise the runtime of the program. Each crop can be checked against the database of known portraits and the champion that best matches the image will be considered identified.

OpenCV's `matchTemplate()` function is used to check each crop of the frame against the database, just as in Figure 3.13. For every possible champion, `matchTemplate()` will return a percentage value, denoting the probability that the champion image is in the given crop. The champion with the highest percentage value will be recorded along with the role that the crop corresponds to. It will attempt to find all blue side champions before looking for their red side counterparts. The program needs to be at least 65% certain that a champion is in the game for them to be identified and then tracked.

If the program finds too few champions, the problem is often with the frame itself. It may be caused by a graphic overlay, or the program thinking the game started earlier than it actually did. Thus, the best solution to this

is to skip one second of video and try again until at least five champions per side can be found.

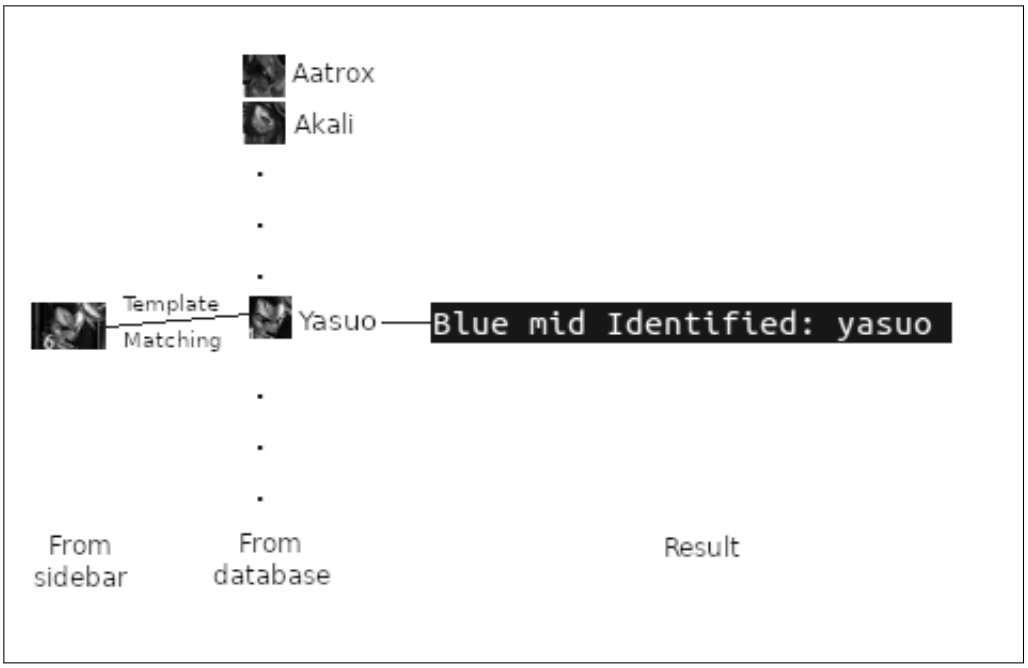


Figure 3.13: Identifying the blue side Midlaner

Once the program has found five champions per team, it can move on to tracking them.

Identifying Live Footage

This will work similarly to the champion identification. As mentioned previously, unless the overlay is customised by the broadcast team, there will always be a little sword icon in the top-centre of the screen. This icon will only show up when the game is being broadcast live, which means it can be used to identify the frames that we want to use. Once again, the `matchTemplate()` function can be used here with a pre-saved image of the sword as a template.

More often than not, the start of a video will not be live footage. As such, the program will check every second at the start of the video until the sword icon appears. Once it does, the champions can be identified and the process can begin. Once again, the frames are cropped and colours are removed before template matching in order to improve computational efficiency.

As the game continues on from here, every frame that we analyse will firstly be checked for the sword icon. This means that any interruptions will be ignored and the data will solely be gathered on the frames that show live footage.

Tracking Positions

As with the previous sections, `matchTemplate()` is used to track the champions throughout the game. At this point, the program has identified the ten champions being played. It takes these ten champions and grabs their in-game portraits from another database of portraits (slightly different images to the sidebar portraits, so a separate database is needed). These images are used as templates and matched against the minimap, as featured in figure 3.14. For every champion portrait template that the program checks, `matchTemplate()` will return a percentage for every point on the minimap denoting how likely it is that the template is found here. If the highest percentage is above a certain threshold, then the corresponding point is saved.

Each point is an x,y coordinate representing a location on the minimap, with the top left corner as the origin. The templates are all 14x14-pixel images, meaning the program adds seven to each coordinate when the point is saved. This ensures that we track the centre of the image, rather than the top corner. If the champion cannot be found by the program, it saves the point as a missing datapoint to be interpolated later.

Each point is saved to a Python dictionary. This dictionary holds a list of every point for every champion tracked and grows as each new frame is analysed. This dictionary is then converted into a Pandas dataframe for

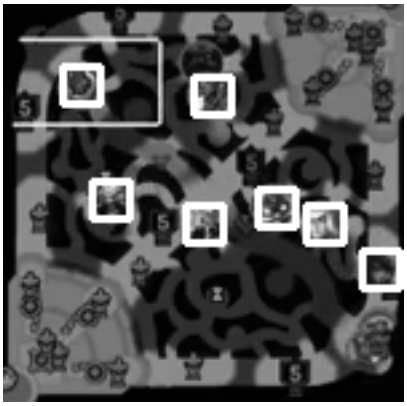


Figure 3.14: Minimap with champions highlighted by program, note the missing champions that will have to be interpolated

easier data wrangling.

