August 2021

Process mining applied to League of Legends to achieve performance insight by using API data extraction

Industrial Engineering & Management
Bachelor Thesis

Author: Vafi, M.A. (Student B-IEM,M-IEM)
First supervisor: Dr. G.W.J. Bruinsma
Second supervisor: Dr. I. Seyran Topan

UNIVERSITY OF TWENTE.

Bachelor thesis Industrial Engineering and Management

Process mining applied to League of Legends to achieve performance insight by using API data extraction

Author

M.A. Vafi (Matthijs)

University of Twente
Drienerlolaan 5
7522 NB Enschede
(053) 489 9111

Supervisors University of Twente
First supervisor: Dr. G.W.J. Bruinsma (Guido)
Second supervisor: Dr. I. Seyran Topan (İpek)

Management summary

eSports is an industry which has been growing over the past decade. This growth has been realised in terms of viewers, players, and money involved. Due to the growth, the eSports scene has been catching up on traditional sports. One particular game with a large eSports scene is the video game League of Legends. Individual players, as well as the eSports teams, benefit from improving gameplay. Performance improvement can be achieved in terms of improving physiology, psychology, strategies, and team coherence. Data analysis is also a way of gaining insight in gameplay, which can lead to performance improvement. There are websites online which show dashboards where information is summarised and shared for an inputted player to analyse. However, these dashboards only show statistics, and do not take into account the timing when events during a match occurred. Furthermore, such dashboards primarily show data which League already shows to the player in the game itself. There seems to be a lack of applications which analyse in-game objectives, which play a critical role in winning a match in League. Academic research also does not provide a tool or application which combines the idea of analysing in-game objectives as well as the timing of such events. Thus, the research of this thesis aimed to create a tool which analyses the timed sequence of events concerning in-game objectives, in order to gain insight in an inputted player's playstyle, which could be used for personal improvement goals, as well as analysing and creating counter strategies against opponents.

To realise this analysis tool, requirements had to be devised, which makes sure that the tool would be sufficiently useful. Three requirement categories were created, namely "Collect data", "Analyse data to distinguish winning patterns", and "Guarantee user friendliness". Each requirement category has its own requirements which need to be satisfied in order to sufficiently satisfy the corresponding requirement category. The "Collect data" category aimed to make sure that the online data source for the tool was reliable and that the user defined inputted data can be analysed without errors. The "Analyse data to distinguish winning patterns" category aimed to make sure that the tool scans and analyses the match history of the inputted player to analyse by checking for each match the impact of taking an objective, as well as the timing of it in a match. The "Guarantee user friendliness" category aimed to make sure the tool runs automatically with minimal input from the user, as well as providing options to visualise and filter information.

League provides an online API for players to use. They can use it to extract information about their matches played. This API was used by the tool constructed in this thesis. The tool consists of a Python code which asks input from the user, such as which player account to analyse, and then extracts information of the player's matches concerning in-game objectives. Whether the player won the concerning match when a certain objective was taken is used as a measurement of success for the objective. The collected data is put into an Excel file which uses VBA coding to interpret the data and create a Pivot table and chart to visualise it. The user has the possibility to filter information in the table and graph.

The constructed tool was tested by inputting a professional eSports player. The tool worked as intended and the requirements were sufficiently satisfied. A sensitivity analysis was executed for certain requirements where possible. For example, one requirement was that

every player in the world could be inputted in the tool. This was tested by inputting players from different regions and check whether the tool would adjust accordingly, which was indeed the case.

A survey was constructed as a validation method to check to what extent potential users thought the tool was suitable, and what improvements points they could think of. League players from the Esports Team Twente were sent the survey to fill in. The response concluded that the analysis of in-game objective events as well as their timing could indeed be beneficial. Certain points of improvement were also formulated. One example was that a more detailed analysis could be more beneficial, since League contains a lot of factors which also influence the chances of victory.

The tool is initially used to analyse the success of taking objectives for the inputted player. Even though the tool is not able to analyse the success of other events that occur during a match, it requires relatively small changes in the Python code to adjust this, because the API also contains data of other events. Furthermore, the design of the tool could be based initially on the opinions of eSports players, instead of using their opinions as validation. Such a design plan could also benefit if software would be used to create a standalone application, instead of separate Python and Excel files, which could improve user friendliness. Finally, the API has its limitations, as data is gathered every minute, and not every second. Using analysis software to gather data from replays of matches can improve the data and make it more detailed. All these aforementioned possibilities can be used as starting point for further research.

Preface

In front of you lies my bachelor thesis which I conducted for my bachelor study Industrial Engineering & Management on the University of Twente. The assignment was conducted internally on the university with Guido Bruinsma as my first supervisor.

First of all, I would like to thank my first supervisor Guido Bruinsma. I felt lucky that I was able to do a bachelor assignment in a field of study that seemed interesting to me, namely eSports. I was also glad that this assignment had a significant part of coding, since I thought it is nice to make a product which works on its own. I want to thank him for the feedback he provided, his patience, and his friendliness. I felt that I could do the things that I want without feeling too much pressure.

Second of all, I would like to thank Ipek Seyran Topan, my second supervisor. She has helped me since the graduation preparation of module 11. She helped me by relieving working pressure and by answering all my questions. Her help continued when I started and worked on my thesis assignment, and I would receive answers almost instantly whenever I asked something. I want to thank her friendly commitment to me and how she supported me throughout my whole graduation process.

I would also like to thank the League players of the Esports Team Twente for helping me out with the external validation part for the tool.

Finally, I would like to thank my mother and my friends for supporting me throughout the graduation. Especially Casper and Pim were two fellow graduate students who were also busy with all the ups and downs that come with graduating, and hanging out with them helped me charge up.

Matthijs Vafi, August 2021

Table of Contents

Management summary ii

Preface iv

Chapter 1: Context analysis 1

Chapter 2: Theoretical framework 3

Chapter 3: Method of approach 13

- Collect data 13
- Analyse data to distinguish winning patterns 15
- Guarantee user friendliness 17
- Prototype 18
- Testing of the tool 21

Chapter 4: Results 22

- Analysis of Rekkles 27
- Requirements sensitivity analysis 28
- Validation 29

Chapter 5: Discussion and conclusion 31

- Collect data 31
- Analyse data to distinguish winning patterns 32
- Guarantee user friendliness 32
- Conclusion 33
- Discussion and limitations 33

Chapter 6: Further research 35

References 38

Appendix A: The game League of Legends 41

- Game set-up 41
- Objectives 42
- Time periods 44
- Remake 44

Appendix B: Functioning of the tool 46

- Python 46
- Excel 51

Appendix C: Python script 55

Appendix D: VBA macros 67

Appendix E: External validation survey 69

Chapter 1: Context analysis

eSports, which is short for Electronic Sports, is the world of organised and competitive video gaming, where millions of fans all over the world attend live or online events to watch the competition play (Willingham, 2018). Professional gaming teams or individual competitors battle each other for a title or championship, which includes prize pools, ad revenues (Reyes, 2021), and other income streams like selling merchandise (Pan, 2018). Research has been done on the growth of eSports and a continuation of growth is expected (Newzoo, 2021).

The eSports scene has also grown compared to traditional sports and gathered in 2019 more viewers than rugby and American football combined (Olya, 2021). Furthermore, the amount of eSports watchers is expected to surpass the baseball scene in 2022. Especially younger watchers, ranging in age from 18 to 25 years old, spend considerably more time watching other people playing games than watching more traditional (Limelight Networks, 2020). The amount of eSports players also grows when comparing 2014 with 2019.

Certain parties have already started anticipating on the growth of eSports over the year. For example, Madison Square Garden, which has a capacity of 20.000 people, has been used for eSports events and managed to consistently fully fill it with eSports watchers. Japanese entertainment company Konami has constructed a big facility of 12 stories high which is focussed on eSports activities, called the Konami Creative Center Ginza (Olya, 2021). Furthermore, eSports has also been speculated to be part of future Olympic Games, and eSports is even already part of the 2022 Asian Games.

In eSports, investments take place. This aspect is investigated on further in the following part.

The prize pools of eSports tournaments also have increased over the years and are starting to become comparable to the prize pools of more traditional sports, such as football, golf, cricket, darts, and cycling (Wagner, 2018). To highlight the growth of the eSports scene with an example: the world championship event of DotA 2 held in 2019 had a prize pool of over 34.3 million USD, which is also the biggest prize pool so far of an eSports tournament (Nordmark, 2021).

The top 10 Forbes eSports organisation ranking list has the trait that 7 of the top 10 eSports organisations currently have franchise League eSports teams (Settimi, 2020). The top 3 organisations, TSM, Cloud9 and Team Liquid, all have a history in the North American League of Legends eSports scene. TSM, with its value around 410 million USD, is the biggest eSports organisation in the world, and has been part of the League eSports scene from the beginning.

As highlighted, especially League of Legends is an eSports scene which has seen growth and is currently one of the biggest eSports industries in the world (Shelp, 2020). The growth over the years of this eSports scene will be explained and elaborated on next.

The overall number of viewers who watch League of Legends grows over the years (Statista, 2021). To illustrate, the number of viewers for the League of Legends World Championship finals grew to 100 million in 2019 (Pei, 2019). For comparison, the Super Bowl of that year had roughly 98 million viewers and its viewership is on a declining trend.

This growth of the League of Legends eSports scene manifests in, among others, viewers, players, sponsors, investors, and business partners. Football and basketball sports clubs like FC Schalke 04, Milwaukee Bucks and Golden State Warriors have started to create their own professional League of Legends eSports teams and are competing on a professional level (Church, 2020).

The eSports scene keeps growing over the ages. League of Legends is one of the more prominent games with an eSports scene and it has been established for roughly ten years now (Shelp, 2020).

League of Legends is a competitive video game. Performance optimisation could be beneficial for the professional eSports teams, its players on the team, and players who do not participate in the eSports scene but focus on improving their gameplay. Other more traditional sports, like football, has staff to improve performance with relation to coaching, game analysis, fitness and medical issues. There are also companies, like SciSports, which aim to improve football performance via data algorithms. Similarly, League of Legends eSports teams also has coaches, analysts, and data engineers, which could optimise the performance of their eSports players. There are papers of academic research available which explain about possibilities for performance optimisation concerning League of Legends, such as optimisation with regard to physiology, psychology, game strategies, and team coherence. Performance optimisation could also be achieved by insight in the game performance of the individual. Concerning such game analysis, there are websites available which show statistics about what happened during a match played. However, these websites do not provide insight in the analysis of events that happened during a match with relation to time. Such time series analysis could provide more insight in an individual's performance, which leads to performance improvement.

The available League of Legends academic research is investigated upon further in Chapter 2. To understand the material researched in Chapter 2, it would be beneficial to understand the video game League of Legends better. Thus, basic information about the game will be explained briefly.

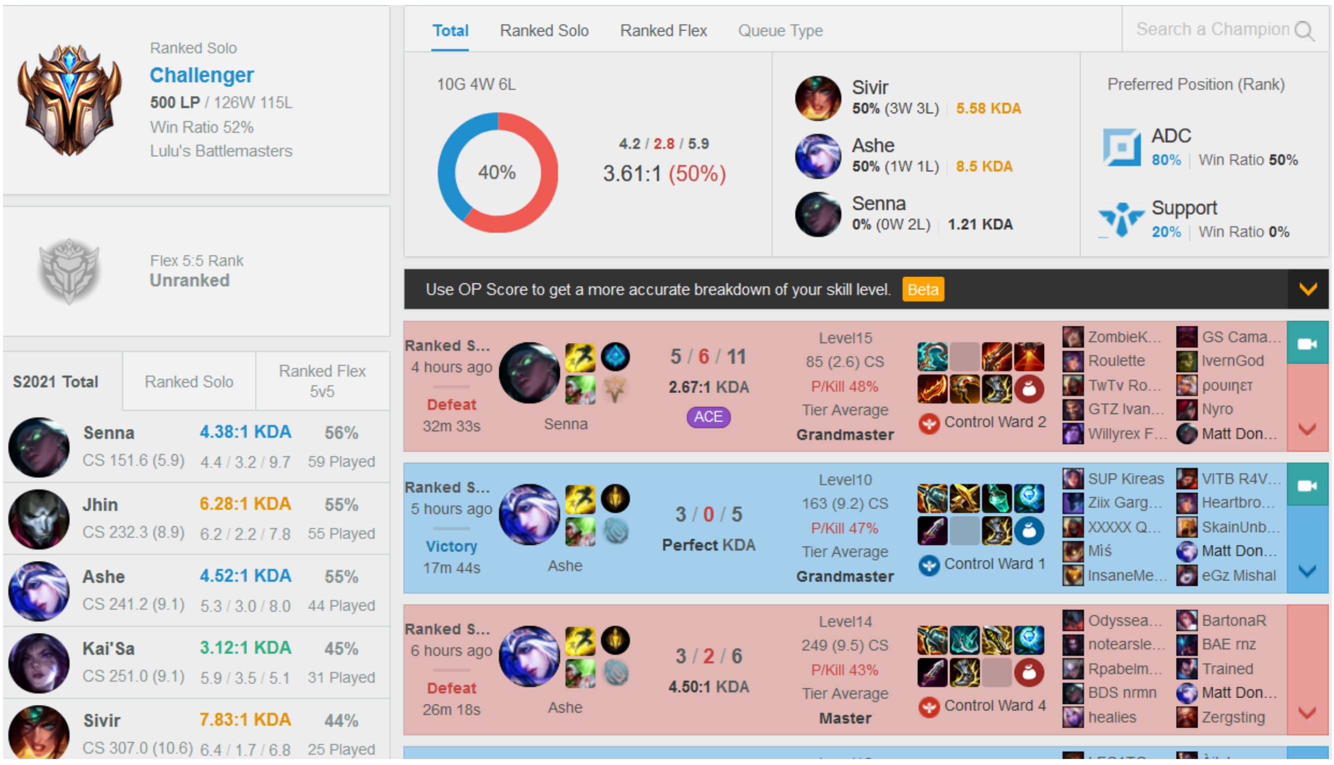
The strategy video game League of Legends (abbreviated as League) consists of matches played between 2 teams of 5 players each and the players try to destroy each other's base, which is situated on their respective side of the map. Bases are guarded by turrets which needs to be destroyed first. Across the map, there are certain neutral monsters which can be killed to make the team who killed it permanently stronger for the rest of the match. Destroying turrets or these monsters are labelled as 'objectives'. More in-depth explanation about League is provided in Appendix A.

Chapter 2: Theoretical framework

By investigating previous work and research, a starting point can be identified to find a way to solve the main research opportunity faced in Chapter 1, which was about performance optimalisation with regard to game analysis of League matches. Similar research might have provided some answer regarding the research opportunity, but this should be analysed to what extent this is done. Such research can be used as a starting point to modify the research goals and adjust it in such a way so that a new innovative solution can be created.

This chapter starts off by listing current available data analysis visualisation methods. A few websites and applications are shown to sketch the current state of visualisation possibilities, to help understand what is offered. This shows what is currently done, and also shows what is left to improve. Next up, the academic database Scopus is used to search for other methods, such as data visualisation, that were used on eSports and League specifically. This can show which methods are researched, and can highlight what state-of-the-art possibilities there are concerning data analysis. The focus during this orientation phase is to see whether a method truly adds value in the sense that the data can provide insight to a team's or player's performance and whether or not it can contribute to winning. The main focus is that either the research or applications should be able to improve the win rate of a team or player. The checking of current available data visualisation applications, and the analysis of literature of the topic, will realise a requirement list to accomplish setting steps in the direction of creating an innovative solution concerning data analysis in League. The new method created in the next chapter should be able to meet the requirement list, and thus provide both a practical and innovative new method.

The current available data analysis visualisation methods will now be analysed. Most of these available methods are realised in the form of websites. There are already websites available which visualise and summarise the data that happened in a game. The best known website for this is OP.GG, which is a website where you can fill in a summoner name of a player and check their match history, along with their Ranked Queue ranks and other data. A nice feature about OP.GG is that it easily also shows the rank, level, and runes of your opponents and teammates in the game you are currently in. Figure 1 shows what OP.GG looks like. OP.GG uses the League of Legends Riot API as data input. As an example, the main account of a famous League eSports player called Rekkles is analysed. League players often change the name of their account, which is the summoner name. The main name of the player is Rekkles, but he has roughly 5 other accounts which each have other summoner names. Thus, at the time of creation of Figure 1 and 2, one of his accounts had the name: "Matt Donovan". This account was used to show what the concerning websites look like in the figures.



Ranked Solo
Challenger
500 LP / 126W 115L
Win Ratio 52%
Lulu's Battlemasters

Flex 5:5 Rank
Unranked

S2021 Total	Ranked Solo	Ranked Flex 5v5

Champion	CS	KDA	K / D / A	Win %	Played
Senna	CS 151.6 (5.9)	4.38:1 KDA	4.4 / 3.2 / 9.7	56%	59 Played
Jhin	CS 232.3 (8.9)	6.28:1 KDA	6.2 / 2.2 / 7.8	55%	55 Played
Ashe	CS 241.2 (9.1)	4.52:1 KDA	5.3 / 3.0 / 8.0	55%	44 Played
Kai'Sa	CS 251.0 (9.1)	3.12:1 KDA	5.9 / 3.5 / 5.1	45%	31 Played
Sivir	CS 307.0 (10.6)	7.83:1 KDA	6.4 / 1.7 / 6.8	44%	25 Played

Total | Ranked Solo | Ranked Flex | Queue Type | Search a Champion

10G 4W 6L
40%
4.2 / 2.8 / 5.9
3.61:1 (50%)

Champion	Record	KDA
Sivir	50% (3W 3L)	5.58 KDA
Ashe	50% (1W 1L)	8.5 KDA
Senna	0% (0W 2L)	1.21 KDA

Preferred Position (Rank)

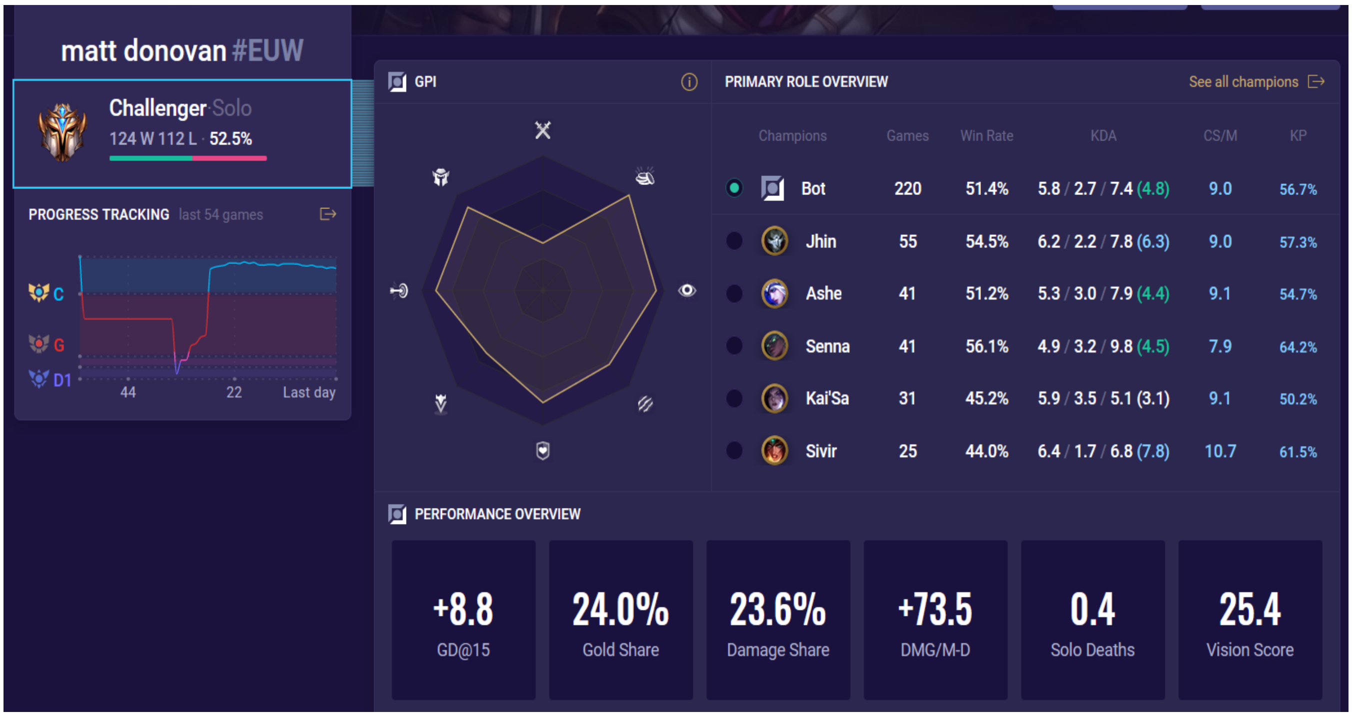
Position	Share	Win Ratio
ADC	80%	Win Ratio 50%
Support	20%	Win Ratio 0%