Syncing the Times

Syncing the points gathered with the in-game timer should be trivial, as we can read the time from the countdown in the top right corner. However, the traditional method of reading text from an image, Optical Character Recognition (OCR), was found to have a surprisingly low accuracy on the images. It is assumed that this is due to the low pixel count combined with a difficulty in whitelisting digits. The Python Tesseract library was used but it would consistently find letters and rarely identify the correct digits, even when the negative of the image was used as input. It proved difficult to limit the program to identifying digits and it was decided not to use OCR. In lieu of traditional OCR, a solution using template matching was formulated.

Every digit is saved as an image and template matched against the timer from the video. Once the digits in the image are identified, they are sorted according to their horizontal position. This means that the timer will be in the correct order before the program checks to make sure it's valid. One common mistake is that the program will identify the same digit twice, '17:34'

could be read as '177:34', so a small function is needed to rectify these. Any times that are too short or too long after this check are deemed invalid and changed to '9999', allowing these points to be ignored during analysis.

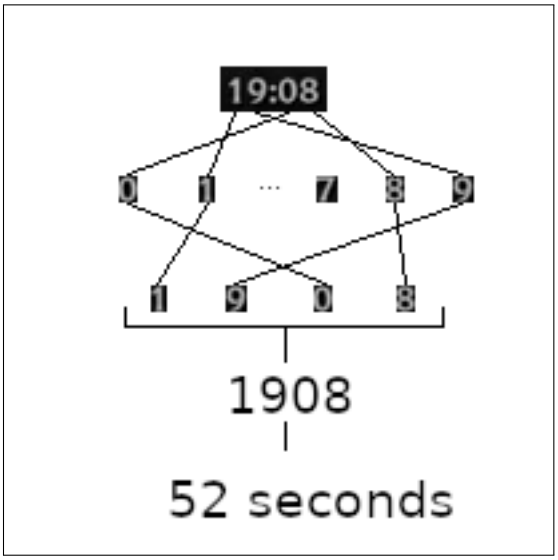


Figure 3.15: Process of reading the timer. The digits from the timer are template matched against pre-saved images of each digit

Once the time is checked, it is converted to the corresponding seconds value. For example, '19:30' implies that there are nineteen minutes and thirty seconds until the 20 minute mark, so this corresponds to an in-game time of 30 seconds. This value is then added to another list, one which will match up with the lists of points in our dictionary from the previous section. This process is illustrated by Figure 3.15.

Interpolation

Once the data has been gathered, the program has to interpolate as many missing points as possible. Unfortunately, a naive implementation will lead to highly inaccurate data, meaning the program must interpolate the data while considering the context of each point.

The main cause for missing data is two players standing right next to each other. Their portraits overlap on the minimap when this happens, which means that the program can only see one of them at a time. Thus, the first thing to check when dealing with a significant chunk of missing data is the allied champions that were in their immediate vicinity. If a given champion disappears while standing right next to an ally then reappears five seconds later next to the same ally, it is fair to assume that they were near each other for the duration of those five seconds. When it comes across a large period of missing data, the program will check the distances between the player and their allies before and after disappearing. If any ally is found to be very close to the champion before and after they disappeared, then the missing data is replaced with the ally's steps. A slightly modified version will also be used for the beginning and end of the data stream, as the program only needs to check for nearby allies at one position in these cases.

Another major cause is when a player returns to their base or dies. At any point in a League of Legends game, you can channel a *recall* spell to return to your base. As the bases are in the very corners of the map, it is often difficult for the program to see enough of their portrait to match the template. If a player dies, then their portrait is not shown on the map and they respawn in the base. To counteract this, the program will check the first place a player reappears after disappearing for a significant length of time. If that point is within their base, then they are determined to have either recalled or died. The missing data is then interpolated with their last known location before returning to base, as shown in Listing 3.2.

```
# If first location after disappearing is in the base
if(norm(np.array(point) - np.array([circle_x,circle_y])) < radius):
   # Fill in with the location just before disappearing
   x2 = pd.concat([pd.Series(x2[:k-count]),
                   pd.Series([x2[k-count-1]]*count),
```

```
                    pd.Series(x2[k:])], ignore_index = True)
```

Listing 3.2: Interpolating when champion reappears in their base

Now that the data has been mostly interpolated using the custom interpolation, the remaining missing data can be interpolated using a standard implementation from the Pandas library.

Accounting for Different Leagues

To account for each league, a measurement of the minimap location is taken for each respective broadcast. As described above, this consists of four values denoting the four corners of the minimap. These are stored in a dictionary along with the league they refer to. Selecting the league (as in Listing 3.1) will tell the program which dimensions to take. The four values will be stored as variables and used anytime the image needs to be cropped to the map. Fortunately there are a number of leagues that use the same map dimensions, which allows for multiple leagues to share the same measurements.

For the graphs in the single game analysis section, these dimensions won't be needed. Each league has a map with slightly different dimensions to the rest, which means that the graphs for one league may not be fully accurate if the exact same dimensions are used for another. For instance, the European League's map is a 150x149-pixel image. The map used by the Chinese & Korean league's broadcast, on the other hand, is a 172x170-pixel image. If graphs or calculations based on one map are directly mapped to the other, there will naturally be a loss of accuracy. To avoid this, any and all calculations are based on the map size, as opposed to any fixed values.

3.2.2 Single Game Analysis

Level One

Plotting the level one paths as in Section 3.1.2(a) is simple. The program takes every point that occurs before the 90 second mark and plots their (x,y) coordinates on a scatterplot. By adding a picture of the minimap as the background of the plot, the positions can be matched up with their actual locations on the map. This will clearly and effectively show the path that any given player took. The program uses the Plotly graphics library to build these graphs, as the interactivity that Plotly graphs offer is a powerful tool for visual analysis.