Use OP Score to get a more accurate breakdown of your skill level. Beta

Game	Champion	KDA	Stats	Wards	Team 1	Team 2
Ranked S... 4 hours ago Defeat 32m 33s	Senna	5 / 6 / 11 2.67:1 KDA ACE	Level15 85 (2.6) CS P/Kill 48% Tier Average Grandmaster	Control Ward 2	ZombieK... Roulette TwTv Ro... GTZ Ivan... Willyrex F...	GS Cama... IvernGod pouιηεт Nyro Matt Don...
Ranked S... 5 hours ago Victory 17m 44s	Ashe	3 / 0 / 5 Perfect KDA	Level10 163 (9.2) CS P/Kill 47% Tier Average Grandmaster	Control Ward 1	SUP Kireas Ziix Garg... XXXXX Q... Miś InsaneMe...	VITB R4V... Heartbro... SkainUnb... Matt Don... eGz Mishal
Ranked S... 6 hours ago Defeat 26m 18s	Ashe	3 / 2 / 6 4.50:1 KDA	Level14 249 (9.5) CS P/Kill 43% Tier Average Master	Control Ward 4	Odyssea... notearsle... Rpabelm... BDS nrmn healies	BartonaR BAE rnz Trained Matt Don... Zergsting

Figure 1: The summoner stats from the summoner Matt Donovan. Extracted from [Matt Donovan - Summoner Stats - League of Legends]. (2021). OP.GG Europe West. Retrieved from https://euw.op.gg/summoner/userName=matt+donovan

OP.GG is not the only website which shows data visualisation of games played. There are other websites which show and visualise data or winning strategies off champions, like Champion.gg, Blitz.gg, Mobafire.com and Mobalytics.gg. Since Mobalytics tries to show more data that other sites do not, like showing a playing profile. This is unique to Mobalytics and an example of such a playing profile is shown in Figure 2. Its graph has aspects such as Fighting, Consistency, Versatility, Survivability and Aggression. This graph is purely meant to give an idea of what the data analysis roughly looks like, and thus these aspects are not explained further.



matt donovan #EUW

Challenger Solo
124 W 112 L · 52.5%

PROGRESS TRACKING last 54 games

PRIMARY ROLE OVERVIEW See all champions

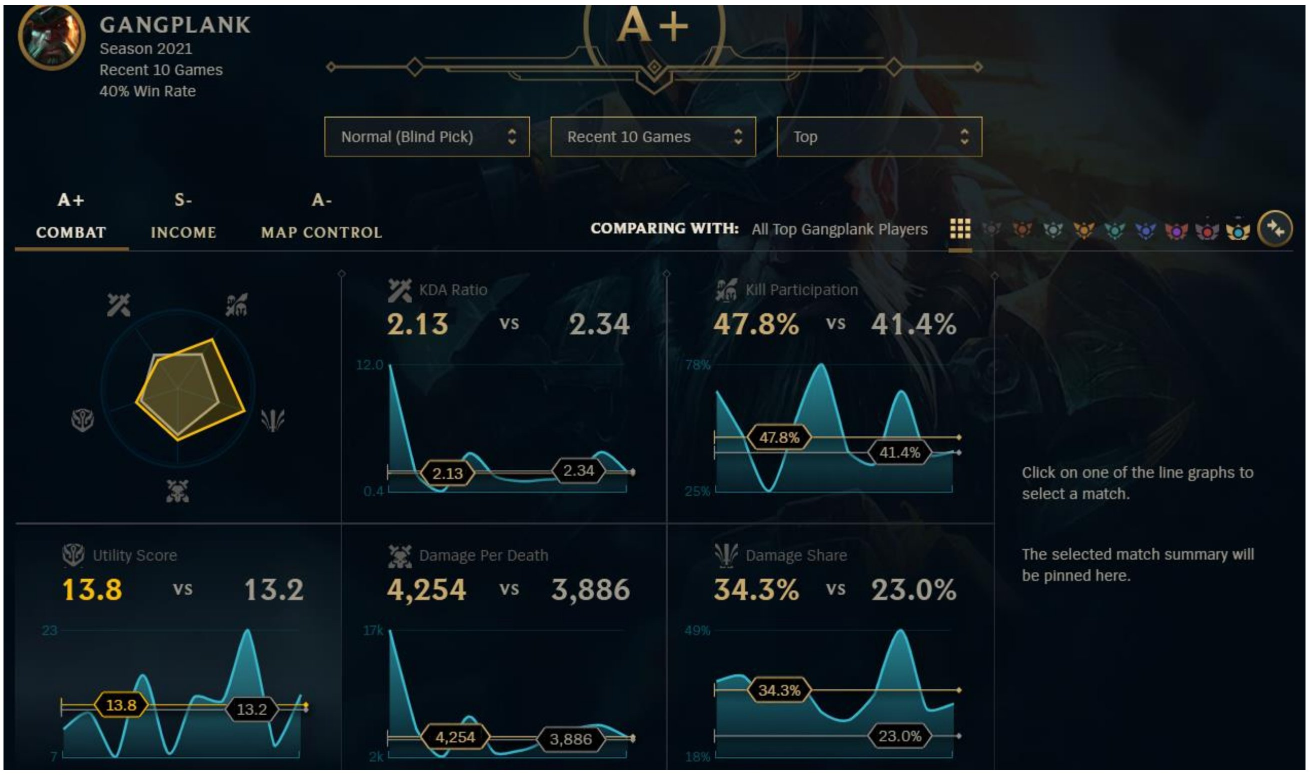
Champions	Games	Win Rate	KDA	CS/M	KP
Bot	220	51.4%	5.8 / 2.7 / 7.4 (4.8)	9.0	56.7%
Jhin	55	54.5%	6.2 / 2.2 / 7.8 (6.3)	9.0	57.3%
Ashe	41	51.2%	5.3 / 3.0 / 7.9 (4.4)	9.1	54.7%
Senna	41	56.1%	4.9 / 3.2 / 9.8 (4.5)	7.9	64.2%
Kai'Sa	31	45.2%	5.9 / 3.5 / 5.1 (3.1)	9.1	50.2%
Sivir	25	44.0%	6.4 / 1.7 / 6.8 (7.8)	10.7	61.5%

PERFORMANCE OVERVIEW

+8.8	24.0%	23.6%	+73.5	0.4	25.4
GD@15	Gold Share	Damage Share	DMG/M-D	Solo Deaths	Vision Score

Figure 2: The playstyle of summoner Matt Donovan according to Mobalytics. Extracted from [Matt Donovan (region: EUW) - Champion Stats - League of Legends (Season 11)]. (2021). Mobalytics. Retrieved from https://app.mobalytics.gg/lol/profile/euw/matt%20donovan/overview

League of Legends itself also has a way of analysing player specific data and uses it to make a playing profile for each champion on each position played, if enough data is gathered. It's similar to the graph of Mobalytics, but not completely. It uses three categories of gameplay to visualise someone's playstyle, which are Combat, Income, and Map Control. Aspects shown are fairly general and can be easily viewed in tables and graphs of post-match data. Concerning aspects are, for example, Damage Share, Kill Participation, minions killed per minute, early gold advantage, Objective Control Ratio, and Vision Score. A dashboard of the Combat category is shown in Figure 3. The higher the grade, the better your performance compared to other players of different selected ranks.


GANGPLANK
Season 2021
Recent 10 Games
40% Win Rate

Normal (Blind Pick) | Recent 10 Games | Top

COMPARING WITH: All Top Gangplank Players

Click on one of the line graphs to select a match.

The selected match summary will be pinned here.

Figure 3: The playing profile visualisation from the game League itself. Extracted from: Riot Games, Inc. (2009). League of Legends (Version: patch 10.7) [Video game]. Los Angeles, CA: Riot Games, Inc.

As shown, there are ways to visualise data of players in different contexts. However, none of these options take the sequence of events that happen during a game into account, which are events in the context of when something happened during the game. Especially concerning objectives taken during a game. Objectives have a large impact on the winning probability of a game, but a clear analysis lacks so far. Only static data is analysed, categorised and visualised. Thus, academic literature should be analysed to check whether there are already methods developed to analyse the importance of the occurred sequence of events during a game.

Thus, an innovative dashboard should so far contain the following tentative requirements to distinguish itself from current visualisation applications:

- The sequence of events of a match should be taken into account, instead of just analysing simply post-game statistics
- In -game objectives should be analysed

The next goal is to follow up on the search for innovative requirements by looking at academic literature. It could be that academic research has already created methods which meet to the tentative requirements list, even though such methods may perhaps not be created in the form of an application or website yet. The main goal is to provide an innovative solution, and if such a solution is already done in academic research, then such a solution is not innovative after all, thus the academic literature should be analysed. Such academic literature was reviewed by using the academic database Scopus. The total relevant list of sources are listed in the appendix. Keywords were used such as MOBA, League of Legends, data, mining and strategy. The searching started of simple by searching for general research on the eSports aspect of League of Legends.

Data analysis in League of Legends, and also for MOBAs in general, appears to be a useful endeavour according to, for example, Sangster et al. (2016). In this paper, data about kills, assists and the familiarity with teammates are analysed in an exploratory way. The researcher conclude that expanding research in this field of study has beneficial possibilities. The first article found specifically about this data analysis on League was about a dashboard build which is called VisuaLeague (Afonso et al., 2019). It uses in-game data to visualise what happened during the game, like making a 2D animation of how each player walked during the game, thus creating walking routes. The data in the article used was said to be extracted from the API of Riot. This API is accessible to general users and thus could be used as a data extraction source to collect information about League and about matches played. Simply for giving an idea of the dashboard, the VisuaLeague dashboard is shown in Figure 4. In comparison to previously mentioned data visualisation, the VisuaLeague dashboard also shows walking routes which happen during a game, which can be viewed on the upper left map, thus it shows some information about the occurred sequence of events. Perhaps the API data could be used to find more information about the timing of events that happen during a match.

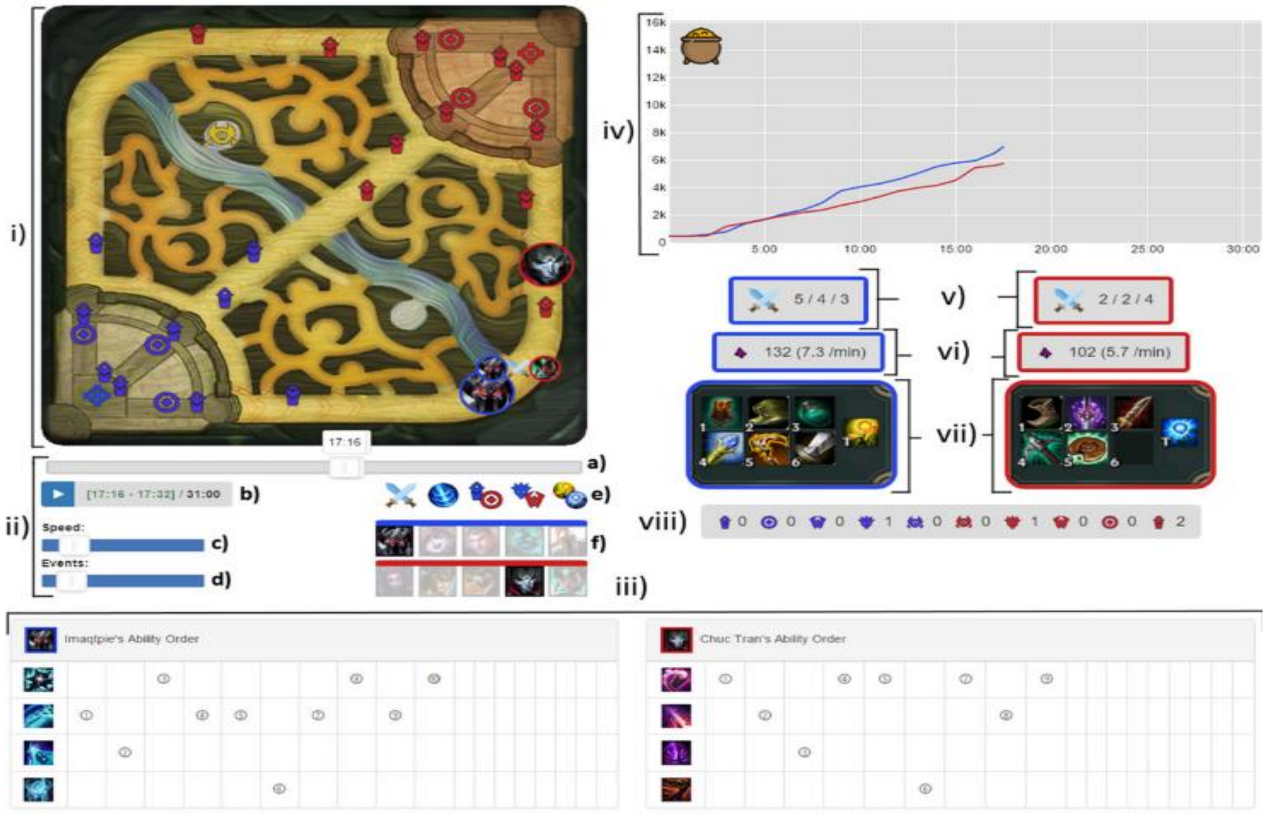


Figure 4: The VisuaLeague dashboard from Afonso, A. P., Carmo, M. B., Gonçalves, T., & Vieira, P. (2019b). VisuaLeague: Player performance analysis using spatial-temporal data. *Multimedia Tools and Applications*, *78*(23), 33069–33090. https://doi.org/10.1007/s11042-019-07952-z

In the article of (Charleer et al., 2018), dashboards were built for League and also another popular shooter game called Counter Strike: Global Offensive. The researchers tried to make a dashboard for League which helps spectators watch official eSports matches better by visualising potentially more useful information. They created a new visualisation module which has not been known before, called Vulnerability, which is an indicator of how likely a champion is to die in the current situation. This dashboard also shows other data like gold progression and damage dealt by each champion. Even though this dashboard uses a new way of analysing data which is also constantly updated, in the form of the Vulnerability module, no clear overview is given over the sequence of events at the end of the game. The data module simply shows which champion is vulnerable to dying at the moment, which is meant to inform the spectator. For this dashboard, a Websocket was used which was provided by the League eSports broadcasters, instead of API data like in the VisuaLeague paper (Afonso et al., 2019a). In another paper of Afonso et al. (2019b), various visualisation tools for League are compared. Interestingly, a VisuaLeague version 2 was developed in for this research. The research was meant to test which visualisation is liked more by players, where the result was that players favoured dynamic spatio-temporal visualisation, like watching a replay of a match or a walking route animation. The walking route animation, along with the replay of a match, shows the distinction between early events and late events happening in a game, and thus seems to show that players prefer indeed data which give insight in the sequence of events of the concerning game.

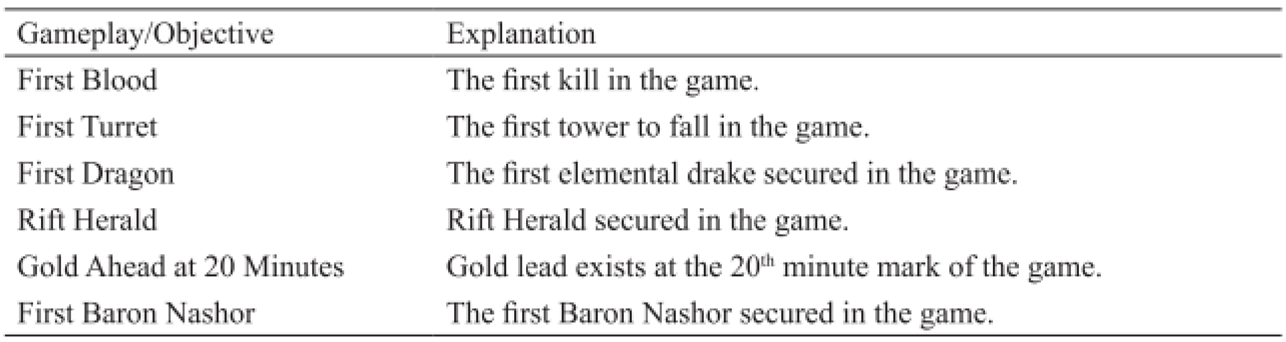
A nice goal of visualising and analysing the sequence of events in a game is to see what went wrong, fix it, and measure the improved results. Thus, the analysis of useful and less useful strategies should also be considered. The paper of Gerber et al., 2019 paper uses API data to measure team performance and check performance improvement after an improvement program was executed. This improvement program consisted of a three-day sleep-over summer camp to do team building exercises with the goal of improving the mental aspect of a player and, eventually, in-game team coordination. This research aimed at organising activities which are totally separate from the game League of Legends itself. However, another paper, which is from Lee & Ramler (2017), tested various different in-game team strategies to check whether such behaviour would result in more victories. These team strategies referred to mixing up roles. Thus, for example, the team composition of having two junglers instead of the usual team role composition of one top laner and one jungler, was considered. Readily available post-game statistics delivered by Riot was used to compare the success of different team composition. Another paper (Sapienza et al., 2018), also analysed the in-game behaviour of players. It used patterns in post-match kills, assists, deaths and amount of earned gold to cluster behaviour types.

There is also research done about comparing the success of deviating strategies to a norm for other MOBAs, like DotA 2 (Cavadenti et al., 2016). The research paper from Sapienza et al., (2017) also researches DotA 2 to analyse which factors determine success for players. Analysed factors were about overall player experience with the game and the type of player specific in-game character choice. Another DotA 2 study which analyses success factors is from Xia et al., (2019), and another study focussed on the application of machine learning on predicting the win probability after the team composition of each team are set by using, just before the match starts (Semenov et al., 2016). The aforementioned DotA studies do not distinguish between the timing of occurred in-game events to draw conclusions.

Another research paper which dived into the analysis of successful strategies for League, is from Do Nascimento et al., (2017). In this paper, different behavioural profiles of teams were categorised and for each profile, highly correlated post-game statistics were registered. Essentially, the post-game statistics which fit a high win rate and a low win rate were distinguished to conclude which statistics seemed the most relevant for a high win rate. However, this paper uses static post-game statistics and thus does not differentiate between the timing of occurrence of different in-game events. Another paper which also checks team composition strategies is from Costa et al., (2019), where a Genetic Algorithm was used to compose team compositions by selecting champions which all would fit a certain playing strategy. There has also been done some research about victory prediction. However, the data gathered and used is fairly basic. Examples of such papers are papers from Deja & Myslak, (2015) and Ani et al., (2019).

There are multiple other research papers which identify winning strategies and other behavioural patterns. Many of them use post-game data which does not differentiate between the timing of occurred events. Most studies look if something happened, but not when such an occasion occurred. However, some articles use this differentiation to some extent. The research in Eaton et al., (2017), checks the importance of the survival of a team's Carry, which is the most important player who often does the most damage. This is analysed by checking when either Carry of a team dies, and what effects these events have on the amount of kills, assists, and buildings destroyed. Even though the timed sequence of events of a match are taken into account, this paper does not provide some sort of advice in the form of strategies or behavioural patterns, since it seems obvious that the team who kills the other team's Carry the most is extremely likely to win.

Another paper (Kho et al., 2020) also takes into account this sequence of events principle. However, the focus is on objectives mainly. It analyses teams from professional eSports League scenes from different regions and compare sequence of events with regard to objectives result in victories. This objective focussed analysis is relatively unique compared to other articles. The paper classifies 6 different objective events, which are first turret or first dragon for example, and then looks which objective events happen most often for winning teams in the concerning region. The analysed events are shown in Figure 5.



Gameplay/Objective	Explanation
First Blood	The first kill in the game.
First Turret	The first tower to fall in the game.
First Dragon	The first elemental drake secured in the game.
Rift Herald	Rift Herald secured in the game.
Gold Ahead at 20 Minutes	Gold lead exists at the 20th minute mark of the game.
First Baron Nashor	The first Baron Nashor secured in the game.

Figure 5: The in-game objectives identified to keep track of in eSports matches. Extracted from Kho, L. C., Kasihmuddin, M. S. M., Mansor, M. A., & Sathasivam, S. (2020). Logic mining in league of legends. *Pertanika Journal of Science and Technology*, *28*(1), 211–225.

However, this paper uses fairly basic sequence of events analysis because it only checks when the first objective is taken by either team. This could be elaborated by checking more data with regard to objectives. Also, every team of a region is analysed and allocated towards the whole region. Thus, individual teams are not analysed, let alone individual eSports players. In short, the depth of the sequence of events and the individuality of players has a lot of room to be researched on.

The final paper discussed (Kho et al., 2020) suggested that there is enough research to be done in the field of the sequence of events with relation to objectives. Objectives have a large impact on the game but not a lot of academic research has been done to study this extensively. Winning patterns for the sequence of events of objectives could be very useful knowledge to help teams improve. Especially an automated method would be very useful because the game League itself changes significantly after every year when the new season arrives, and also after patches which happen roughly every half a month.

Overall, the academic research analysis concludes that the amount of research done in the analysis of in-game objectives and the timed sequence of events is not researched extensively. This further supports the requirement list which was constructed after the orientation on current available data visualisation methods.

However, the academic theory orientation also helped extending requirements, since common characteristics can be found among the methods done. For example, it was stated multiple times in the aforementioned literature that the Riot API portal was used to access match data online, and this gives the impression of being a reliable online source for collecting data of League matches played. Visualised data in the form of a dashboard is also used multiple times (Afonso et al., 2019a; Afonso et al., 2019b), and shows that a dashboard has added value. Other aspects, such as quality of life additions, could be realised by the customisability of the input and data visualisation. As a side note, in League, there are different types of game modes. For example, a casual game mode can be played where the results do not matter after a win or loss, or a ranked game mode can be played which will impact a player rank depending on whether the player won or lost. Matches which are considered serious have different playstyles than matches which are considered casual.

Thus, the new method should be able to make a distinction in game modes when analysing a player's match history.

The full requirement list with its new requirements is as follows. Categories are added to add overview:

- Data collection
    - The method should extract data from a reliable online source about League
    - The method should be flexible to analyse different players, from every region
    - The method should change its sample data of matches analysed if the user wants to
    - The method should only collect data from matches played which are considered serious
- Data analysis to identify winning patterns
    - The method should focus on the importance of in-game objectives with relation to winning
    - The method should differentiate between early and late events in matches (sequence of events)
- Ensure user-friendliness
    - The method should have settings to view categories of analysed match data
    - The method should use visualisation like graphs on a dashboard to help create overview
    - The method should run as automatic as possible

The requirements list yields a valuable research opportunity with a fitting research question, which is:

How can the performance of League eSports players be identified by analysing behavioural in-game patterns with regard to the sequence of events of objectives?

Essentially, the answer to this research question should be able to expand the research on winning strategies and the analysis of player in-game behaviour. To make sure this happens, three main aspects should be answered. First of all, it is important to check which data is needed to answer the research question, and how it can be collected. Most likely, certain data of League itself should be collected to analyse behavioural in-game patterns of players during a match, such as with the aforementioned Riot API online portal. Second of all, the gathering of the data should be able to be stored. Then, a method should be constructed to analyse the data. Perhaps certain software can function as a way to store and analyse the extracted data. Third of all, the analysed data should be translated and insightful conclusions should be able to be drawn from them. This should result in an advice about the performance of the player and would thus identify winning strategies. Meeting the requirements of the requirements list will result in a tool which achieves the aforementioned sub-goals, and thus would create an innovative method or tool which achieves the required needs. For visualisation purposes, the figure with the requirements is shown in Figure 6, and will be the blueprint for the actual development and creation of the tool which will be done in Chapter 3.

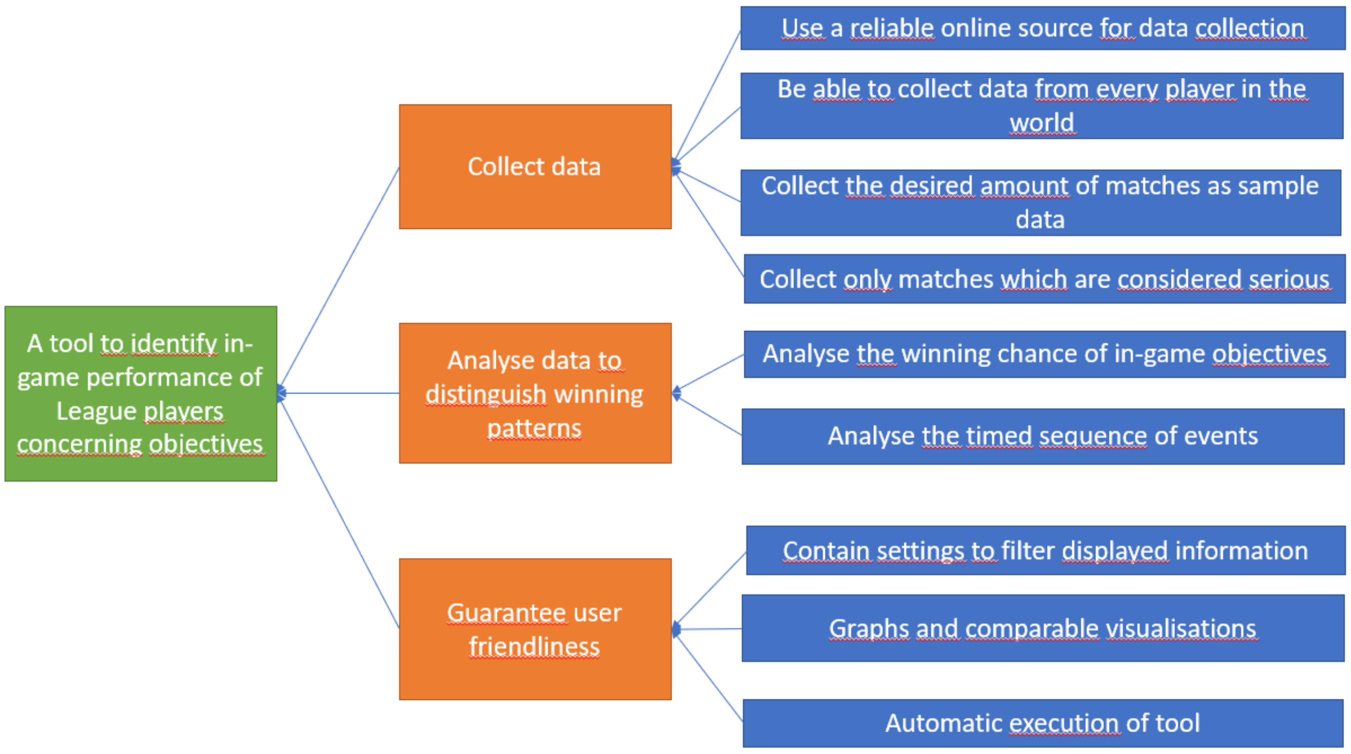

A tool to identify in-game performance of League players concerning objectives
Collect data
Analyse data to distinguish winning patterns
Guarantee user friendliness
Use a reliable online source for data collection
Be able to collect data from every player in the world
Collect the desired amount of matches as sample data
Collect only matches which are considered serious
Analyse the winning chance of in-game objectives
Analyse the timed sequence of events
Contain settings to filter displayed information
Graphs and comparable visualisations
Automatic execution of tool

Figure 6: Visualised overview of blue requirements list

Chapter 3: Method of approach

The aforementioned research opportunity was that research with relation to game analysis to achieve performance optimisation in League can be beneficial. As mentioned in chapter 1, such research would be significant due to the growth of the eSports scene in League, and can be achieved with insight into the performance of a player. In this chapter is explained how the research question posed in chapter 2 is answered.

The analysis of behavioural in-game patterns could be achieved by creating a tool. The requirements, made in chapter 2, of such a potential tool are shown in Figure 6. The orange boxes are the categories and the blue boxes are the requirements which can be used as a checklist. Each category in this chapter has its own heading and the aspects of the tool which would fit each category are explained. The end goal is to create a tool which is able to identify in-game performance of League players concerning in-game objectives. This chapter is about how each requirement is translated to an aspect of the tool. In Appendix B, a more expansive and technical description about the tool itself is provided.

Collect data

The first category is "Collect data". This refers to how data is collected and how it is stored.

The first requirement in this category is to use an online source for data collection. Mentioned data collection methods in chapter 2 are, among others, a Websocket or the Riot API. The latter is used by most of the analysed academic sources in chapter 2. The Riot API is freely accessible to all players and is able to show in-game match data in the form of structured documents, which highlights the impression of being a reliable online data source.

The second requirement is that the data collection method should also be able to collect data from every player in the world. Every player plays in its own region, and is not matched with people from other regions. The Riot API is able to collect data from different regions, which makes it a flexible data collection source. Thus, using the Riot API for data collection would be a reliable option and also able to collect data from every player.

The third requirement is that the tool should be able to collect a desired amount of matches, which would be inputted by the user. This should be a customisable option in the tool to provide the user the possibility to analyse different amounts of data. The fourth requirement also refers to data filtration. This requirement states that only data in the form of matches should be included which are considered 'serious'. Serious matches need to be classified as matches where all participants had maximum effort and all played to win, in contrast to playing casually for fun. 'Ranked' games are the most serious game types because the outcome of the match played directly influences the public rank of the player profile. The game type called 'normal' games, are matches which do not influence this public rank and thus are played to a less serious extent. To conclude, the tool should have a customisable match sample size and the ability to filter ranked games from normal games.

The Riot API gives the impression to be able to meet all four requirements of the "Collect data" category, because of the data it offers. The Riot API can tell what has happened every minute within a match. To extract data from the Riot API, a program is needed. Python is a programming language which is able to do this. Hence, the decision will be made to program the data extraction procedure of the Riot API via Python.

In order to collect data of the Riot API in Python, functions have to be defined which extract the match data of the inputted player to analyse when called. The Riot API uses JSON files which can be accessed with an custom URL, which also requires a valid API key to access the URL. The URL also contains the region of the player that is analysed, which means that every player in the world can be analysed by inputting the region of the analysed player. The API key for using the Riot API can be obtained by logging in with a League account and clicking on the button to generate an API key, and this API key iteration expires after 24 hours. Python is able to turn the JSON file into a 'dictionary' file, which makes it possible in Python to extract data. In order to extract the data with an URL, the Python library called 'requests' needs to be imported. Four Python functions will be created to access summoner data, the summoner's match history, match details, and the match timeline. These functions yield all the data to view what happened every minute in a match, because summoner data is required to access the match history, and the match history contains the match IDs of the matches played by the analysed player, which is used to access the match timeline and the events of each minute. Not every match from the match history should be analysed, because only the matches that are deemed as 'serious' matches should be extracted, which are ranked matches. The Python code will only include such matches.

As mentioned before, the URLs which are used to retrieve the API data are customisable, which means that specific data can be recalled if the URL is customised appropriately. Customisable aspects for the URLs are: the region of the analysed player, the summoner name of the player, a valid API key, a match ID, and the amount of matches to be analysed. The region, summoner name, API key, and amount of matches analysed will need to be inputted by the user. The match IDs would have to be automatically iterated through by the code itself, and thus requires no manual input.

The "Collect data" requirements should be covered by the plan for the Python code:

- A reliable online source is used, namely the Riot API portal
- Every player in the world from each region can be analysed by the tool, due to the customisable input for the region
- The amount of matches to be analysed is implemented as user input
- Only matches are collected which are considered serious, since ranked matches are filtered by the code

Once the data is collected, the tool will need to analyse the data by registering objectives and the timing of such events, which relate to the "Analyse data to distinguish winning patterns" requirement category.

Analyse data to distinguish winning patterns

The fifth and sixth requirements both refer to the desired ability of the tool to analyse data to distinguish winning patterns. Successful in-game behaviour is defined as patterns which elicit positive outcomes, which means outcomes that ultimately result in a player winning a match.

The fifth requirement tells that a certain indicator should be provided by the tool which shows which objectives seem to be more desirable. This indicator could be a win ratio, which shows how likely it is for the analysed player to win when a certain objective is chosen. This can be achieved by analysing matches where such an objective is taken and the outcome of each match is registered.

The sixth requirement is that the tool should distinguish events based on their timing during a match. This means that not only the acquired objective itself should be taken into account, but also when it occurred. For example, whenever a tower is either destroyed in the early game or in the late game, it has a different impact on the match. This requirement can be realised by creating a data sheet which not only lists when an objective taken and whether the outcome of the match was a win or loss for the player, but also at what time period this event occurred.

The Python code explained in the ”Collect data” category should be able to gather the data needed for the analysis. More coding is required to register the win rate of taking an objective at what time period. The code should iterate through the filtered match IDs and check what events concerning objectives occurred during the match. The code should also register whether the analysed player won the match. Every match will be scanned by analysing the match timeline JSON file by using a function to access the corresponding URL, namely requestMatchTimeline. The timeline shows what has happened every minute in the match, and thus can be used to register every objective that was taken during a match. Other data, such as kills made or items bought, should be ignored because these are not objective events. Another similar function is used to register which team won the match, which is called requestMatchDetails. This is required to analyse whether the occurrence of an objective event resulted in a victory or defeat for the analysed player. The script should iterate every minute of a match and register all objectives taken by either team in an Excel file which is used for data storage. For each row in this data storage file, the following columns are used:

- Summoner (name of the player’s account)
- MatchID
- Outcome (1 means victory for the analysed player, 0 means defeat)
- Minute
- Period (early, middle, or late)
- Tower (whether a team destroyed a turret)
- Monster (whether a team has slain rift herald, dragon or baron nashor)
- Dragon Soul (whether a team acquired the dragon soul buff)

Every objective event is registered in the Excel data storage file, along with its timing in the form of minutes. This is done by using the Python library called 'openpyxl', which enables Python to put data into an Excel file. The script should also determine whether an event happened in the early, mid, or late game. The mid game is defined as the period from 15 to 25 minutes. The early game occurs before this time window while the late game occurs afterwards. The outcome column will be used to calculate the expected win rate of each occurred objective in the data set, which is done by dividing the amount of victories by the amount of time a match was registered where the corresponding objective event occurred.

A summary table should be made where each objective event and its timing are shown next to its expected win rate, which would grant the user insight into how successful taking an objective is. The Excel data storage file will change whenever a new player will be analysed. In order to make sure that the summary table reselects the new range of data in the Excel data storage file, a VBA code should be created which automates this reselection and data update. The library 'openpyxl' does not allow data to be pasted in a non-macro Excel file. Thus, a separate dashboard Excel file with the summary table should be created which uses the data from the data storage Excel file as input. The summary table should also include the amount of time a match occurred where an objective event happens, as this amount gives an indication of the reliability of the expected win rate. A minimum threshold for this amount should be implemented to avoid exceptional matches as they are not representative. A possible minimum threshold could be 3, which means that only matches should be included with 3 or more occurances.

The "Analyse data to distinguish winning patterns" requirements should be covered by the finished Python code and the plan for the Excel and VBA part:

- The winning probability is analysed of in-game objectives, because a summary table is made in Excel which calculates the expected win rate of each objective event
- The timed sequence of events is analysed, because the filtered match data categorises the objective events according to either the early, middle, or late game

A flow-chart of how the finished Python code works is shown in Figure 7.

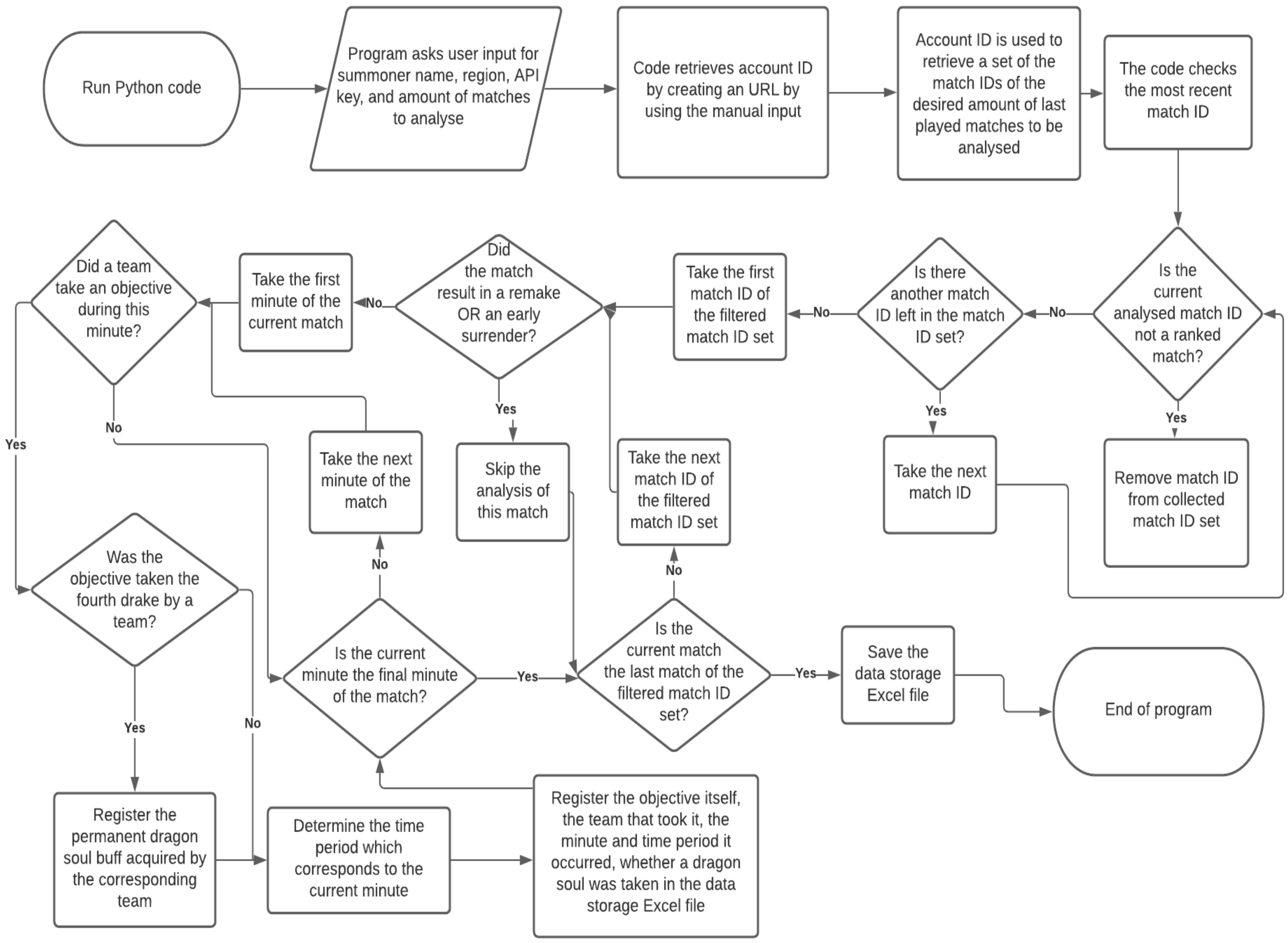

Run Python code
Program asks user input for summoner name, region, API key, and amount of matches to analyse
Code retrieves account ID by creating an URL by using the manual input
Account ID is used to retrieve a set of the match IDs of the desired amount of last played matches to be analysed
The code checks the most recent match ID
Is the current analysed match ID not a ranked match?
Yes
Remove match ID from collected match ID set
No
Is there another match ID left in the match ID set?
Yes
Take the next match ID
No
Take the first match ID of the filtered match ID set
Did the match result in a remake OR an early surrender?
Yes
Skip the analysis of this match
No
Take the first minute of the current match
Did a team take an objective during this minute?
Yes
No
Was the objective taken the fourth drake by a team?
Yes
Register the permanent dragon soul buff acquired by the corresponding team
No
Determine the time period which corresponds to the current minute
Register the objective itself, the team that took it, the minute and time period it occurred, whether a dragon soul was taken in the data storage Excel file
Is the current minute the final minute of the match?
No
Take the next minute of the match
Yes
Is the current match the last match of the filtered match ID set?
No
Take the next match ID of the filtered match ID set
Yes
Save the data storage Excel file
End of program

Figure 7: flow chart of the Python code

Guarantee user friendliness

The final three requirements are about visualisation and user friendliness aspects. The seventh requirement is that information should be able to be filtered. This requirement aims to make sure that customised navigation of the data analysis is possible, and thus only chosen aspects of the data can be showed.