Simply plotting the points is not ideal, however, as it does not capture the movement of players temporally. Seeing a path of monochrome dots shows a path, but it doesn't show which direction the path is in, nor does it show what time the player was at any given point. Plotly graphs can be easily interacted with to show this information, but at first glance it is not obvious. A simple solution is to colour the path based on the time of each point. So points taken at the start of the game (usually around 20 seconds in) will be coloured white, but will slowly turn red as the points continue from there, as demonstrated in Figure 3.6. This means that it will be very easy to quickly register the path that the player took. Seeing these patterns can confer a tactical advantage as it can identify any non-standard routes that were taken. It can also indicate the timings of opposition movement. For example, if the enemy jungler moves from his defensive position at the 65 second mark then you may have had an opportunity to make an aggressive invade at the 70 second mark. Seeing the paths so clearly laid out can show these abusable patterns for any future games.

To account for different leagues, the size of the graph is automatically set to conform to the height and width of the corresponding minimap image. A graph is created for each champion in the game and added to the output

directory as a separate HTML file.

Role-based Maps

(a) The point to be classified

(b) Checks if point is near the corners

(c) Checks if point is above the midlane's upper border

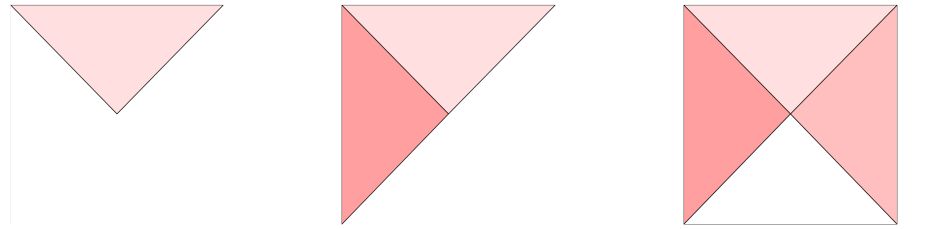
(d) Checks if point is above the lower border

Figure 3.16: Process of classifying a point that lies in the midlane. After the point is deemed to be above the lower border of the midlane, it is determined to be in the midlane and classified as such

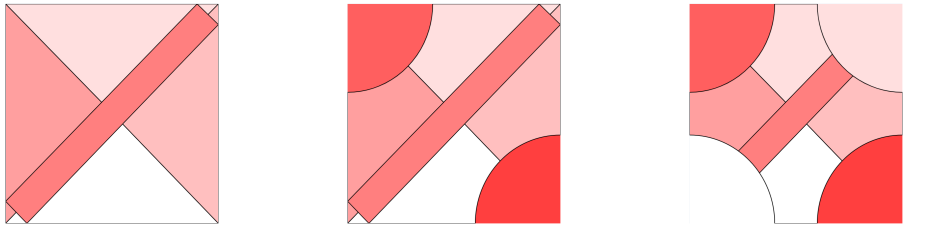
To create the role-based maps described in Section 3.1.2(b), the first consideration is the regions that the program focuses on for each role. All three 'key' roles play completely differently, so the region borders need to reflect that.

For the jungler, the early game usually consists of running around the area

between lanes (the jungle) and assisting the lanes themselves. Therefore, the important information to track is which quadrant of the jungle they're in or which lane they're in. The jungler being in the base should also be a consideration, as standing in the base has little information to be gathered. Implementing this isn't particularly tough.



(a) Red side upper (b) Blue side upper (c) Lower half of jungle



(d) Midlane (e) Sidelanes (f) Bases

Figure 3.17: Creation of a Jungler Region map

The program will first check the distance from the point to each corner, if this distance is below a certain threshold it can be deduced that the point is either in the base or in a sidelane (depending on which corner it lies in). If not, a straight line is then drawn just above and parallel to the midlane, the 'upper border' of the lane. If the point is above this line it is considered in the northern half of the jungle. Another line is then drawn perpendicular to the midlane, the same point being above this line means that it is on the red side of the northern half (red side red buff, as it's called), while it is on the blue side if the point lies below the line. If the point lies below the upper border previously drawn, the midlane's 'lower border' is drawn. Any points

that lie above this second line are attributed to the midlane. Any points that lie below the line are considered to be in the southern half of the jungle and another perpendicular line is drawn to check whether they are classified as being on the red or the blue side. All areas of the map have now been accounted for. A quick example of this can be found in Figure 3.16, where a point in the midlane is classified using this algorithm.

This method means that every point can quickly be classified, and the regions can very easily be graphed on a map. The number of points attributed to each of the regions is then scaled to better capture the relationship between each region and how often the player is found there. These scaled numbers will provide the reference for the colours of the region maps.

The same methodology will be used for the support and midlane maps, with different region borders which can be seen in Figure 3.7. For example, the midlaner stays in the centre of the map, so it is more important to see which side they tend to move towards throughout the game as opposed to which quadrant of the jungle. It also takes into account whether or not the midlaner pushes past the midway point of the lane, as this is a good indicator of aggressive play. The support plays in the southernmost lane, but tends to move around the map a lot as supports generally don't gather many resources. To this end, the regions focus more on where the support may go when providing assistance to their team. The method of using linear equations to classify the points, like in Figure 3.16, is repeated but with different (but similar) equations for each role.

To graph the regions, the Plotly package will once again be used. Using the equations of the lines, shapes bounded by these equations can be stacked on top of each other with the use of an SVG path. Each shape will represent one region, and are coloured based on the scaled value corresponding to that region. This process is illustrated in Figure 3.17. The darker the colour of an area is, the more often the target player is spotted in this area. This means that looking at the graph will allow you to quickly deduce the most

commonly traversed regions for the given player.

Splitting the maps into the key time periods is trivial. The data is synced up with the timer, as in the *Syncing the Timer* section. Thus, the datapoints that pertain to a given time period can be classified and graphed separately.