The eighth requirement is to include graphs or comparable visualisations, to make sure conclusions can be drawn from a picture instead of lines of text, which aims to improve visual clarity. Both the seventh and eight requirement can be satisfied by creating a dashboard with customisable graphs and corresponding tables. The tables should be filtered on chosen information by the user and the graphs should adjust accordingly. This provides the user two types of ways to observe the conclusions of the data analysis by the tool.

The ninth requirement should make sure that the tool runs automatically without termination of the program and minimise the actions required by the user when running it.

The ability to filter information displayed by the tool can be implemented by the introduction of an Pivot table in the Excel dashboard file, which should be a replacement for the summary table as discussed in the part “Analyse data to distinguish winning patterns”. A Pivot table categorises data and allows the user to only check for certain data. For example, a Pivot table enables the user to only show objective events which happened in the early game time period, along with its expected win rate. The objective events will be listed under the ‘Situation’ heading in the Pivot table. Furthermore, it is also possible in Excel to add a chart for the Pivot table. Changing which data is shown in the chart automatically updates the Pivot table, which shows that data filtration is also possible in the chart. Thus, a customisable visualisation is added this way, which fulfils both the seventh and eighth requirements.

The final requirement aims to make sure the running of the tool is automated, in so far possible. Since the API key has a limit of being able to be used 100 times per 2 minutes, the tool should pause the program for 2 minutes once the API request counter comes close to 100. A threshold for this could be 98, because every analysed match requires 2 API requests. Once this threshold has been surpassed, then the tool should pause and automatically resume after 2 minutes.

After each run, new data is pasted in the Excel data storage file. The tool should automatically reselect the new data for the Pivot table and graph, which can be achieved with VBA macros in Excel.

The “Guarantee user friendliness” requirements should be covered by proposed interventions:

- Customisable settings to filter displayed information is guaranteed because a Pivot table and graph offers such options
- A visualisation of the data is added in the form of a Pivot chart
- Process automatization of the tool is guaranteed with built-in Python functions and VBA macros

The method of approach is extensive enough to create a prototype, which will be discussed in the “Prototype” heading.

Prototype

The prototype is a practical implementation of the plans proposed in the three aforementioned requirement category headings. As mentioned before, the tool starts off by running the Python code, which asks for user input. The full Python script is shown in Appendix C. This user input is used to analyse the match history of an inputted player, concerning in-game objectives. The tool summarises the data of objective events, called situations in the Excel file, by putting them into a data storage Excel document. Another dashboard Excel document reads this other file and turns the data into a Pivot table and graph, which enables the user of the tool to observe the impact of each analysed objective on the outcome of a match. A flow chart of how the tool works is shown in Figure 8.

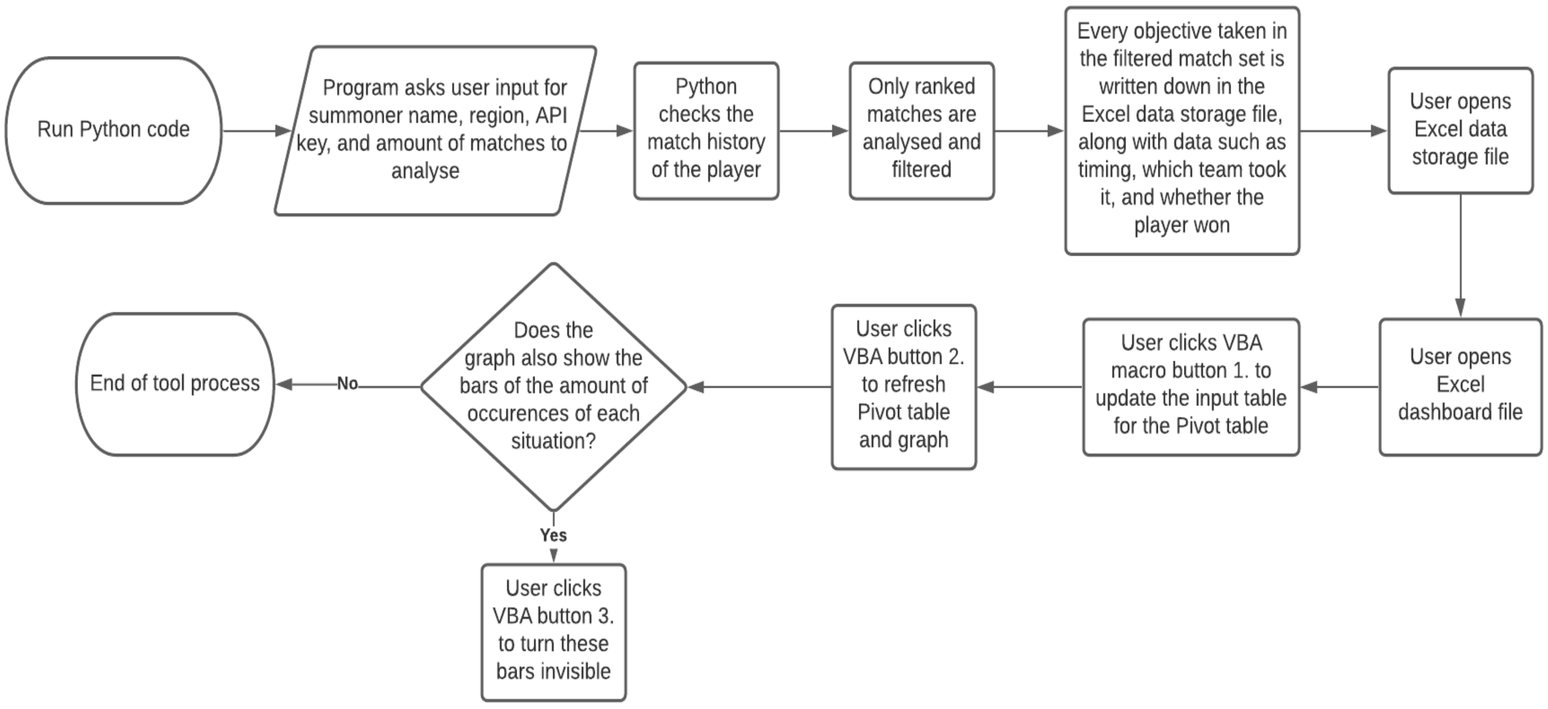
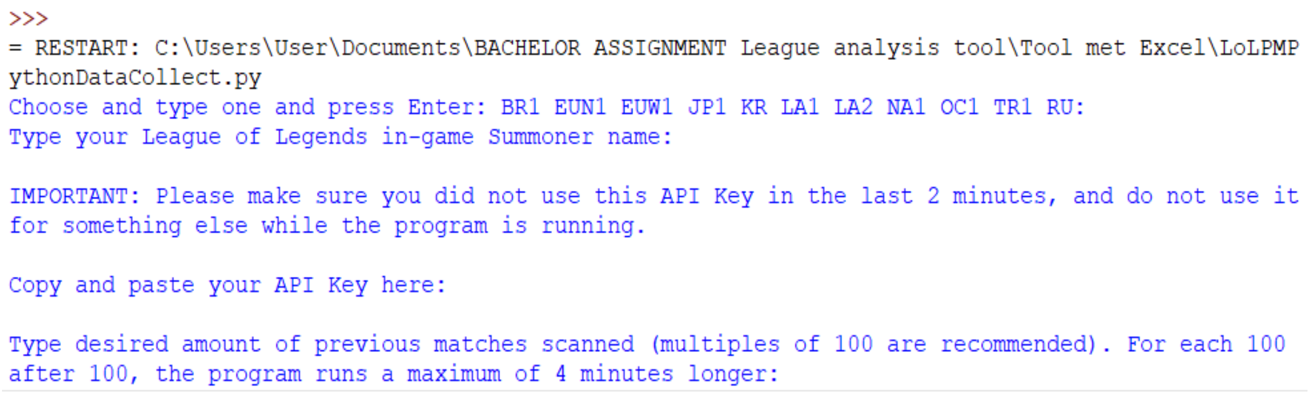

Run Python code
Program asks user input for summoner name, region, API key, and amount of matches to analyse
Python checks the match history of the player
Only ranked matches are analysed and filtered
Every objective taken in the filtered match set is written down in the Excel data storage file, along with data such as timing, which team took it, and whether the player won
User opens Excel data storage file
User opens Excel dashboard file
User clicks VBA macro button 1. to update the input table for the Pivot table
User clicks VBA button 2. to refresh Pivot table and graph
Does the graph also show the bars of the amount of occurences of each situation?
No
End of tool process
Yes
User clicks VBA button 3. to turn these bars invisible

Figure 8: Flow chart of how the tool works

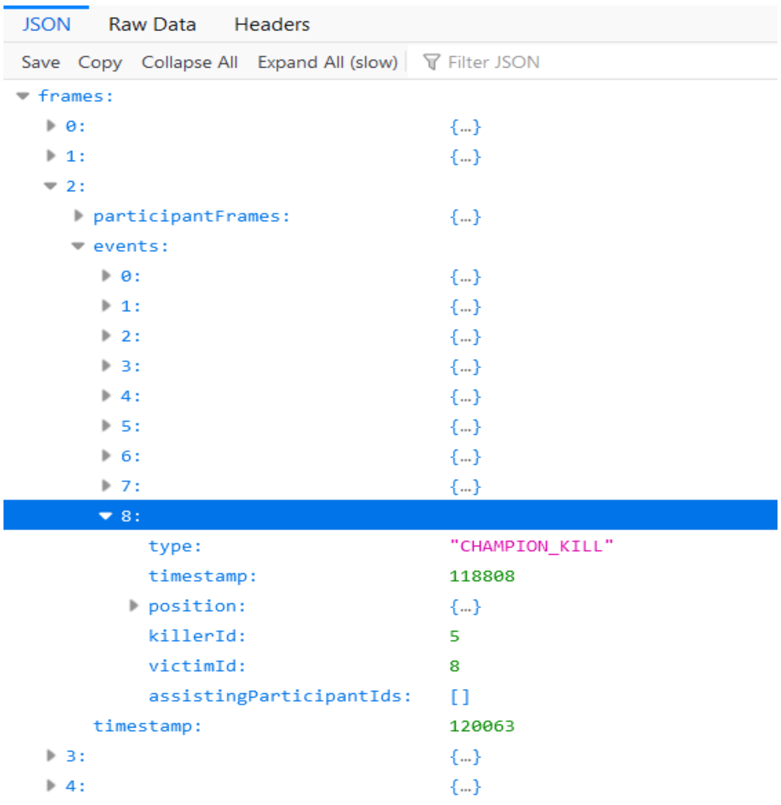
Running the Python code yields the interface as shown in Figure 9. The user types their desired input and presses the Enter key to make the program save the input. As mentioned before, the input is the region of the player to be analysed, the summoner name of the player's account, a valid API key for the Riot API, and the amount of previous matches to be scanned.



```
>>>
= RESTART: C:\Users\User\Documents\BACHELOR ASSIGNMENT League analysis tool\Tool met Excel\LoLPMP
ythonDataCollect.py
Choose and type one and press Enter: BR1 EUN1 EUW1 JP1 KR LA1 LA2 NA1 OC1 TR1 RU:
Type your League of Legends in-game Summoner name:

IMPORTANT: Please make sure you did not use this API Key in the last 2 minutes, and do not use it
for something else while the program is running.

Copy and paste your API Key here:

Type desired amount of previous matches scanned (multiples of 100 are recommended). For each 100
after 100, the program runs a maximum of 4 minutes longer:
```

Figure 9: Python input interface

The code first accesses summoner data of the inputted account to find the account ID of the account, which is done by extracting data from a JSON file reached by a URL which contains the API key. This account ID is used to collect the match history of the account, which is filled with match IDs. The Python code accesses the API key two times per analysed match. An example of a JSON file accessed by a constructed URL is shown in Figure 10. It shows the timeline of a match, where each frame is a minute, which contains events. Events tell what happened during a minute, such as whether a champion is killed, an item is bought, or an objective is taken. Every event is characterised by a participant ID. Another URL, which is about match details, shows which player fits to which participant ID. Thus, both the match details and match timeline needs to be analysed to analyse one match, which means that the API key is requested twice. The limit API request limit is 100 times per 2 minutes. The Python code will automatically pause and continue if the code would otherwise request the API key above this limit.



```
▾ frames:
  ▸ 0:                              {…}
  ▸ 1:                              {…}
  ▾ 2:
    ▸ participantFrames:            {…}
    ▾ events:
      ▸ 0:                          {…}
      ▸ 1:                          {…}
      ▸ 2:                          {…}
      ▸ 3:                          {…}
      ▸ 4:                          {…}
      ▸ 5:                          {…}
      ▸ 6:                          {…}
      ▸ 7:                          {…}
      ▾ 8:
          type:                     "CHAMPION_KILL"
          timestamp:                118808
        ▸ position:                 {…}
          killerId:                 5
          victimId:                 8
          assistingParticipantIds:  []
    timestamp:                      120063
  ▸ 3:                              {…}
  ▸ 4:                              {…}
```

Figure 10: A JSON file of the timeline of a match

After the Python code has finished running, the Excel data storage file should be opened by the user. The name of this Excel file is shown at the end of the Python interface, which is “LoLPMData”. The first row of this file contains the headings as discussed in the “Analyse data to distinguish winning patterns” part of this chapter. Every next row will write down the data of every objective taken by either team of the scanned set of matches by the tool. The interface will also show the name of the Excel dashboard file, which should be opened only after the Excel data storage file has been opened. This file is called “LoLPMDashboard”. This file contains at the very left a table which transforms the data from the Excel data storage file to a table which is able to be used as input for the Pivot table and graph. Thus, this table is not meant to be looked at by the user. Instead, the Pivot table and graph are the focal point of the data visualisation. They categorise each objective according to the time period in which the objective events occurred. The user is able to adjust which time periods or objective event, or situation, are filtered by the graph, which will automatically update the Pivot table as well. It is important to note that the user should only change the filter settings of the Pivot chart, instead of the Pivot table, because this is the only way to update both the graph and the table. The x-axis of the Pivot chart shows each objective event and each time period, and the y-axis shows the corresponding expected win rate of each of these situations. As mentioned in the “Analyse data to distinguish winning patterns” part, the Pivot table filters out situations with their corresponding time period which only occurred 3 times or less, in order to avoid exceptional scenarios. If preferred by the user, this threshold can be manually adjusted in the value filter of the Situation settings of the Pivot chart. Finally, in order to make sure the tool functions without errors, the Python code, the Excel data storage file, and the Excel dashboard file should be placed in the same folder. Furthermore, the Excel files should be closed when the Python code is running.

Testing of the tool

In chapter 4, the prototype of the tool will be tested in order to check whether the requirements posed in chapter 2 are achieved. An initial run will be executed by analysing one of the accounts of an eSports player, whose alias is Rekkles. This initial run is used as a demonstration to show how a run works. Additional runs and manual actions will be tried to check the reliability of the tool to meet the blue requirements of Figure 6. However, not every requirement will need a sensitivity analysis, since some requirements are already tested in the initial run. For each requirement, the following testing procedures are executed:

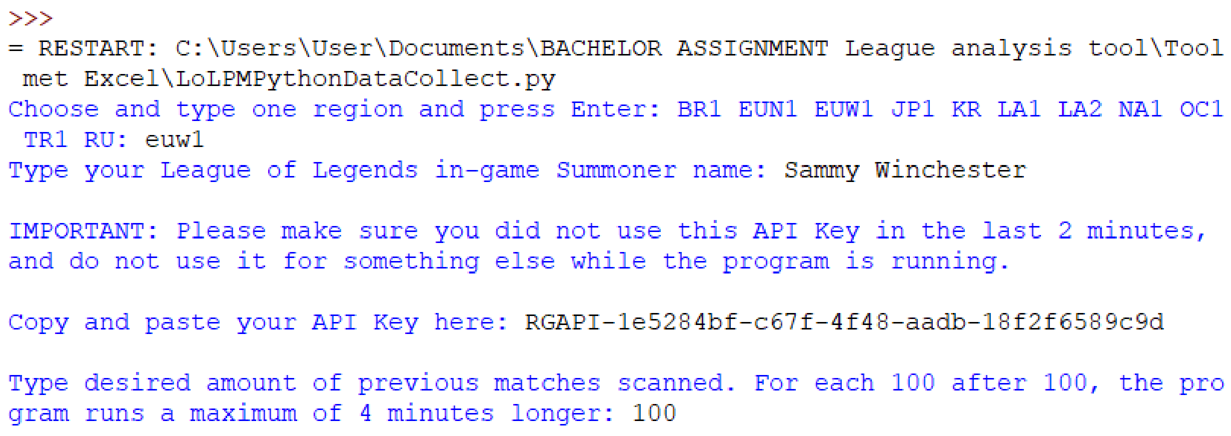
1. None, since the Riot API is already defined as a reliable online source in chapter 2
2. Players from different regions are inputted, also fake accounts
3. The amount of matches differs as input
4. The set of preferred queue types in the Python code, called ‘desiredqueuetypes’, is changed to other match types than only ranked matches
5. None, tested in initial run
6. None, tested in initial run
7. None, settings are adjusted in initial run
8. None, tested in initial run
9. None, tested in initial run

The discussion of the results will take place in chapter 5.

Chapter 4: Results

In this chapter, the tool created in chapter 3 is being tested to check to what extent the requirements posed at the end of chapter 2 are met. The analysis of these results takes place in chapter 5. One of the accounts of the eSports player, whose alias is Rekkles, is used for the initial run demonstration. The account's summoner name is Sammy Winchester.