Proximity

To graph the proximity, first the program must calculate the distance between points. For this, the `linalg.norm()` function of the Numpy library can be used. This will return the Euclidean distance between any two points. The dataframe is then iterated through, getting the distance between players for every second, casting these distances to another data structure. Once each distance has been found, they are plotted on a time-series graph using Plotly's `plotly_express.line()`.

To aggregate this data to a single *proximity* measure, an arbitrary distance is chosen as being in 'close proximity'. Anytime the distance between two players is smaller than this value, they are considered to be in close proximity to each other and a counter is incremented. Dividing the final counter value by the total number of points gives a percentage that represents the percentage of time the two spend in close proximity to one another. Higher values represent players who stick together for a significant amount of time, vice versa for lower values.

3.2.3 Large-Scale Analysis

Level One Patterns

Once the data has been gathered, the process isn't particularly difficult. The data is split into blue and red side and a model is built for each. The datapoints were limited to those before the 90 second mark, as this is when the 'level one' period ends.

A logistic regression model from the scikit-learn Python package is used

to make the predictions. That is to say, a model is built such that the probability of a given point and time being attributed to a certain role can be represented as:

$$Probability(Y_i = Role_j \mid time_i, x_i, y_i) = \frac{e^{\beta_{j,0}+\beta_{j,1}time_i+\beta_{j,2}x_i+\beta_{j,3}y_i}}{1+\sum_{k=1}^{4} e^{\beta_{k,0}+\beta_{k,1}time_i+\beta_{k,2}x_i+\beta_{k,3}y_i}}$$

With the model attempting to predict $role_j$ given $time_i$ and x & y coordinates x_i & y_i. The $\beta_{j,n}$s are the n-th parameters of the model that correspond to each $role_j$, which will be found by training the model on the data. The $role_j$ with the highest probability value will be the prediction for that particular $(time_i, x_i, y_i)$.

```
x_mesh, y_mesh = np.meshgrid(np.linspace(0,150, 700),
                             np.linspace(0,149, 700))

timer = 60

time_mesh = np.zeros_like(x_mesh)
time_mesh[:,:] = timer

grid_predictor_vars = np.array([time_mesh.ravel(),
                                x_mesh.ravel(),
                                y_mesh.ravel()]).T

preds = model.predict(grid_predictor_vars)
```

Listing 3.3: Predicting across every point in the map

This model is trained on the collected data and used to predict every point in a mesh that spans the size of the map while keeping time constant, using the code in Listing 3.3. This will give a prediction that covers the

entire map.

The set of predictions will take a form similar to that of a raster image, with a value for every pixel of the map. We can use the Matplotlib library to plot this, colouring each pixel with the role predicted. The `imshow()` function fulfils this role, taking the 2D array of pixel colours and returning a graph just like Figure 3.9. The minimap is then used as the background of the image, for clarity. By only including the roles for the side the model is focusing on, be it blue or red side, the map becomes very clear.

```
# Grabbing probabilities instead of predictions
preds = model.predict_proba(grid_predictor_vars)

# 2 denotes the 3rd value, meaning blue side midlaner
hp_mesh = preds.T[2].reshape(x_mesh.shape)
```

Listing 3.4: Getting prediction probabilities for blue side midlaner

The process is essentially the same for the individual maps. Instead of getting a 2D array of predictions, we get a 2D array of probabilities for a given role, as shown in Listing 3.4. So for each point in the array, the probability of it being attributed to a chosen role is returned instead of the prediction itself. This can then be graphed using the same `imshow()` function for a 2D graph, as well as a 3D representation using the `mpl_toolkits.mplot3d.axes3d.Axes3D()` function from Matplotlib. This results in the graph shown in Figure 3.10.

Using the model, two functions were created; `red_levelonefull(timer)` and `red_leveloneplayer(timer, role)`. The former creates a graph which shows the areas where you are most likely to find each player at a given time, as in Figure 3.9, while the latter shows how likely it is for a player you find at a given time to be the desired role, as shown in Figure 3.10. To put it another way, if you were to find someone at a given location at a given time,

the chance that this someone is your target role is denoted by the intensity of the colour at that point in the graph. The darker the colour, the more certain the model is that any person you find is the role you used as input. So in figure 3.10, the model is fairly certain that if you happened to find someone on the right-hand side of the minimap, that it would be the support.

Proximity patterns

For this proof-of-concept, the implementation is fairly trivial once the data is collected using the program. The proximities are gathered between supports/junglers and their 4 teammates, a total of 8 data entries (7 unique entries) per team per game. The data was gathered from the North American League of Legends Championship Series (NA LCS) Summer split 2020. A total of 75 games were analysed and data gathered from them, for a total of 45 video hours. The program took just over 2 hours to gather the data. The winners for each game are manually added to the database and the winrates for each team are calculated with a simple formula:

$$\frac{games\_won}{games\_played}$$

The average proximities between the jungler and midlaner are then taken for each team, for the n games that involve that team:

$$Average\_proximity(Team_i, mid) = \frac{\sum_{k=1}^{n} proximity_{jgl_{k,i}, mid_{k,i}}}{n}$$

With $proximity_{jgl_{k,i}, mid_{k,i}}$ denoting the proximity percentage between the jungler and midlaner for $Team_i$ in $game_k$. This will return a single percentage value denoting the average proximity between the two players for the team in question. This is repeated for every team, as well as for the jungler - support proximities. These values are then plotted on separate Plotly scatterplots

to see if there is a significant correlation between these average proximities and a team's winrate. If there is a significant correlation, then it could be deduced that one style is better than another and professional teams may want to emulate that.