After the Python code has been activated to run, the interface to input data appears. The interface along with the filled in data is shown in Figure 11. The API key was retrieved from the Riot Developer API Portal by signing in with a League account and generating an API key. The inputted amount of matches to analyse are 100. When observing the match history of the account, it plays approximately 5 matches each day, which means that 100 matches should cover 50 days. This amount of matches is chosen to make sure enough data is gathered to yield logical conclusions, which was tested via trial-and-error.



```
>>>
= RESTART: C:\Users\User\Documents\BACHELOR ASSIGNMENT League analysis tool\Tool
 met Excel\LoLPMPythonDataCollect.py
Choose and type one region and press Enter: BR1 EUN1 EUW1 JP1 KR LA1 LA2 NA1 OC1
 TR1 RU: euw1
Type your League of Legends in-game Summoner name: Sammy Winchester

IMPORTANT: Please make sure you did not use this API Key in the last 2 minutes,
and do not use it for something else while the program is running.

Copy and paste your API Key here: RGAPI-1e5284bf-c67f-4f48-aadb-18f2f6589c9d

Type desired amount of previous matches scanned. For each 100 after 100, the pro
gram runs a maximum of 4 minutes longer: 100
```

Figure 11: Filled in interface of Python code for the account called Sammy Winchester on the EUW server region

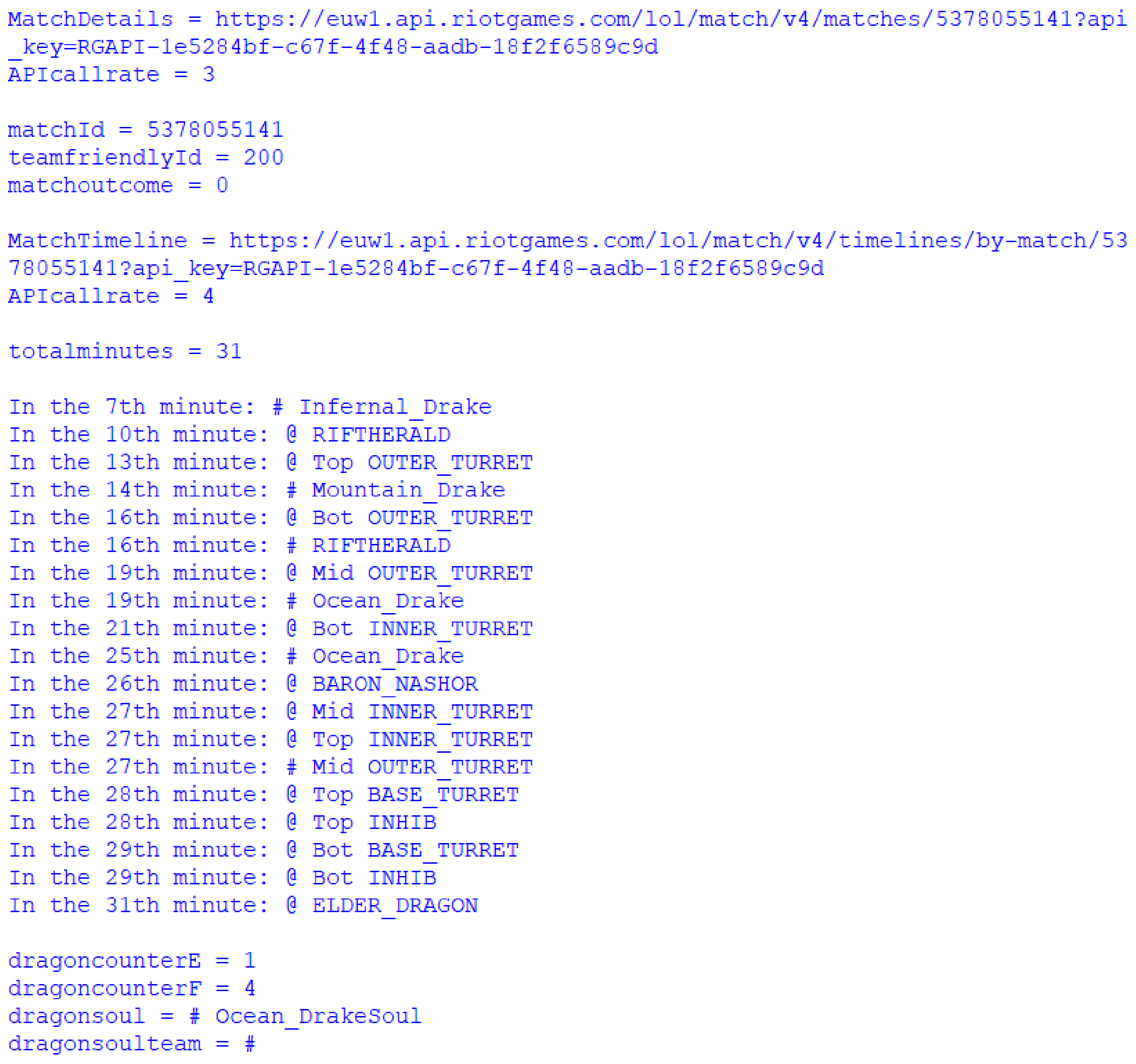
The interface will print data for the user to see, which informs the user about the run. However, this information only shows that the tool is running and it shows what it is currently processing. The data shown in the interface does not provide conclusions for the user with relation to the analysis. Such information will only be shown in the Excel dashboard file.

Pressing the Enter key will continue the Python run and it uses the input data. The tool will start printing the URL, which refers to the corresponding JSON file, of the summoner data and match history of the account. The summoner data is used to find the account ID, and the account ID is used to find match IDs. The script also prints the match IDs of the filtered set of match IDs which are only ranked matches. Furthermore, the script keeps track of how often the API key is used by the program with counter variable 'APIcallrate', which is also printed. The first lines of output printed by the Python file on the interface is shown in Figure 12.

```
SummonerData = https://euw1.api.riotgames.com/lol/summoner/v4/summoners/by-name/Sa
mmy Winchester?api_key=RGAPI-1e5284bf-c67f-4f48-aadb-18f2f6589c9d

APIcallrate = 1
AccID: " IFmZFQ7QiTOr6D8vJz6jnwTiUoGhkNiGvhWHOQFhOkbfYg "

desiredqueuetypes = [420, 440]
(See Python code for more details about queue types)

MatchHistory = https://euw1.api.riotgames.com/lol/match/v4/matchlists/by-account/I
FmZFQ7QiTOr6D8vJz6jnwTiUoGhkNiGvhWHOQFhOkbfYg?endIndex=100&beginIndex=0&api_key=RG
API-1e5284bf-c67f-4f48-aadb-18f2f6589c9d

APIcallrate = 2
matcheswindow = 100

amount of matchesfiltered = 100
matchesfiltered = ['5378055141', '5377704751', '5377702088', '5376558481', '537660
3366', '5376580503', '5376066547', '5374881497', '5374423532', '5373448294', '5373
425207', '5372998246', '5373015159', '5373011523', '5372899113', '5372926536', '53
72895697', '5372903867', '5371705796', '5371588393', '5371545332', '5371505060', '
5371420920', '5371356925', '5371363800', '5370481893', '5370378067', '5370315649',
'5368802359', '5368590867', '5368478548', '5367226260', '5365189944', '5365136715'
, '5362348413', '5362313137', '5362266390', '5362241120', '5362096738', '536214245
5', '5361083474', '5361049372', '5361014404', '5360907228', '5360903476', '5360858
769', '5360804205', '5360832056', '5360739384', '5360736999', '5359464608', '53593
19770', '5359344377', '5359321976', '5359236521', '5357879202', '5357855900', '535
6915369', '5356881306', '5356816687', '5356722365', '5356715796', '5356622030', '5
356322331', '5355090178', '5354814366', '5353993282', '5353898014', '5353005197',
'5353072102', '5352558894', '5352505307', '5352435868', '5352460592', '5352180156'
, '5352117242', '5352027123', '5352011120', '5351936797', '5351951427', '535188572
3', '5351883380', '5351870922', '5351708929', '5351775342', '5351713737', '5351770
694', '5351335725', '5351362139', '5351226773', '5351283105', '5350706553', '53506
28084', '5350543712', '5350488631', '5350484555', '5350442890', '5350349472', '535
0375447', '5350351435']
```

Figure 12: The first lines of codes printed on the interface of the Python file

Each match is scanned to extract every objective event and their timing and put them into the data storage Excel file. The scanning of each match is also printed by the program, as shown in Figure 13. The URLs of the match details and match timeline are printed, as well as the objectives taken during the match. Other details that are printed are the match ID, the team on which the player was (100 is bottom left, 200 is top right), whether the player won (1 is victory, 0 is defeat), total minutes of the match, amount of dragons taken by each team, and which team took the dragon soul, if applicable. The # means the team of the player and the @ means the enemy team. Unbalanced matches, as mentioned in chapter 3, are removed and not scanned.



```
MatchDetails = https://euw1.api.riotgames.com/lol/match/v4/matches/5378055141?api
_key=RGAPI-1e5284bf-c67f-4f48-aadb-18f2f6589c9d
APIcallrate = 3

matchId = 5378055141
teamfriendlyId = 200
matchoutcome = 0

MatchTimeline = https://euw1.api.riotgames.com/lol/match/v4/timelines/by-match/53
78055141?api_key=RGAPI-1e5284bf-c67f-4f48-aadb-18f2f6589c9d
APIcallrate = 4

totalminutes = 31

In the 7th minute: # Infernal_Drake
In the 10th minute: @ RIFTHERALD
In the 13th minute: @ Top OUTER_TURRET
In the 14th minute: # Mountain_Drake
In the 16th minute: @ Bot OUTER_TURRET
In the 16th minute: # RIFTHERALD
In the 19th minute: @ Mid OUTER_TURRET
In the 19th minute: # Ocean_Drake
In the 21th minute: @ Bot INNER_TURRET
In the 25th minute: # Ocean_Drake
In the 26th minute: @ BARON_NASHOR
In the 27th minute: @ Mid INNER_TURRET
In the 27th minute: @ Top INNER_TURRET
In the 27th minute: # Mid OUTER_TURRET
In the 28th minute: @ Top BASE_TURRET
In the 28th minute: @ Top INHIB
In the 29th minute: @ Bot BASE_TURRET
In the 29th minute: @ Bot INHIB
In the 31th minute: @ ELDER_DRAGON

dragoncounterE = 1
dragoncounterF = 4
dragonsoul = # Ocean_DrakeSoul
dragonsoulteam = #
```

Figure 13: An example of the print created by the Python program for each match scanned

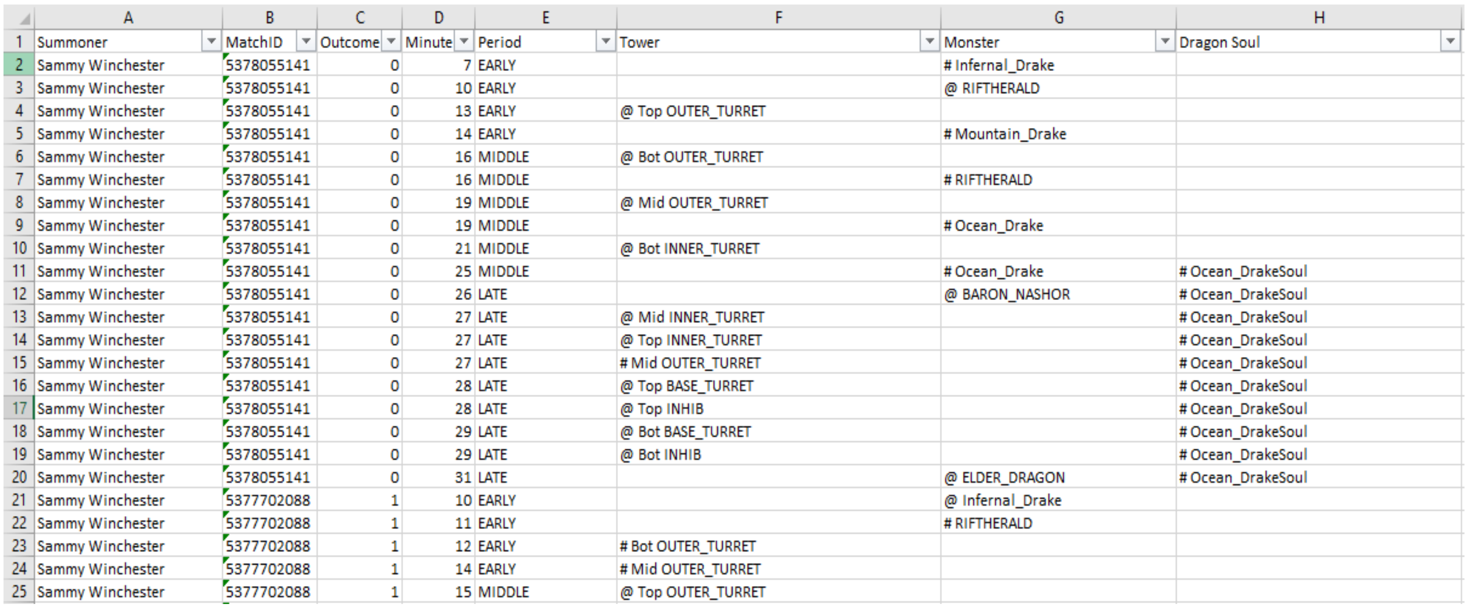
Whenever the amount of API requests crosses 98 requests, the program pauses 2 minutes, because the allowed usage of the API key is 100 times per 2 minutes. In this case, the program will pause and automatically continue after 2 minutes has passed, and the API request counter variable will be reset.

Once the inputted amount of matches are scanned, the program updates the Excel data storage file, and a summary is shown of the matches analysed, as shown in Figure 14. As mentioned, the Excel data storage file needs to be opened first before the Excel dashboard file is opened.

```
Matches analysed = 64
Final APIcallrate = 166
Amount of matches skipped due to unexpected unimportant errors during run = 0
Amount of heavily unbalanced matches removed = 36
Excel raw data table size = A1:H1264



Program finished successfully! First open:
LoLPMData.xlsx

After it has successfully loaded, open:
LoLPMDashboard.xlsm
```

Figure 14: Ending message of the Python program

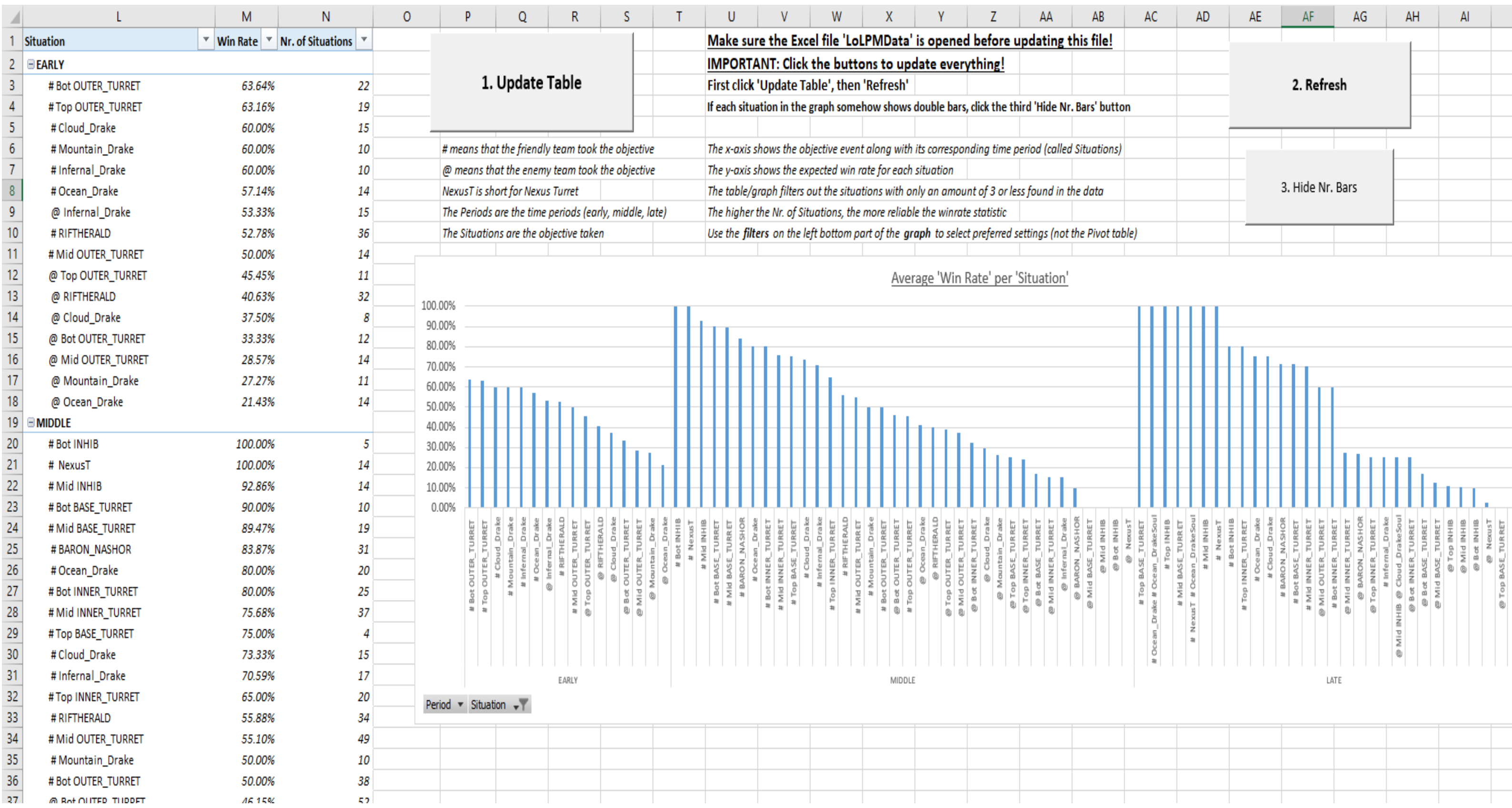
The Excel data storage file which resulted in the run is partially shown in Figure 15. Every objective taken in every match is registered.



	A	B	C	D	E	F	G	H
1	Summoner	MatchID	Outcome	Minute	Period	Tower	Monster	Dragon Soul
2	Sammy Winchester	5378055141	0	7	EARLY		# Infernal_Drake	
3	Sammy Winchester	5378055141	0	10	EARLY		@ RIFTHERALD	
4	Sammy Winchester	5378055141	0	13	EARLY	@ Top OUTER_TURRET		
5	Sammy Winchester	5378055141	0	14	EARLY		# Mountain_Drake	
6	Sammy Winchester	5378055141	0	16	MIDDLE	@ Bot OUTER_TURRET		
7	Sammy Winchester	5378055141	0	16	MIDDLE		# RIFTHERALD	
8	Sammy Winchester	5378055141	0	19	MIDDLE	@ Mid OUTER_TURRET		
9	Sammy Winchester	5378055141	0	19	MIDDLE		# Ocean_Drake	
10	Sammy Winchester	5378055141	0	21	MIDDLE	@ Bot INNER_TURRET		
11	Sammy Winchester	5378055141	0	25	MIDDLE		# Ocean_Drake	# Ocean_DrakeSoul
12	Sammy Winchester	5378055141	0	26	LATE		@ BARON_NASHOR	# Ocean_DrakeSoul
13	Sammy Winchester	5378055141	0	27	LATE	@ Mid INNER_TURRET		# Ocean_DrakeSoul
14	Sammy Winchester	5378055141	0	27	LATE	@ Top INNER_TURRET		# Ocean_DrakeSoul
15	Sammy Winchester	5378055141	0	27	LATE	# Mid OUTER_TURRET		# Ocean_DrakeSoul
16	Sammy Winchester	5378055141	0	28	LATE	@ Top BASE_TURRET		# Ocean_DrakeSoul
17	Sammy Winchester	5378055141	0	28	LATE	@ Top INHIB		# Ocean_DrakeSoul
18	Sammy Winchester	5378055141	0	29	LATE	@ Bot BASE_TURRET		# Ocean_DrakeSoul
19	Sammy Winchester	5378055141	0	29	LATE	@ Bot INHIB		# Ocean_DrakeSoul
20	Sammy Winchester	5378055141	0	31	LATE		@ ELDER_DRAGON	# Ocean_DrakeSoul
21	Sammy Winchester	5377702088	1	10	EARLY		@ Infernal_Drake	
22	Sammy Winchester	5377702088	1	11	EARLY		# RIFTHERALD	
23	Sammy Winchester	5377702088	1	12	EARLY	# Bot OUTER_TURRET		
24	Sammy Winchester	5377702088	1	14	EARLY	# Mid OUTER_TURRET		
25	Sammy Winchester	5377702088	1	15	MIDDLE	@ Top OUTER_TURRET		

Figure 15: The partial output of the Excel data storage file

The data of the Excel data storage file is used to create the Pivot table and Pivot chart in the Excel dashboard file, as shown in Figure 16. The VBA macro buttons need to be clicked as enumerated in the file to update the table and chart. The coding for the VBA macros is shown in Appendix D. The Pivot table and chart show the expected win rate of every objective event and its timing. The Situation heading denotes which concerning objective event is taken, while the Nr. of Situations heading shows the amount of times this objective event occurred in the match data set. The Nr. of Situations of each objective event are filtered to make sure only objective events are included in the analysis which occurred more than 3 times, in order to avoid exceptional scenarios which may be unrepresentative. Data can be filtered to the user's preference by adjusting the filter settings at the bottom left of the Pivot chart. For example, if the user wants to view the expected win rate of taking turrets in the early game, then the user types "Turret" as filter in the Situation settings filter on the Pivot chart, while also selecting the EARLY filter with the Period settings filter. The result of such adjusted settings is shown in Figure 17.



Situation	Win Rate	Nr. of Situations
⊟EARLY		
# Bot OUTER_TURRET	63.64%	22
# Top OUTER_TURRET	63.16%	19
# Cloud_Drake	60.00%	15
# Mountain_Drake	60.00%	10
# Infernal_Drake	60.00%	10
# Ocean_Drake	57.14%	14
@ Infernal_Drake	53.33%	15
# RIFTHERALD	52.78%	36
# Mid OUTER_TURRET	50.00%	14
@ Top OUTER_TURRET	45.45%	11
@ RIFTHERALD	40.63%	32
@ Cloud_Drake	37.50%	8
@ Bot OUTER_TURRET	33.33%	12
@ Mid OUTER_TURRET	28.57%	14
@ Mountain_Drake	27.27%	11
@ Ocean_Drake	21.43%	14
⊟MIDDLE		
# Bot INHIB	100.00%	5
# NexusT	100.00%	14
# Mid INHIB	92.86%	14
# Bot BASE_TURRET	90.00%	10
# Mid BASE_TURRET	89.47%	19
# BARON_NASHOR	83.87%	31
# Ocean_Drake	80.00%	20
# Bot INNER_TURRET	80.00%	25
# Mid INNER_TURRET	75.68%	37
# Top BASE_TURRET	75.00%	4
# Cloud_Drake	73.33%	15
# Infernal_Drake	70.59%	17
# Top INNER_TURRET	65.00%	20
# RIFTHERALD	55.88%	34
# Mid OUTER_TURRET	55.10%	49
# Mountain_Drake	50.00%	10
# Bot OUTER_TURRET	50.00%	38

1. Update Table

Make sure the Excel file 'LoLPMData' is opened before updating this file!

IMPORTANT: Click the buttons to update everything!

First click 'Update Table', then 'Refresh'

If each situation in the graph somehow shows double bars, click the third 'Hide Nr. Bars' button

2. Refresh

3. Hide Nr. Bars

# means that the friendly team took the objective

@ means that the enemy team took the objective

NexusT is short for Nexus Turret

The Periods are the time periods (early, middle, late)

The Situations are the objective taken

The x-axis shows the objective event along with its corresponding time period (called Situations)

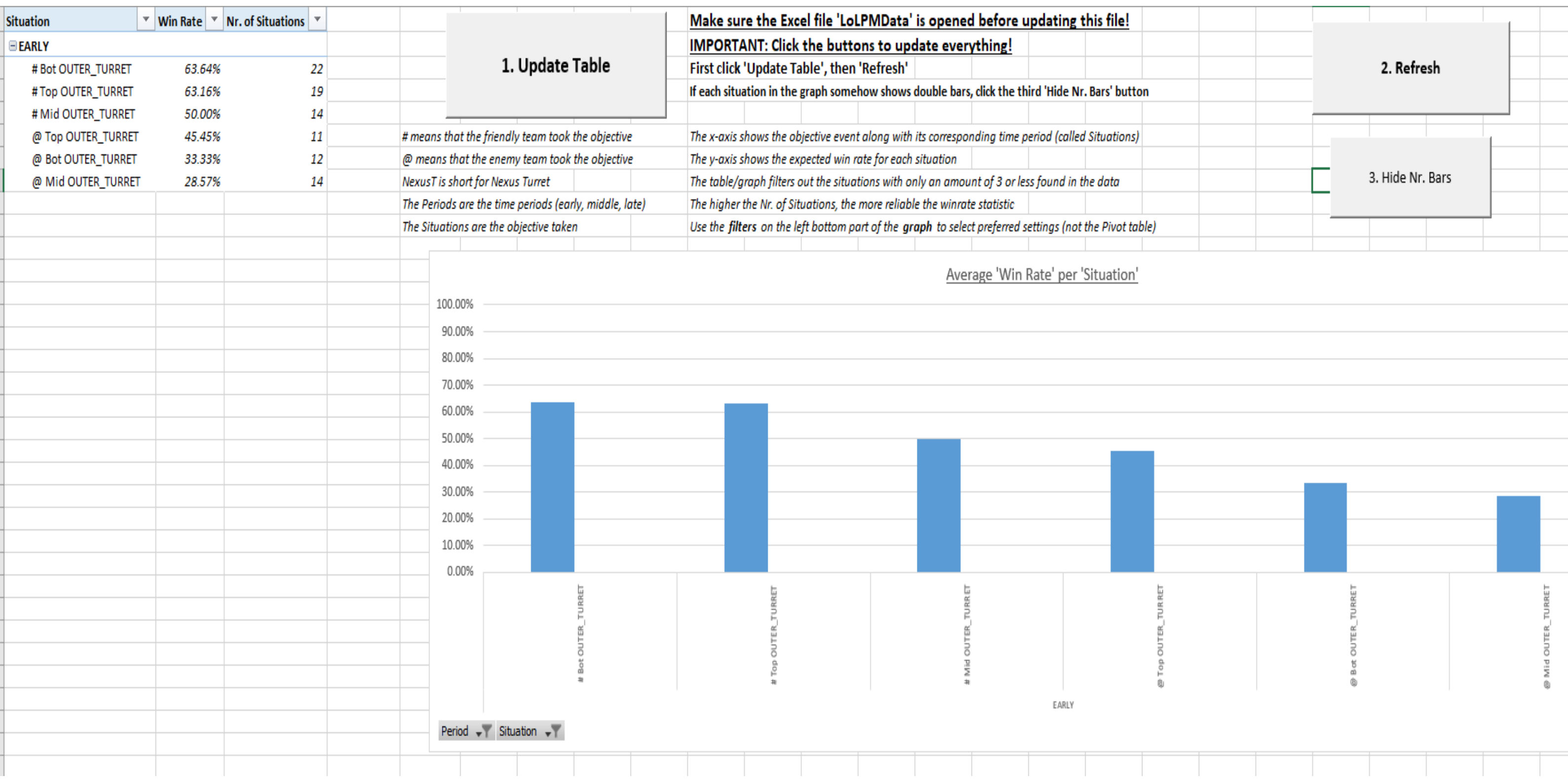
The y-axis shows the expected win rate for each situation

The table/graph filters out the situations with only an amount of 3 or less found in the data

The higher the Nr. of Situations, the more reliable the winrate statistic

*Use the **filters** on the left bottom part of the **graph** to select preferred settings (not the Pivot table)*

Figure 16: The output of the Excel dashboard file



Situation	Win Rate	Nr. of Situations
⊟EARLY		
# Bot OUTER_TURRET	63.64%	22
# Top OUTER_TURRET	63.16%	19
# Mid OUTER_TURRET	50.00%	14
@ Top OUTER_TURRET	45.45%	11
@ Bot OUTER_TURRET	33.33%	12
@ Mid OUTER_TURRET	28.57%	14

1. Update Table

Make sure the Excel file 'LoLPMData' is opened before updating this file!

IMPORTANT: Click the buttons to update everything!

First click 'Update Table', then 'Refresh'

If each situation in the graph somehow shows double bars, click the third 'Hide Nr. Bars' button

2. Refresh

3. Hide Nr. Bars

# means that the friendly team took the objective

@ means that the enemy team took the objective

NexusT is short for Nexus Turret

The Periods are the time periods (early, middle, late)

The Situations are the objective taken

The x-axis shows the objective event along with its corresponding time period (called Situations)

The y-axis shows the expected win rate for each situation

The table/graph filters out the situations with only an amount of 3 or less found in the data

The higher the Nr. of Situations, the more reliable the winrate statistic

*Use the **filters** on the left bottom part of the **graph** to select preferred settings (not the Pivot table)*

Figure 17: Example of filtered settings in the Excel dashboard file with regards to turrets taken in the early game

Analysis of Rekkles

As mentioned before, the dashboard shown in Figure 16 shows the Pivot table and chart for the account named Sammy Winchester, which is an account of the eSports player Rekkles. An analysis is done to illustrate possible conclusions which the tool can yield.

Rekkles is a player who only plays in the ADC position, which is in the bottom lane. The dashboard shows that he achieves the highest win rate (64%) in the early game when his team takes the outer turret at bot lane in this time period, which is the first line of turret defence of the enemy in the bot lane. This is an indication that Rekkles is playing well against his lane opponent. Another point to note is that his win rate is relatively low (50%) when his mid laner takes the enemy middle lane outer turret, which would indicate that his mid laner is playing well against their lane opponent. These two win rates show that Rekkles is benefited more from playing well as an ADC than when the mid laner of his team is playing well. This may highlight that helping the ADC position get ahead in the early game is a more impactful and beneficial decision than helping the mid lane position. It could also show that Rekkles is a very impactful player and that he can dominate the game and achieve victory for his team if he is doing well in the early game.

Certain objective events result in a 100% expected win rate. These objective events are less suitable to use for drawing conclusions, because the tool cannot make a distinction in importance between two objective events which always resulted in victories according to the match data set. However, the number of occurrences may prove a measure for how reliable the according expected win rate is. For example, taking the enemy nexus turret in the late game yields in a 100% expected win rate. This objective event occurred 35 times in the data set of 100 matches, which is relatively high compared to the other objective events in the late game which yielded the same win rate. These other objective events had around 6 to 11 occurrences in the data set. Thus, taking the enemy nexus turret in the late game yields in the highest reliable win rate. This makes sense because the nexus turrets are the last turrets of defence of the enemy. Destroying them almost always results in also being able to destroy the enemy nexus, and thus winning the match. Hence, it is obvious that nexus turrets should be destroyed, and thus this specific data cannot be used to draw meaningful conclusions. It is more beneficial to check the objective events that result in a high win rate but are not mandatory to win the game (unlike nexus turrets). An example in the data set of Rekkles is that taking the bot lane inhibitor in the middle game results in a 100% win rate according to the data set. As mentioned before, Rekkles plays in the bot lane. This data may highlight that Rekkles is good in achieving victory if he is doing well, and it may show that the performance of his team is less important than his own performance.

Taking as much objectives as possible is an indication that the corresponding team is doing well. However, the tool also gives an indication on whether a certain objective is preferred over another. This does not only apply to turrets, but also to dragons for example. There are 4 types of dragons in the game. The tool shows the win rate of each dragon taken. In Rekkles' data, it shows that taking the cloud drake in any time period yields in a higher win rate than taking the infernal drake. This may highlight the impression that either cloud drakes are more useful than infernal drakes at the moment, or that specifically Rekkles makes better use of the cloud drake than the infernal drake.

Finally, the objective events taken by the friendly team, signified with a #, yield in a higher expected win rate than objective events taken by the enemy team, signified with a @. Taking objectives helps the concerning team. Hence, it makes sense that the expected win rate of the friendly team is lower when the enemy team takes objectives. The tool can also show which objectives are optimal for the enemy to take. This data can also show which objectives taken by the enemy team have the highest probability of making Rekkles lose, which may give an indication of the weaknesses of Rekkles concerning enemy objectives. For example, the data of Rekkles shows a low win rate (28.5%) whenever the enemy team destroys the middle outer turret of his team in the early game. This win rate is lower than when the enemy team destroys the top outer turret of Rekkles' team in the early game, which has a win rate of only 45.5%. This may imply that either the top lane is a less impactful role than the middle lane concerning defeating the team of Rekkles. It could also imply that Rekkles is able to help his middle laner whenever possible, but not the top laner. This would make sense because the middle lane is closer than the top lane to the bot lane, which is where Rekkles plays.

As explained, the data provided by the tool can be used to draw conclusions from an analysed player's playstyle. This gameplay insight can be used to improve in-game decision making concerning objectives, and thus lead to possible performance optimisation. The information provided by the tool can also be used to derive conclusions about the playstyle of the enemy team, which could be used for formulating strategies to counter certain opponents.

Requirements sensitivity analysis

As mentioned at the end of chapter 3, each requirement of Figure 6 is tested in order to analyse the reliability of the tool to fulfil the requirements. Additional testing is required for requirements 2, 3, and 4. The rest of the requirements are already tested by the initial test run. To recap, the second requirement is tested by inputting players from different regions, and also testing fake accounts. The third requirement was tested by varying the amount of matches to be analysed by the tool. The fourth requirement was tested by changing match queue types to analyse.

For requirement 2, inputting a region which does not exists results in an error in Python, which is what the program should do because the inputted region must exist. When a user name is inputted which does not exist, the customised URL yields a JSON file which contains only an error, namely "Data not found – summoner not found".

For requirement 3, inputting only one match to analyse the aforementioned account of the player Rekkles, makes the program work as expected, as only one match is analysed. Inputting a 0 for the amount of matches analysed also does not result in an error for the Python program. However, the Excel files will not work, as errors will appear when these files are opened. Running the initial run for the account of Rekkles again will work as intended, which means that faulty runs do not permanently damage any of the files of the tool. Furthermore, even if all cells of the Excel data storage file have been deleted, the Python program automatically rewrites the headers and assimilates the table. Whenever a higher number of matches to analyse than 100 are inputted in the interface, the tool will also work without errors, because runs have been successfully tested with 200 and 300 matches. However, the user will have to wait 2 extra minutes for every next 50 matches to analyse, because the API request limit is 100 per 2 minutes and a match requires 2 API requests. Hence, the waiting time increases.

For requirement 4, inputting different match queue types will make the tool work without errors. For example, by changing the 'desiredqueuetypes' set in the script to only containing '400', which is the code for normal draft matches, the tool works. For the case when a player does not play any normal draft matches, the tool will treat the input as if 0 matches are analysed. This also occurs when the 'desiredqueuetypes' set is left empty, because no match would fit the match type criterium of the set.

Validation

In order to test the applicability of the tool according to potential users, League players from the Esports Team Twente were sent a survey to fill in. The survey is shown in Appendix E. Every player's account was inputted in the tool to scan their 60 last matches played, of which the normal draft and ranked games are chosen to be analysed. This survey aimed to illustrate the eSports player's opinions of the tool, which could show which aspects of the tool are desired and which parts might need adjustment in future research. A summary of the received answers will be elaborated upon next.

According to the response of the survey, the idea of analysing the effectiveness of taking in-game objectives as well as the dashboard itself were received positively. However, it is also stated that the tool in its current state would seem to be quite difficult to draw strategic conclusions from, because League contains a lot of factors and variables which are correlated. It is claimed that taking a dragon would obviously be a wise decision to make after a teamfight is won, because winning a teamfight would make it easier to take any objective. Thus, whether dragon should be taken also significantly depends on the current state of the match.

Furthermore, it is stated that the tool might not work optimally if only one player uses it. Random players who also play the game could have different ideas on objective decision making than the tool. For example, if the tool shows that certain late game decisions can win the game, and if randomly allocated online teammates experience a bad early game, then they might already start giving up. In such a case, the advice of the tool could not be used to its full extent when the user is playing alone, and this shows that the tool would be more useful if all teammates use it instead. However, this would not be a problem if the tool is used for eSports teams, because in this case, the whole team decides to follow the instructions of the tool.

It was stated that the sample size of the survey dashboard was deemed too small to draw conclusions from the data regarding decision making with regards to objectives. The reason for this is that the last 60 matches played are scanned, instead of the last 60 normal draft and ranked matches. Thus, if the analysed user played a lot of other match queue types, then the final amount of matches to be analysed may be small. However, the sample size can be increased to the user's preference in the tool, which makes sure the amount of analysed matches of the desired match queue type is achieved.

Furthermore, according to the response of the survey, it is recommended that only ranked solo queue games are taken as input for the tool, as the skill difference could be too big in other queue types. A recommendation of analysing at least 30 ranked games is stated, as well as adding the analysis of how each objective is taken, instead of only declaring that it is taken, which could make the tool analysis more detailed. It is also stated that the champions chosen on each team could be included in the analysis, because every champion has different advantages. For example, some champions are better at taking dragon.

Other received tips are that different events than objectives could also be taken into account. An example raised is that finding a pattern in enemy ward placement could help your team avoid being spotted by the wards if there is an accurate expectation of where the enemy wards will be placed.

The overall conclusion was that the idea of analysing the success of taking objectives could be useful to eSports teams, but it is recommended that more factors should be taken into account. Furthermore, the user friendliness of the dashboard gives the impression of being sufficient according to the response of the survey, because the reactions were positive overall and no significant changes are proposed. Since the survey specifically asked about the user friendliness, a few options are listed, such as the possibility of changing the colour of the bars of the Pivot chart depending on their win rate.

Chapter 5: Discussion and conclusion

In this chapter, the results posed in chapter 4 are analysed to form conclusions and to discuss potential aspects of improvement. The results are analysed to check whether the requirements posed at the end of chapter 2 are satisfied. To recap, the requirements diagram is shown again in Figure 18.

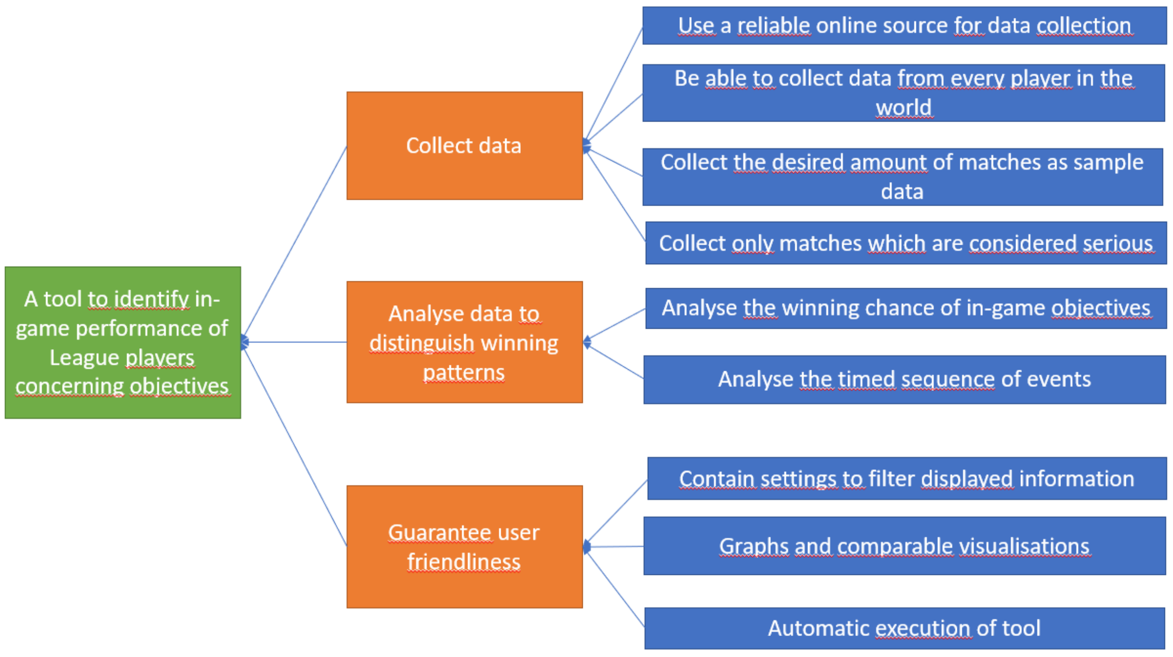

A tool to identify in-game performance of League players concerning objectives
Collect data
Use a reliable online source for data collection
Be able to collect data from every player in the world
Collect the desired amount of matches as sample data
Collect only matches which are considered serious
Analyse data to distinguish winning patterns
Analyse the winning chance of in-game objectives
Analyse the timed sequence of events
Guarantee user friendliness
Contain settings to filter displayed information
Graphs and comparable visualisations
Automatic execution of tool

Figure 18: Visualised overview of blue requirements list

Collect data

The first requirement was that the tool should have a reliable online data collection source. This requirement was already satisfied in chapter 2 by analysing academic data. The Riot API portal qualified for the requirement, which means that the tool satisfies this requirement.