For the radar plots, the same proximity data is used. The data for supports is grouped by team and 4 separate average proximities are calculated, one for each teammate. The data is then normalised. That is to say, we scale these values to capture the contrast between all supports in the league. The proximities between all supports and their respective toplaners are scaled between 0 and 1 using a standard scaling formula:

$$Proximity_{new} = \frac{Proximity - min(Proximity)}{max(Proximity) - min(Proximity)}$$

With $Proximity_{new}$ being a list of scaled support-toplaner proximities, one for each team. The proximities to their junglers are then scaled and so on. Having each value on a scale of 0 to 1 will allow us to graph the data on a Plotly radar plot with the `graph_objects.Scatterpolar()` function. The values for any given team can thus be extracted and plotted to see how they compare to the rest of the league.

Pressure patterns

Once again, this study isn't particularly difficult once the data is gathered. We use the same subset of NA LCS games as in the *Proximity Patterns* project for this demonstration. Luckily, each of the four exits (for both sides of the jungle) lies on a straight line. This means that we simply have to check for every time the jungler walks over this line and save the locations of his team whenever this happens.

The main difficulty with this is accounting for times where they will exit the jungle and then quickly come back before exiting again right after. While this would be logged as two events, it doesn't make sense to dilute the data

considering the short time frame between each event. Thus, the algorithm tracks the time between each crossing of the line. Each time the jungler exits his jungle, it checks how long it's been since they last crossed over. If they've been on one side of the line for at least 10 seconds, then crossing the line is considered significant (for both entering and exiting the jungle). This algorithm, as described in Listing 3.5, is then applied to a large set of games and the locations aggregated.

```
if(x > (5/4) * y - 25): # If the jungler exits his jungle
   # If he has been in his jungle for at least 10 seconds
    if(count > 10 and not pressure):
      # Save points
      jgl.append((x,y))
      top.append(df.iloc[jungler_index-1][time])
      mid.append(df.iloc[jungler_index+1][time])
      adc.append(df.iloc[jungler_index+2][time])
      sup.append(df.iloc[jungler_index+3][time])

      pressure = not pressure # He's now outside his jungle
      count = 0 # Reset timer
    # If he's been in his jungle, but for fewer than 10 seconds
    elif(not pressure):
        pressure = not pressure # Toggle pressure
        count = 0 # Reset timer
    # If he was already outside his jungle
    elif(pressure):
        count += 1 # Increment timer
```

Listing 3.5: The algorithm for saving the locations anytime the jungler exits his jungle

Once these locations have been saved, they have to be cleaned. The

locations span the first 20 minutes of each game, but it is rare for all three lanes to stay in their lane the entire time. This means there are several points that aren't as useful for the purpose of this study. We want to see the state of the three lanes anytime the jungler plays forward, meaning any points outside the lanes are less useful and will distort the graphs. To this end, we filter the points to make sure they only cover their respective lanes. This is a simple case of classifying points using linear equations, just like in the role-based maps in Section 3.1.2(b). Once these points are found, they're used in density plots using Plotly's `graph_objects.Histogram2dContour()` function. This will serve to effectively show the areas a jungler's teammates tend to stand in anytime they push past the halfway point of the map.

Chapter 4

Evaluation

4.1 LolTracker Program

The program that was created was given the name *LolTracker* (League of Legends Tracker). It should be noted that originality in naming conventions was never a major goal of the project. LolTracker was then tested extensively, checking the data gathered from multiple different leagues and checking for obvious issues. The data was gathered from multiple games and the coordinates saved were overlayed over a replay of each game to see how well the players were tracked. No significant issues were found, though a small subset of champions were harder for the program to track due to the colour scheme of their portraits blending in well with the background.

The data was clean and the program had almost perfect accuracy in identifying the champions being played. Over 75 games were tested with one single misidentification out of 750 champions. Some leagues did prove troublesome due to the broadcast team slightly changing the overlay, but this was easily fixed by changing the dimensions of the identification crops. This is a demonstration of how slight changes can be made to the code in order to get around these discrepancies without much trouble.

The program also ran quickly. Tests were run using a Lenovo Thinkpad

T430 and a Youtube playlist of replays with an average video length of just under 40 minutes. It took approximately 100 seconds per video, fully automated. This shows how using the program can rapidly expedite the process of gathering data from videos.

The single game analysis graphs are also created automatically during this process. While the raw location data is saved, the graphs also provide an easily readable set of insights to the game itself. Overall, LolTracker performs to expectations and matches up well with the original objectives of the project

4.2 Large-Scale Projects

The projects did a good job of showcasing the potential use cases of the data. While it is surely possible to delve deeper into the wide variety of uses for spatiotemporal data in League of Legends, that was not the focus of these projects. When considered as showcases of potential, rather than in-depth studies into the data, they prove more than adequate.

4.2.1 Level One Patterns

The level one model was capable of showing the tendencies of players at the early stages of the game. The data used was the locations of G2 Esports, a professional team based in Europe, over the course of their League of Legends European Championship (LEC) 2020 Spring split. The model was built and graphs were drawn to showcase the difference in level one patterns between the 60 and 80 second marks. Overall, the model had an accuracy of 70%. While this may not seem like a lot, the confusion matrix shown in figure 4.1 paints the model in a better light.