The second requirement was that the tool should be able to collect data from every player in the world, which means that every server region should be accessed by the tool. In the Python part of the tool, the input interface asks the user to input a region. As mentioned in chapter 4, the Python code is able to function by inputting every possible region. Furthermore, the tool only accepts eligible summoner names that exist, which shows that the tool is consistent. Consequently, this shows that the second requirement is satisfied.

The third requirement was that the tool should be able to gather the amount of matches to scan as inputted by the user. In chapter 4 is shown that both the scenarios where an amount of 0 or 1 match is inputted, as well as an amount of 300 works. However, no tests have been run on an amount of matches higher than 300, because increasing the match data set will not create a representative image on what objectives are most important, because the game is updated around every 2 weeks in the form of patch updates. Choosing a match data set which would cover matches from multiple different patch versions could possibly lead to inconsistent conclusions. If the analysed player plays on average only one match per day, then the previous 300 matches of a player will cover more different patch versions than for a professional eSports player who would play 5 matches per day on average. Still, for such an eSports player, a match set of 300 matches covers 60 days and this data set would also contain matches of multiple different patch versions. Inputting a higher amount would increase the amount of patch versions in the data set, and thus the results would be less consistent. Hence, the tool has been adequately tested for reasonable match amounts, which means that the third requirement is satisfied.

The fourth requirement was that only matches should be gathered by the tool which are considered serious, which is classified as ranked matches as mentioned in chapter 3. The initial run of Rekkles' account in chapter 4 showed that the tool works with these settings. As tested, inputting different queue match types codes in the 'desiredqueuetypes' set will also enable the tool to run successfully, which demonstrates that the fourth requirement is satisfied.

Analyse data to distinguish winning patterns

The fifth requirement was that the winning chances of taking in-game objectives by either team should be analysed. The initial run of chapter 4 showed the expected win rate of such objectives in the Excel dashboard file with its Pivot table and chart, and thus this requirement is satisfied.

The sixth requirement was that the timed sequence of events with relation to the objectives is analysed. The initial run of the tool showed that the objectives are categorised for each time period, which demonstrates that the sixth requirement is satisfied by the tool.

Guarantee user friendliness

The seventh requirement was that the tool should have adjustable settings with regards to the displayed information in the dashboard. The initial run of chapter 4 showed that filter settings with regards to each objective event and each time period can be implemented, as shown in Figure 17. The objective events and their time periods can also be filtered on the amount of times they occurred in the match data set, which can be adjusted in the Situation settings filter on the bottom left of the Pivot chart. Thus, the seventh requirement is satisfied.

The eighth requirement was that visualisations such as graph should be added. As shown in Figure 16 and Figure 17, the Pivot chart works and adjusts accordingly to filters, which shows that this requirement has been satisfied.

The ninth requirement was that the tool should be executed automatically. The initial run in chapter 4 showed that the Python file automatically pauses when the tool is reaching the API request limit. The initial run also showed that clicking the VBA macro buttons in the Excel dashboard file will update the data accordingly. Even though the user still has to click buttons, the coding covers the updating procedure. Besides the clicking of the macro buttons, the tool has automated the other processes, which satisfies the ninth requirement.

Conclusion

In this chapter, each orange requirement category in Figure 18 has shown to be satisfied. This means that the green end goal to create a tool which identifies in-game performance of League players concerning objectives has been achieved. The tool could be used to analyse every player in the world, which provides insight in the impact of objectives taken on their chances of victory or defeat. This insight in gameplay could be used to achieve in-game performance optimisation for the analysed person, but also for analysing the enemy and constructing counter strategies.

Discussion and limitations

The tool is able to categorise objective events in time periods and show the expected win rate, but there are more events that happen in a match, such as kills made or wards placed. Matches could also have a different time classification than the early, middle, and late game. However, it is important to note that the amount of events and time periods should not be too detailed, such as having every minute used as time period. Essentially, every minute played in a match of League is unique because there are a lot of variables in play, such as players, their rank, how they play, and current patch version. This is why it is important to generalise data to a certain extent, such as creating the three time periods as used in this thesis. Otherwise, the tool would draw conclusions from in-game scenarios which only happened once, which makes it susceptible for generalising such rare scenarios.

Furthermore, the tool only checks the win rate when a certain objective event along with their timing occurs. The tool does not check the increase in in-game experience or gold of either team whenever an objective is taken. Measuring the direct impact of taking an objective could be a possible point of improvement for the tool. This could be done by defining different degrees of advantages gained from taking objectives. An advantage could be defined by a gold advantage. The Riot API registers the gold of each team every minute. Thus, for example, the tool could be reprogrammed to define a gold advantage within a certain amount of minutes after a team has taken an objective, and base the success of taking objectives on the size of the gold advantage.

Analysing the highest ranked players in a region to find which objectives seem most important in that level of play is not possible with the current iteration of the tool. Such an extension could be used to learn about the most effective tactics available, abbreviated as META, concerning objectives. The META changes after every patch version update. Thus, analysing the META with a tool could lead to performance optimisation. The tool also does not classify matches according to their patch version or season, which means that differing METAs might overlap in the analysis of the tool, and thus may draw inconsistent conclusions. The season changes every year and is a yearly overhaul of the game. Hence, combining matches from different patch versions in the tool might be acceptable, but combining matches from different seasons would create conclusions which combine different seasonal METAs, which would be undesirable.

The tool uses unofficial definitions for determining each time period. For example, the middle game is not necessarily defined by the time between 15 and 25 minutes, but it is used as a guiding measurement. Other definitions of when each time period starts could be used when adjusting the tool. Whenever the early game transitions to the middle game differs per match, and differs per META.

The user friendliness could be examined better in the development of the tool. The tool is not fully automated and it requires three separate files to function, namely one Python file and two Excel files. However, as shown in the validation in chapter 4, the survey gave the impression that the dashboard seemed to have a sufficient user friendliness. Still, this validation was executed in hindsight. The design of the tool was not based on initial needs and requirements posed by potential users such as eSports players.

Finally, the tool uses the Riot API, which is restricted to providing information of what happened after every minute. The API registers where every player is situated on the map, but since this is only registered every minute, it is relatively harder to track how a player walks when compared to positional data which is refreshed every second.

Chapter 6: Further research

This chapter is about the potential to execute new academic research by using the conclusions of the research of this thesis as a basis. Chapter 5 showed aspects which could be improved. Further academic research could be executed to improve some of these aspects, which will be elaborated upon in this chapter. However, not all aspects discussed in chapter 5 needs academic research to solve them, as some of them can be achieved by relatively small adjustment in coding of the tool. The limitation aspects that were named in chapter 5 were the following:

1. The tool is only applied on objectives, not other events such as kills or ward placement.
2. Only the win rate is checked per objective taken. The tool does not check to what extent the in-game experience and gold of either team is affected.
3. Players to analyse has to be inputted sequentially after each run, instead of simultaneously which could be used to analyse a whole eSports team at once.
4. Analysing a sample of players from a certain rank to check the META concerning objectives is not possible.
5. Matches are not classified in patches or seasons. The tool does not make a distinction between different versions of the game, and thus combines matches of different versions, which could lead to inconsistent conclusions and insights
6. Time periods use static measurements, even though time periods are flexible in practice.
7. User friendliness is not necessarily guaranteed because no potential users were interviewed in order to compile requirements.
8. The tool is not fully automated, because the user still has to click buttons in order for the tool to work.
9. The tool only analyses what happens every minute, instead of every second. This is a limitation of the Riot API.

The tool was designed to analyse the gameplay of one specific player concerning in-game objective decision making. The first limitation of the current version of the tool is that it applies to objectives, but not to other events that happen in game. As mentioned in the survey response in chapter 4, it could be beneficial to track the warding placement and prepare counter strategies. The Riot API provides information on warding and other events such as kills. The Python code could be reprogrammed to keep track of other events than objectives, and analyse which events are the most favourable for achieving victory, which would be done by pasting this data in the Excel data storage file. Instead of focussing on events, the tool could also be reprogrammed to focus on champions, such as analysing favourable objective event decision making for specific champions. Each champion has a different playstyle. Thus, by analysing what makes each champion win a match, it could be possible to optimise the way a champion is played. However, this change in focus requires significantly changing the Python code, as well as the Excel files, which means that this intervention is more extensive than changing the type of events to analyse.

The second limitation of the tool is that it only checks whether the occurrence of an objective leads to victory. It only focusses on the final outcome of the match, instead of the direct impact of taking an objective on the game. This limitation could also be relatively simply solved by reprogramming the Python code to categorise objective events in terms of gold advantage in the following minutes by the concerning team. Gold advantage categories such as “small”, “medium”, and “large” could be defined in Python, where each category has its own weight. A higher weight would mean more favourable. Every objective event that occurred in an analysed match sample data could be weighed against each other in Excel, where the success of each objective event could be shown in a graph.

The third limitation is that the tool currently uses one player as input to analyse, which means that teams cannot be analysed simultaneously as a whole. The tool requires relatively little adjustment to make this possible. The Excel data storage file could be filled with multiple accounts, which can be achieved by adjusting the Python code with for loops and adjusted user input, because multiple names would have to be inputted.

The fourth limitation could also be addressed by adjusting the input of the Python file. The Python interface should ask for the desired rank of players to analyse, as well as the amount of players in that rank to analyse. The API can gather random players from the inputted rank and it could analyse all of them at the same time by inputting each player in the data storage Excel file. However, either the amount of matches analysed or the amount of players analysed increases the waiting time for the tool to finish.

The fifth limitation can be fixed by registering the season and patch version of every match analysed. This is registered in the match details JSON file of the API. Categorising matches by patch and season would make sure that the there are no overlapping METAs among analysed matches.

The sixth limitation could be researched by defining in-game patterns which signify the transformation of the early game to the mid game, as well as the mid game to the late game. Usually, the early game is defined when every player, except the jungler, sticks to their lane and remains close to the outer turret of their lane. The API registers the position of every player at every minute. Thus, for example, the tool could define the start of the middle game when an outer turret is destroyed and when each player does not reside as much time in their lane as before. The transition from middle to late game could be defined when the average experience level or amount of items bought by each team reaches a certain threshold, defined by the researcher. This data is provided by the API.

The seventh limitation can be researched by defining potential users of the tool. Interviewing them could yield in requirements for guaranteeing user friendliness. The external validation done at the end of chapter 4 used League eSports players as potential users of the tool. The ideas and opinions of these players could also be used as starting point for designing a tool.

The eighth limitation could be researched by creating a new standalone application which only requires the initial input of the user to function. This would combine the Python and Excel files to one file and would improve the design of the tool and the user friendliness, because the user would only have to open the program and input data, and is not required to follow up specific instructions in order for the tool to work.

The ninth limitation, which is that the Riot API only provides data every minute, inhibits the accuracy of the data. League is a fast paced game where actions happen every second. If it is required to analyse how players walk around the map, then the Riot API is not precise enough. For example, if a dragon is taken by a team at 26:30, then the Riot API only registers the positions of every player on minute 26 and minute 27, which makes it impossible to know who were exactly involved in the act of taking dragon. League has the possibility for players to watch replays of the games they played. Using software to analyse such replays could provide even more information about matches than the Riot API, because every second could in theory be recorded and analysed. The type of visual analysis software to use, and what tools are needed to interpret the corresponding data would be a starting point for further research.

References

Afonso, A. P., Carmo, M. B., & Moucho, T. (2019a). Comparison of Visualization Tools for Matches Analysis of a MOBA Game. *Proceedings of the International Conference on Information Visualisation, 2019-July*. https://doi.org/10.1109/IV.2019.00029

Afonso, A. P., Carmo, M. B., Gonçalves, T., & Vieira, P. (2019b). VisuaLeague: Player performance analysis using spatial-temporal data. *Multimedia Tools and Applications*, *78*(23), 33069–33090. https://doi.org/10.1007/s11042-019-07952-z

Ani, R., Harikumar, V., Devan, A. K., & Deepa, O. S. (2019). Victory prediction in league of legends using feature selection and ensemble methods. *2019 International Conference on Intelligent Computing and Control Systems, ICCS 2019*, *Iciccs*, 74–77. https://doi.org/10.1109/ICCS45141.2019.9065758

Breakthrough (2018, September 20). 11 Celebrity eSports Investors you Didn't Know About. Medium. https://medium.com/@breakthrough_lab/celebrity-esports-investors-you-didnt-know-about-eb9a8c395292

Cavadenti, O., Codocedo, V., Boulicaut, J. F., & Kaytoue, M. (2016). What did I do wrong in my MOBA game? Mining patterns discriminating deviant behaviours. *Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016*. https://doi.org/10.1109/DSAA.2016.75

Charleer, S., Gerling, K., Gutiérrez, F., Cauwenbergh, H., Luycx, B., & Verbert, K. (2018). Real-time dashboards to support esports spectating. *CHI PLAY 2018 - Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*. https://doi.org/10.1145/3242671.3242680

Church, B. C. (2020, August 19). League of Legends is growing. Traditional sports better watch out. CNN. https://edition.cnn.com/2020/08/19/sport/league-of-legends-esports-growth-spt-intl-cmd/index.html

Costa, L. M., Souza, A. C. C., & Souza, F. C. M. (2019). An Approach for Team Composition in League of Legends using Genetic Algorithm. *Brazilian Symposium on Games and Digital Entertainment, SBGAMES*, *2019-Octob*. https://doi.org/10.1109/SBGames.2019.00018

Deja, D., & Myslak, M. (2015). Topological clues for predicting outcomes of multiplayer online battle arena games. *Proceedings of the International Conferences on Interfaces and Human Computer Interaction 2015, IHCI 2015, Game and Entertainment Technologies 2015, GET 2015 and Computer Graphics, Visualization, Computer Vision and Image Processing 2015, CGVCVIP 2015 - P*, *July 2015*, 116–122.

Do Nascimento, F. F., Da Costa, I. B., Da Costa Melo, A. S., & Marinho, L. B. (2017). Profiling successful team behaviors in league of Legends. *WebMedia 2017 - Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, *iii*, 261–268. https://doi.org/10.1145/3126858.3126886

Eaton, J. A., Sangster, M. D. D., Renaud, M., Mendonca, D. J., & Gray, W. D. (2017). Carrying the team: The importance of one player's survival for team success in league of legends. *Proceedings of the Human Factors and Ergonomics Society*, *2017-Octob*. https://doi.org/10.1177/1541931213601550

Gerber, H. R., Sweeney, K., & Pasquini, E. (2019). Using API Data to Understand Learning in League of Legends: A Mixed Methods Study. *Educational Media International*, *56*(2). https://doi.org/10.1080/09523987.2019.1614250

Gray, A. (2018, July 3). The explosive growth of eSports. World Economic Forum. https://www.weforum.org/agenda/2018/07/the-explosive-growth-of-esports/

Kho, L. C., Kasihmuddin, M. S. M., Mansor, M. A., & Sathasivam, S. (2020). Logic mining in league of legends. *Pertanika Journal of Science and Technology*, *28*(1), 211–225.

Lee, C. S., & Ramler, I. (2017). Identifying and evaluating successful non-meta strategies in league of legends. *ACM International Conference Proceeding Series*, *Part F1301*. https://doi.org/10.1145/3102071.3102081

Limelight Networks. (2020, 13 October). State of Online Video 2020. Limelight Networks Inc. https://www.limelight.com/resources/market-research/state-of-online-video-2020/

Market Business News. (2020, December 2). What are eSports? Definition and examples. https://marketbusinessnews.com/financial-glossary/esports-definitio/

Newzoo. (2021, 11 March). 's Global Esports & Live Streaming Market Report 2021 | Free Version. https://newzoo.com/insights/trend-reports/newzoos-global-esports-live-streaming-market-report-2021-free-version

Nordmark, S. (2021, May 12). The 10 Largest Prize Pools in Esports. Dot Esports. https://dotesports.com/general/news/biggest-prize-pools-esports-14605

Olya, G. (2021, January 15). How Esports Is Primed To Take Over Traditional Sports — For Good. Yahoo! Finance. https://finance.yahoo.com/news/esports-primed-over-traditional-sports-010037980.html?guce_referrer=aHR0cHM6Ly9kdWNrZHVja2dvLmNvbS8&guce_referrer_sig=AQAAALSeH6-wiz93HidLS9F84iWF6wsEX_Z3XrmtumWRqCSUyrX9loJBxFlTFJCJyNZ22Z1uv8Ehk-omHEoD5Phsi1jad_QY5Y7r8AXAhecaFEGSoIxwqgmNrZr9-Wzx3UVovZF_j8FAOokdMOFL0toBzbk5evyIKgAu3cro3O70d09a&guccounter=2

Pan, J. (2018, 1 June). The Future of Esports - The Nexus. Medium. https://medium.com/the-nexus/esports-the-convergence-of-two-worlds-a2873bc14b70

Pei, A. (2019, April 14). This esports giant draws in more viewers than the Super Bowl, and it's expected to get even bigger. CNBC. https://www.cnbc.com/2019/04/14/league-of-legends-gets-more-viewers-than-super-bowlwhats-coming-next.html

Reyes, M. S. (2021, 5 January) Esports Ecosystem Report 2021: The key industry companies and trends growing the esports market which is on track to surpass $1.5B by 2023. Business Insider. https://www.businessinsider.com/esports-ecosystem-market-report?international=true&r=US&IR=T

Riot Games, Inc. (2009). League of Legends (Version: patch 10.7) [Video game]. Los Angeles, CA: Riot Games, Inc.

Sangster, M. D. D., Mendonca, D. J., & Gray, W. D. (2016). Big data meets team expertise in a dynamic task environment. *Proceedings of the Human Factors and Ergonomics Society*. https://doi.org/10.1177/1541931213601036

Sapienza, A., Bessi, A., & Ferrara, E. (2018). Non-negative tensor factorization for human behavioral pattern mining in online games. *Information (Switzerland)*, *9*(3). https://doi.org/10.3390/info9030066

Sapienza, A., Peng, H., & Ferrara, E. (2017). Performance dynamics and success in online games. *IEEE International Conference on Data Mining Workshops, ICDMW*, *2017-Novem*. https://doi.org/10.1109/ICDMW.2017.124

Semenov, A., Romov, P., Neklyudov, K., Yashkov, D., & Kireev, D. (2016). Applications of machine learning in dota 2: Literature review and practical knowledge sharing. *CEUR Workshop Proceedings*, *1842*.