As we can see in this graph, the model works really well with the exception of the botlane champions (ADC/support), who tend to stay in similar areas throughout the game. This is not a problem as both players are generally

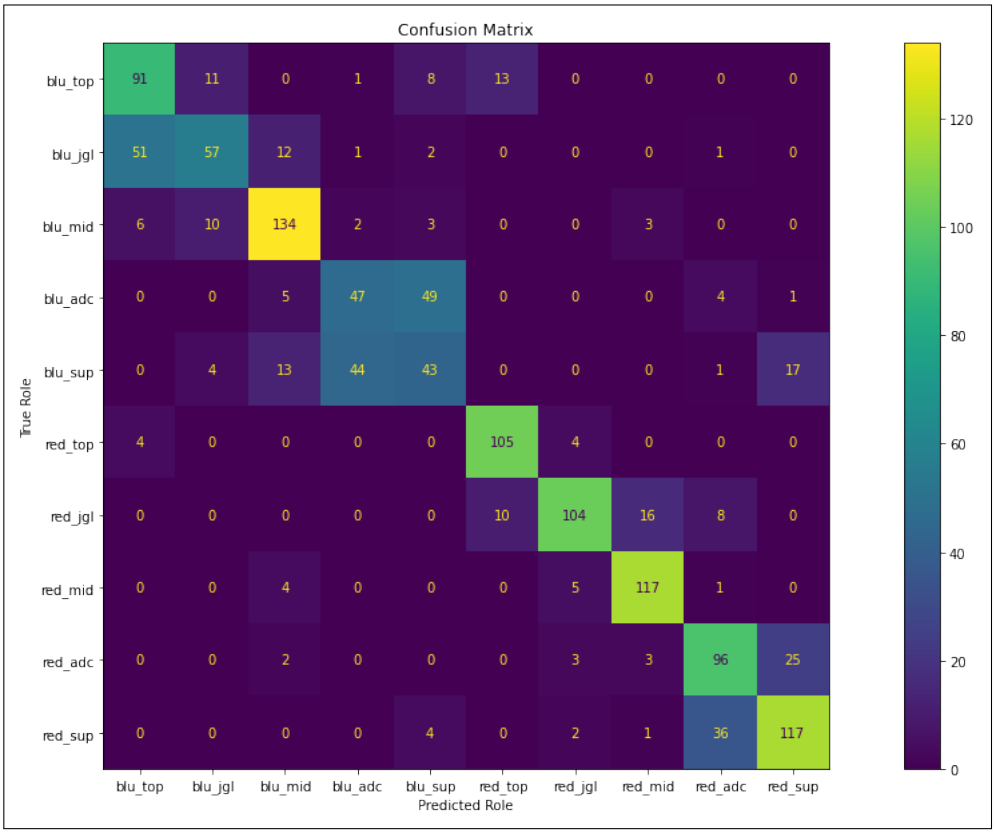


Figure 4.1: Confusion Matrix of the Logistic Regression model when used to predict the test data

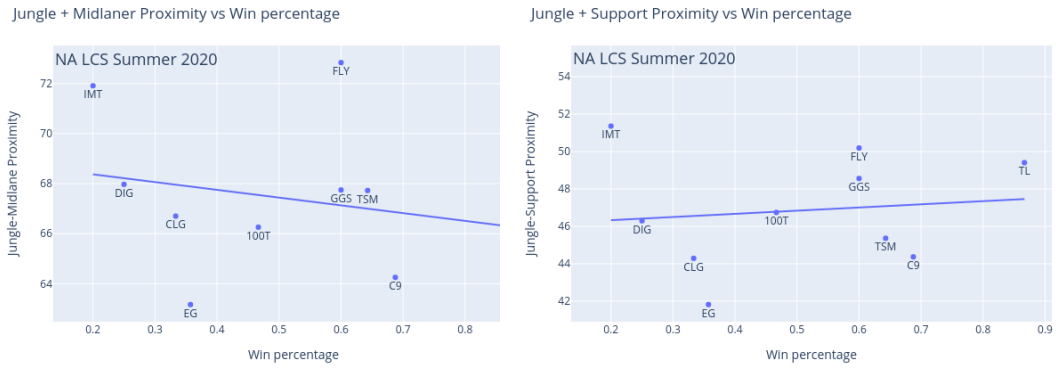
close enough for the model to recognise one or the other. It's also worth noting that the accuracy for red side is significantly higher than blue side, which makes sense as the data used to train the model has the same team (G2 Esports) representing the red side in every game, while the blue side data comes from the other 9 teams in the league. This means there is much more variance in the blue side data. Overall, we can see from this that the model has an acceptable accuracy for the purposes of this project: predicting G2 Esport's level one patterns when playing on the red side.

It showcased the capability of this data in terms of scouting an opposition team and spotting predictable patterns in their play. The graphs can be seen

in figures 3.9 and 3.10. It was found that looking at the maps between 60 and 80 seconds shows clear differences in their formation, meaning that G2 Esports tends to change their formation between these two times. This could prove useful for any of their opponents who wish to gain an early advantage.

4.2.2 Proximity Patterns

The relationship between jungle-midlaner proximity and winrate was analysed and plotted along with the relationship between jungle-support proximity. The resulting graphs are shown in Figure 4.2.



(a) Jungle - Midlaner (b) Jungle - Support

Figure 4.2: Proximity versus Win percentage graphs

As we can see here, there is little correlation between either of these statistics and winrate. There is a negligible slope on the trend line, indicating that these average proximity values have little effect on win percentage in isolation. It is possible that delving deeper into the data will provide more insights. For example, these statistics don't take the champions being played into account, adding extra variables such as this to the model may allow for a more robust study into what makes a team successful. It may also prove useful to test this over a larger selection of games and a wider variety of data sources.

Team	Top	Jgl	Mid	ADC
100T	0.00	0.52	0.37	0.81
C9	0.09	0.27	0	1
EG	0.59	0.26	0.21	0.82
TL	0.56	0.47	0.29	0.72
FLY	0.22	0	0.2	0.97
GGS	1	0.88	1	0
TSM	0.07	0.71	0.34	0.42
CLG	0.31	1	0.67	0.64
DIG	0.55	0.8	0.43	0.71
IMT	0.62	0.37	0.36	0.65

Table 4.1: Scaled proximity values for NA LCS Supports in the 2020 Summer split

For the radar graphs, the data was scaled using a simple min-max scaling algorithm. The resulting data frame is shown in Table 4.1. Each row of the table shows the scaled proximity values for one team's support in the Summer split. A row can then be used to create a radar plot, which will efficiently portray how each support's playstyle compares to the others. Some examples are shown in Figure 4.3, where Evil Geniuses' support Tristan 'Zeyzal' Stidam and Team Liquid's support Jo 'CoreJJ' Yong-in are compared. The graphs show that Jo tends to play near his jungler more than Stidam does, information that may prove valuable when comparing or analysing the two teams. This part of the project is a definite success, as it shows how spatiotemporal data can be used to automatically analyse playstyles of a player and their team. Once again, adding more variables to the data can show how players change their playstyle in different strategies. It could also be useful to compare a player's playstyle in games they lose versus games they win.

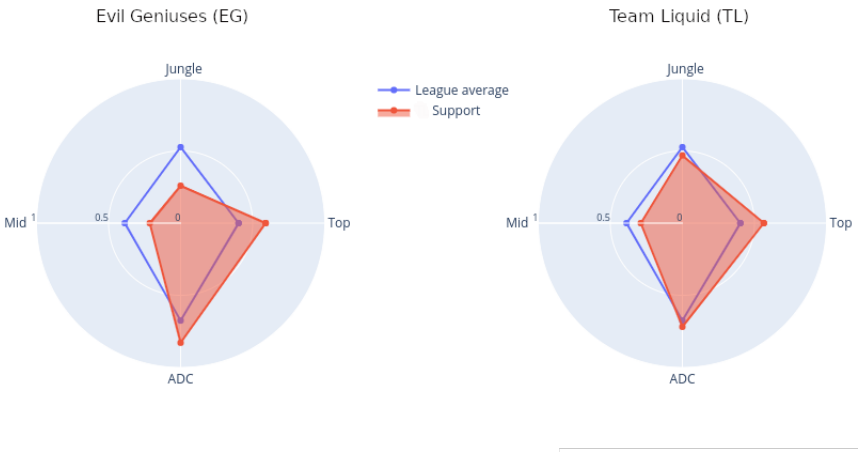


Figure 4.3: Radar graphs showing how two professional team's support players compare against the league average proximities

4.2.3 Pressure Patterns

In this project, we wanted to focus on a single team out of the database, in an effort to show how this method can be used to learn more about how a given team plays. To this end, we focused on the movement patterns of the Evil Geniuses (EG) League of Legends team, though the data is readily available to analyse any of the other teams. The locations of EG's toplaner, midlaner and AD carry were analysed and the graph shown in Figure 4.4 was drawn up.

The denser a region's points population is, the whiter the contours pertaining to that region are. From this graph, we can see that the top- and midlaners tend to stay back in their lanes, while the AD carry will push up past the halfway point, often leaving their lane to move northwards. The toplaner especially tends to play safe whenever their jungler pushes forward.

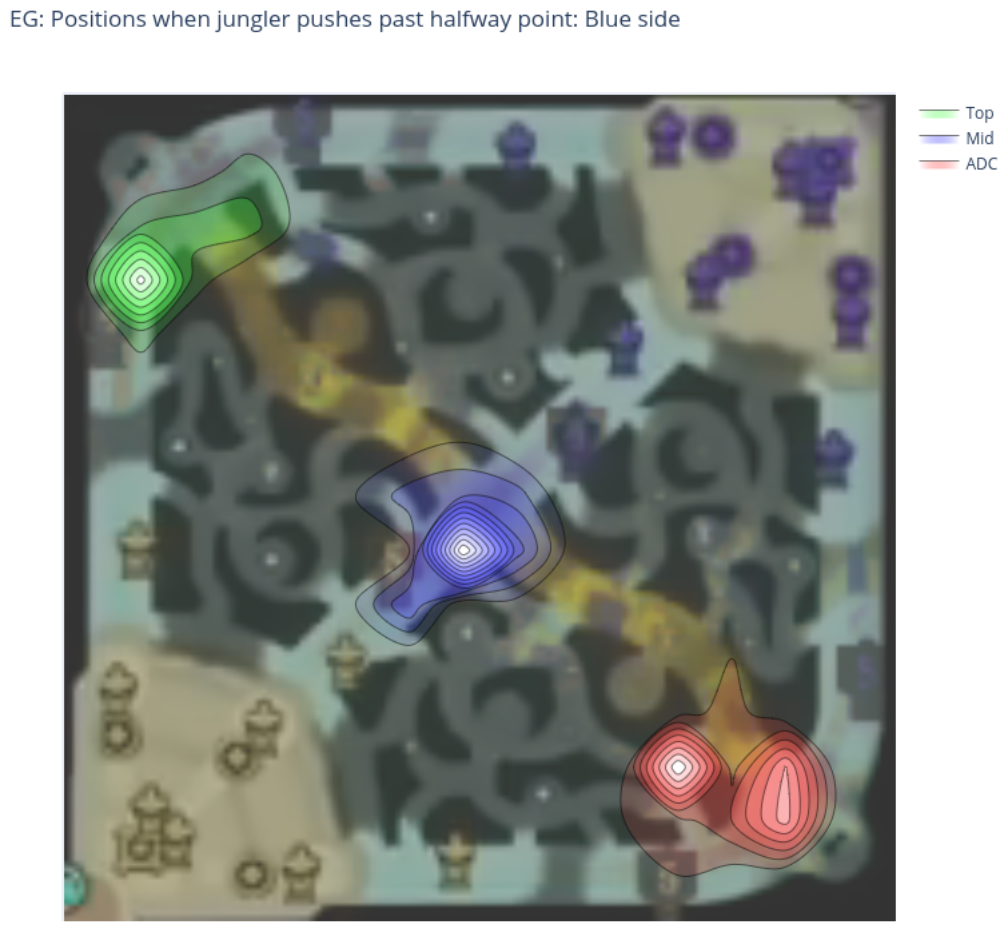


Figure 4.4: Density graph showing how EG's toplaner, midlaner and ADC tend to position when their jungler exits his jungle

Some reasoning for this can be seen in Figure 3.5, where there is a definite pattern to be seen. Dennis 'Svenskeren' Johnsen seems to favour moving through the exits on the south part of the map, especially the one just below midlane. It is thus a good idea for the toplaner to play back during this time, as he will have little support from his team if he plays too carelessly. Looking

Figure 4.5: Density graph showing where EG's jungler, Dennis 'Svenskeren' Johnsen tends to leave his jungle

at this map along with Figure 4.4, it can be deduced that the bottom half of the map is a big focus for Johnsen and the rest of EG. The process can be repeated for other teams to see how they compare to EG's playstyle.

Overall, this proof-of-concept is another success. The code is quick to execute and it does an excellent job of demonstrating patterns in a team's

play. The visualisations are clear and can certainly aid the analysis process of a competitive team.

Chapter 5

Conclusion

The project was aimed at coming up with a way of analysing spatiotemporal data in professional League of Legends. To achieve this, a program was created with the capability of automatically gathering this data and performing some basic analysis on it. The program is efficient and can automatically gather large amounts of data. Some small projects were also done as proofs of concept. While it wasn't an exhaustive study of the field, it can hopefully provide a springboard for further research. The creation of the program capable of gathering this data will allow for any future work to focus on the data itself, allowing for a more in-depth exploration of the analytical possibilities. The initial interest from professional teams and analytics companies is very positive, showing the potential for the program to play a key role in the analytics of a rapidly growing industry.

5.1 Limitations

While the program is easy enough to set up, it's not necessarily intuitive for someone without a technical background. The constantly updating champion pool and broadcast overlays also poses a problem as anyone using the program will need to know how to keep it up to date in line with these changes.

Several scripts were created to make this process easier and are included in the Github, though again a technical background is likely necessary to effectively use them. The overall accuracy is not an issue for larger-scale analysis, but the margin of error may be too high for particularly precise studies.

5.2 Future Work

Future work could easily focus on any three of the main objectives of this project. The program could be further changed to increase accuracy and track additional aspects of the game. Events such as kills, tower destructions or items purchased could potentially be tracked using similar computer vision techniques, leading to more exhaustive and detailed databases to work with.

The single game analysis could be expanded on by adding more graphs to the output. Quickly analysing a single game is a very useful tool; the more insights that can be instantly gained, the more effective the tool is.

Naturally, the large scale analysis is the objective with the highest potential. Gathering and analysing this data on a large scale has countless use cases. If more modern and sophisticated statistical or machine learning methods are employed, then the possible advantages that a professional team could gain are boundless. With a sufficiently large database, it could be used to train models that quantify general statistics, comparing a single player's tendencies against a general population to find their weaknesses/strengths.

It's a realistic expectation for esports organisations to be able to use this data to scout semi-professional leagues in an attempt to find rookie players that fit their desired profile. The data is especially useful for finding nuances in someone's play that is not necessarily possible with traditional statistics such as kills, deaths, vision score and such, making it a powerful tool for finding an inexperienced player with the playstyle you're looking for. It can also be used to scout opposing teams easily, seeing how a given team's

play may differ when they employ one strategy over another can lead to counter strategies being devised. Being able to predict the movement of an opposing team offers a significant advantage in any competition, with League of Legends proving no exception. Simply looking at the traditional sports that have been revolutionised by new streams of data shows the potential of this field, there is no doubt that this project could be a game changer in the right hands.

Appendix A

LolTracker Github

Github Link

Bibliography

[1] Adam Newell. *How many League of Legends players are there?*, 2019 (accessed June 28, 2020). https://dotesports.com/league-of-legends/news/league-of-legends-number-of-players-14488.

[2] LoL Esports Staff. *2019 World Championship hits Record Viewership.* Riot Games, 2019 (accessed June 29, 2020). https://nexus.leagueoflegends.com/en-us/2019/12/2019-world-championship-hits-record-viewership/.

[3] Miles Yim. *People are investing millions into League of Legends franchises. Will the bet pay off?* Washington Post, 2019 (accessed Aug 3, 2020). https://www.washingtonpost.com/video-games/esports/2019/11/18/people-are-investing-millions-into-league-legends-franchises-will-bet-pay-off/.

[4] Microsoft. Cloud9 takes esports to the next level. online, 2020. Available at: https://www.microsoft.com/inculture/sports/cloud9/ (accessed Aug 3, 2020.

[5] SAP. Sap and esports: The next chapter with league of legends. online, 2019. Available at: https://news.sap.com/2019/10/esports-league-of-legends-sap-sponsorships/ (accessed Aug 3, 2020).

[6] Michael Lewis. *Moneyball : the art of winning an unfair game.* W.W. Norton, 2004.

[7] Bruce Schoenfeld. How data (and some breathtaking soccer) brought liverpool to the cusp of glory, May 22, 2019. New York Times https://www.nytimes.com/2019/05/22/magazine/soccer-data-liverpool.html.

[8] Philip Z. Maymin. An open-sourced optical tracking and advanced esports analytics platform for league of legends. *MIT Sloan Sports Analytics Conference*, 2018.

[9] Kirk Goldsberry Ph.D. Courtvision: New visual and spatial analytics for the nba. *MIT Sloan Sports Analytics Conference*, 2012.

[10] J. Brooks, M. Kerr, and J Guttag. Using machine learning to draw inferences from pass location data in soccer. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 9:338–349, 2016.

[11] Ho-Chul Kim, Oje Kwon, and Ki-Joune Li. Spatial and spatiotemporal analysis of soccer. In *GIS '11: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 385–388, 2011.

[12] P.A. Longley. Modifiable areal unit problem. *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, 2017.