Settimi, C. (2020, December 9). The Most Valuable Esports Companies 2020. Forbes. https://www.forbes.com/sites/christinasettimi/2020/12/05/the-most-valuable-esports-companies-2020/?sh=3d80847373d0

Shelp, M. (2020, August 27). The Rising Esports Viewership: Most Watched Esports of 2020. Esports News Network | ESTNN. https://estnn.com/the-rising-esports-viewership-most-watched-esports-of-2020/

Statista. (2021, May 12). League of Legends average viewer count on Twitch 2021. https://www.statista.com/statistics/1108953/league-of-legends-number-viewers/

Summoner's Rift :: League of Legends Wiki :: (2021). MOBAFire. https://www.mobafire.com/league-of-legends/wiki/maps/summoners-rift

Wagner, P. (2018, 5 June). How eSport Prize Purses compare to Traditional Sports. Statista Infographics. https://www.statista.com/chart/14121/esport-prize-pools-in-comparison-to-traditional-sporting-events/

Xia, B., Wang, H., & Zhou, R. (2019). What Contributes to Success in MOBA Games? An Empirical Study of Defense of the Ancients 2. *Games and Culture*, *14*(5). https://doi.org/10.1177/1555412017710599

[Matt Donovan - Summoner Stats - League of Legends]. (2021). OP.GG Europe West. Retrieved from https://euw.op.gg/summoner/userName=matt+donovan

[Matt Donovan (region: EUW) - Champion Stats - League of Legends (Season 11)]. (2021). Mobalytics. Retrieved from https://app.mobalytics.gg/lol/profile/euw/matt%20donovan/overview

Appendix A: The game League of Legends

The video game "League of Legends" (will be abbreviated with just "League") is a MOBA video game, which means Multiplayer Online Battle Arena. League is developed by Riot Games. It is a subgenre of the bigger genre: strategy games. In these type of games, a player controls a chosen unique character for one full match, and tries to achieve a certain main strategic objective to win the game. In MOBA games, the player usually looks from above down at the map which the players play on, which makes it easy to look around at other places at the map where other action takes place. Other examples of MOBA games, are "DotA 2" and "Heroes of the Storm".

Game set-up

League has multiple game modes, but the most popular one which is also used for eSports will be explained for this case. This game mode is a 5 against 5 match played on a map with 3 main lanes, and is played on the map called Summoner's Rift. A match starts with 5 players on two teams each. The two teams battle against each other until the end of the match, which is when the Nexus of either team is destroyed. The Nexus is based in the middle of each team's base, and is guarded by turrets. Most turrets need to be destroyed before the enemy Nexus can be harmed. Each team has to make sure the opposing team does not destroy their Nexus. In Figure 19, Summoner's Rift is shown with the Nexus of each team in each far corner. The text within the picture is elaborated upon later in the Objectives section.

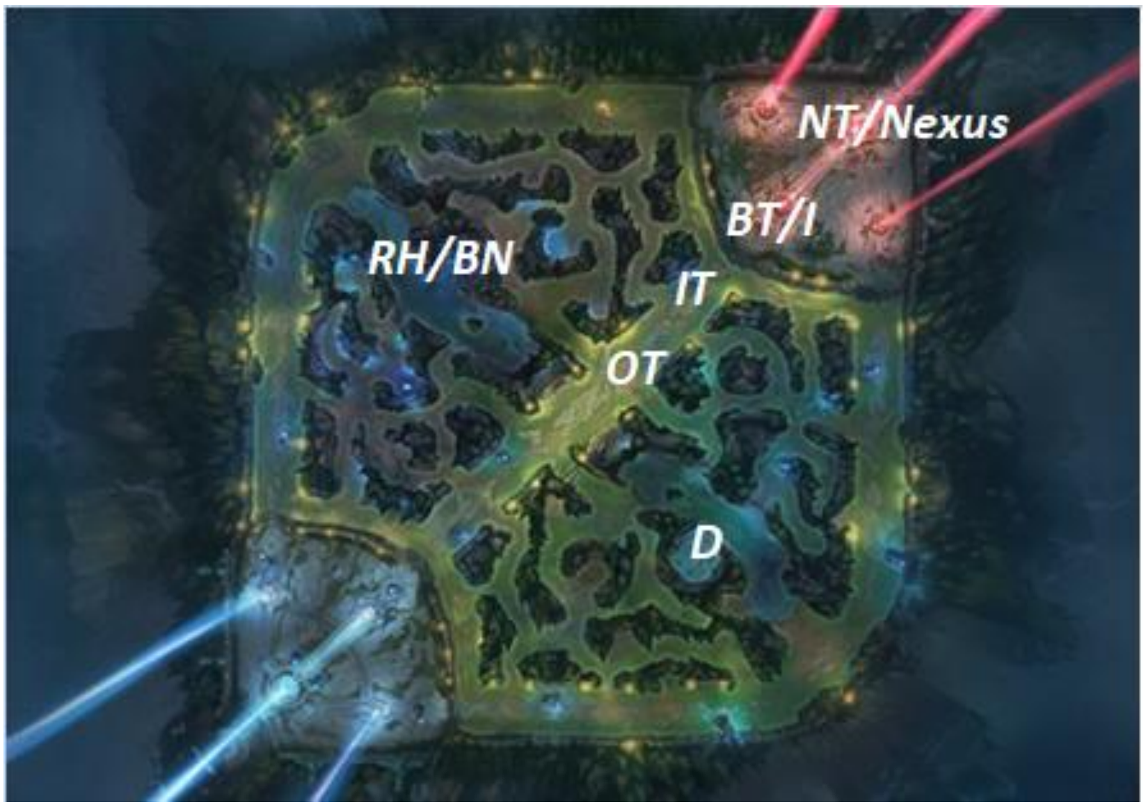


Figure 19: the Summoner's Rift map with its objectives (turrets only shown in mid lane for clarity). Original picture extracted from Summoner's Rift :: League of Legends Wiki :: (2021). MOBAFire. https://www.mobafire.com/league-of-legends/wiki/maps/summoners-rift

When playing League, each player has an account, which is called a summoner. Summoner is also often used synonymously for players. Each player chooses a champion at the beginning of one match and sticks to it the whole match. Each champion has 5 different abilities. There are 3 lanes (top, middle and bottom) and a jungle in between the lanes, which is like a forest. Each player can collect two types of progression in a match: gold and experience. In each lane, minions will spawn constantly for each team. These minions are weak monsters, and killing enemy minions grants the player gold and experience. These minions can be used to take fire from enemy turrets until they die, so that champions can attack the turret safely in the meantime. The jungle contains neutral monsters, which can also be killed for gold. Players can buy up to 6 items from the gold. The shop is at the centre of each team's base. Going back to the allied base also regenerates all health and resources. Each player passively gain gold and experience during the progress of a match, and killing enemies or getting objectives grants even more. Objectives are, for example, destroying enemy turrets or other buildings, or slaying epic monsters. Epic monsters are Baron Nashor, an elemental dragon or drake, or the Rift Herald. On these objectives will be elaborated later in this chapter. Experience accumulates until the concerning champion has enough to level up. A champion starts at the beginning of the match at level 1 and can level up to level 18. For each level-up, the player can choose one of the 4 different abilities to level up and thus make it stronger. However, the fifth passive ability cannot be levelled up this way. When a champion is killed, it will respawn after some time in the allied base. A player does not lose any gold, items or experience when killed. The only penalty is that the player has to wait some time before he respawns in their base, which depends on the current time period the match is in. The more the game has progressed, the longer a dead champion has to wait until it respawns.

Winning the game is often achieved by gaining an advantage over the enemy team in terms of experience and gold. This would mean that your champions are stronger compared to the enemy champions. A clear way to achieve such advantages is by achieving and gaining objectives.

Objectives

Taking objectives grants the concerning team advantages based on the objective taken. Usually, taking an objective refers to taking an elemental dragon, Baron Nashor, Rift Herald or turret. Taking an objective rewards in gold, experience, and also usually grants a long term advantage. For example, if an enemy turret is destroyed, it is permanently destroyed and thus the friendly team gain more possession over the map.

A turret or tower is a structure which has a lot of health and deals a lot of damage to enemies. So, it can be used to gain cover for a player if he retreats back under a turret of his team. Each lane has three turrets. Only the turrets of mid lane are shown in Figure 19 for readability purposes. The other lanes actually also have the same set-up. 'OT' means outer turret, and the other abbreviations of the turret follow the same principle. The outer turret is closest to the middle of the map and thus the first line of defence. Behind the outer turret is the inner turret. Finally, the base turret defends the inhibitor and is the closest of the three to the Nexus. The inhibitor is a structure with no defence and when it is destroyed, it grants the opposing team super minions, which are much stronger than normal minions. In

Figure 19, inhibitor is abbreviated as ‘I’. The inhibitor can respawn or revive however, which happens automatically after 5 minutes. Finally, the base also contains two extra Nexus turrets, which guard the Nexus. For each turret, the following rule counts: if there is a non-destroyed turret in front of the concerning turret in the same lane, then the concerning turret is invincible. Thus, a team has to destroy turrets one by one, from outer turret to nexus turret. The nexus turrets lose their invincibility when a base turret and its inhibitor are destroyed in at least one lane. Similarly, the Nexus can only be harmed if both Nexus turrets are destroyed and if at least one inhibitor in the concerning base is currently destroyed.

The Rift Herald is a neutral monster in the jungle which can spawn up to two times per match. It is located in the top left of the map in Figure 19 and is abbreviated as ‘RH’. The first time will be at 8:00 and the last Rift Herald will simply disappear just before 20:00. The reason for this is that Baron Nashor is about to spawn in the same spot. The Rift Herald can be killed by a team and if so, it grants the team the ability to spawn a Rift Herald in a lane which they see fit. If spawned this way, the Rift Herald will walk with his friendly team down a lane and the Rift Herald will deal tons of damage to enemy turrets. It will continue until the opposing team killed the Rift Herald. The Baron Nashor is an even tougher monster to slay and it often requires the full team to kill it. Slaying Baron Nashor gives a buff to every teammate which makes themselves and the minions of their team stronger. As a team, trying to slay Baron Nashor is one of the hardest objectives to achieve, because it takes a relatively long time and the enemy team can easily interfere and kill your already damaged team. However, the Baron Nashor buff is extremely helpful to win games or gain advantages. The Baron Nashor buff makes it much easier to take down enemy turrets.

Finally, dragons (also called drakes) are other important objectives. These monsters spawn at 5:00 and it respawns 5 minutes after it has been slain. There are 4 basic types of drakes, namely Cloud, Ocean, Infernal and Mountain. Slaying a drake grants all team members permanent buffs for the rest of the match. Thus, a team can rack up drake buffs by slaying the drakes again and again. However, after the second drake has been slain, the game chooses a random dragon type and from that point on, every new drake that spawns is from that type. If a team has slain 4 drakes in total, they gain the Dragon Soul, which is an extra and different powerful buff which is granted to all team members once the last drake has been slain. Every Dragon Soul has different perks. One Dragon Soul may be good at taking down turrets, while the other Dragon Soul may be good at winning teamfights, which is a fight where both full teams are involved. After either team has gained the Dragon Soul, every next drake that spawns is the Elder Dragon. This dragon is comparable to Baron Nashor. The main difference between the two is that Baron Nashor buff is good at taking down turrets, while the Elder Dragon buff is good at winning team fights.

Every objective is harder to achieve in the beginning of the game, because then the champions are still relatively weak. Over the course of the game, taking objectives becomes easier. The course of the game is divided into 3 main time periods: early game, mid game, and late game. Players play differently in every time period.

Time periods

The early game is roughly defined as the time in the match from 0 to 15 in-game minutes. At 15 minutes is the first time a team can surrender, which makes them instantly lose the game if they think defeat is inevitable, and saves the players of the losing team time. However, this surrender vote has to be anonymous by all 5 players to pass. At 20 minutes, it is possible to surrender when only 4 of the players agree. In the early minutes of the early game is also the time when a remake can take place. When analysing match data, it is important to filter out remakes because these matches are not representative of normal matches played. The remake subject is elaborated on later in this section. During the early game, all laners mostly stick to their lane and just farm and sometimes try to get a kill. The junglers mostly farm their jungle and try to gank and get a kill if possible. Drakes and Rift Heralds can also be taken during this period. The first Drake spawns at 5:00 and the first Rift Herald at 8:00. During the early game, the average in-game level of the champions is roughly around 1 to 9.

The mid game is when objectives start to play a bigger role. The time for the mid is roughly signified as between 15 to 25 minutes, but these are not exact thresholds. Players will try to gank more often and try harder to take turrets or epic monsters. It is more common that people will often leave their original lane to help other lanes if needed. The average level of the champions during this period is roughly 10 to 14.

The late game is roughly defined by the time from 25 minutes until the end of the game. Again, this is just an estimate and when the late game starts generally depends on how the game goes. A slow paced game has a late game that starts later. This period is characterised by players having quite some gold and having fully finished most of their items. Some champions are designed to be item reliant, and thus get significantly stronger with their completed items. In this period they become a lot more dangerous compared to other champions who are, generally speaking, less good in the late game. Still, in general, champions are pretty strong at this period due to the lots of experience and gold that each champion has now. Because of this, turrets and monsters die quicker and deal less damage to champions. Thus, taking objectives is much easier in the late game. The respawn timer for champions is a lot longer than in the rest of the other periods, so killing an enemy champion has a much bigger impact. So, in this period, players tend to stick together much more so they can fight back better if they get attacked by multiple enemy players. Getting kills in this period often guarantees getting an objective and thus getting your team ahead. The average level of the champions at this period is around 15 to 18. The level cannot get higher than 18.

Remake

Like mentioned before, a 'remake' matches are important when it comes to analysing matches, because remakes are not representative for average matches played. A remake is possible when the concerning team has one or more players who is or are AFK. A player counts as AFK if he is inactive for 90 seconds. Also, no remake vote can be started if the player who later went AFK was still connected to the game when the opposing team made the first kill of the match against the concerning player's team. From 3 minutes after the beginning (match) of the match, which means 3 minutes of in-game time, the remaining players can vote for a remake to immediately end the game and this match result will not count as a loss for the remaining players who were not AFK. Only two players have to vote yes to make the remake happen, and if two players are AFK instead of one, then only one of the remaining players who are not AFK can vote 'yes' to make the remake happen. However, if the remaining players think the AFK player will come back, they could not vote for a remake, but this basically only happens if the AFK player is premade with most of the remaining players. Once a remake has ensued, the AFK player will be penalised. This can either be a time penalty, which means that the concerning player is not able to start new games for some time. Or, if the type of match played is ranked, the player will lose LP and the match is only for him counted as a loss. LP are points needed to gain a higher rank of the account. The higher someone's rank, the better the player's skill looks. Furthermore, the player will also be reported by LeaverBuster, which is a tool made by Riot, which keeps track of the amount of matches a player has left. Leaving multiple matches in a certain period of time will result in penalties.

When doing research on video games, it is evident that actual matches should be analysed, as this is where the actual gaming takes place. Thus, basic game mechanics, objectives, time periods and remakes were explained.

Appendix B: Functioning of the tool

The tool consists of two parts: a Python part and an Excel/VBA part. In Python, the user inputs certain data such as what summoner they want to analyse. User is referring to the one who is using the tool, and summoner is referred to when talking about the account who is analysed by the tool.

The Python part extracts data from the Riot API portal and scans the chosen summoner's match history. Every summoner playing League across the world can be analysed. Each match is analysed on all the data concerning objectives taken by each team. Python stores all this data in a timeline format, which means that each objective taken is registered at each corresponding in-game minute and time period for every analysed match. Python puts all this categorised information in an Excel file. Another Excel file is used to select and extract certain specified data from the other aforementioned Excel file and puts it in a summary table with the help of VBA coding. This Excel/VBA file makes a Pivot table and graph of the summary table and calculates the average win rate for every situation found in the Excel data storage file. The situations which happened up to 2 times in the big Excel data file are filtered out by default to increase accuracy of the results. The user can adjust this threshold.

Python

The Python part utilises the input data which the user of the tool inputs, including the given API key. With this data and API key, the tool accesses the API portal to find the summoner data first. The API portal can be accessed by constructing a URL with the help of the input data and other looked up values. The URL is a web link which can be reached by typing it in the address bar of an internet browser. The accessed URLs of the code contain an online JSON file, which is a file with data or text values. In this file it is possible for the tool to seek, find and take data of chosen places within the JSON file. In the Python file are four different URL and corresponding JSON files used. One accesses summoner account data, one collects a summoner match history data, one analyses match details of a certain match and one analyses the timeline of each match, which is what happens every minute of the match. Every function needs to use the API key, which the user provides. However, the API key may only be used 100 times per 2 minutes, which means that the user has to wait 2 minutes if the 100 requests have been reached. This waiting process is already integrated in the tool and the program will continue after waiting 2 minutes. This call rate limit is the reason why the API key should not be used 2 minutes before the program is run and it should also not be used while the program is running. The API key expires 24 hours after the last time it has been regenerated in the API portal. Hence, it has to be regenerated before the user uses the API key. It is possible that the API portal will give an error which is an error of the API portal and the website itself, and is not caused by the tool. The tool makes sure that the script continues when such an error occurs, and the appropriate match is skipped which was about to be analysed by the URL and prompted the error.

Code-text simplification

The Python code is explained with the help of a code-text simplification. Texts starting with a hashtag are comments. Texts between square brackets are places where coding resides which aims to execute the goal which is explained between the square brackets. The coding which reside in for loops and if-then statements have been tabbed. For example, if a piece of code resides in two for loops, it is tabbed twice. Showing data, such as the amount of matches filtered or analysed or the dragon counters, have been omitted since these values will not be put in Excel, and thus will not be used in the final analysis. Furthermore, the required Python libraries to execute the code are also omitted in the code-text simplification. In the real Python code, they are located at the top of the script, above the functions. Finally, the API-call-rate counter is also omitted. This counter keeps track of how often the given API key is used by the program. For each fully analysed match, it is required to use the API key twice. Once this counter reached at least 98, it will prompt the program to pause for 2 minutes, which enables the API key rate counter to reset. The code-text simplification is as follows:

```
#Functions defined; each function creates a URL which they extract a JSON data file from:

requestSummonerData (to access accountID)

requestMatchHistory (to access match history of summoner and find matchID’s by using accountID)

requestMatchDetails (find general match data of matches by using matchID’s)

requestMatchTimeline (find timeline data for each minute in a match)


#Main code starts here:

[Define and set base values of variables, which are almost all 0]

[Coding for input data: region, chosen summoner name, API key, matchesdesired (amount of matches scanned)]

[Excel set-up: define Excel file which will be used for data storage]

[Clear old Excel data]


#Goal:  to collect matchID’s

[call function requestSummonerData with input data to collect accountID of summoner]

[call function requestMatchHistory with as input accountID and matchesdesired to collect the summoner’s
match history and all the matchID’s of only ranked matches]


#Goal: analyse every of the filtered matches and register the time when every objective is taken in a match

1: for loop: every matchID in filteredmatches:

        [call function requestMatchDetails to acquire match duration of the matchID and name of every
        participant of a match]

        [filter out unbalanced and thus relatively short matches]
```

```
[check which team is the team of the chosen summoner]
[determine whether the summoner won or lost]
[call function requestMatchTimeline]
[determine match length in minutes]
2. for loop: every minute in matchID:
    [count all events happened in concerning minute]
    [determine the time-period which the minute falls in]
    3. for loop: every event in minute
        if-then: [event happening at the minute] == building destroyed
            [determine in which lane the tower resided]
            [determine tower type]
            [determine whether the friendly or enemy team took it]
            [put data in Excel]
        if-then: [event happening at the minute] == epic monster slain
            [determine monster type]
            [determine whether the friendly or enemy team took it]
            if-then: monstertype == dragon
                [+1 for either enemy or friendly dragon counter]
                [determine dragon 'subtype' and this replaces 'monster type']
            [put data in Excel]
            [put current dragon soul data also in Excel in same row; getting a 4th dragon
            will grant the team the permanent dragon soul positive effect]

#This is where all previous for loops and if-then statements have ended
[print headers in Excel]
[print the value of the total amount of rows in Excel data file]          #this is needed for Excel/VBA
[save Excel file]
#End of code
```

Libraries

Multiple Python libraries have to be imported in order for the script to function properly. A Python library is a type of installation package which contains functions to use in Python. A function is defined as a coding shortcut that can be called upon by typing the function name and its required input data. The libraries that were imported are 'requests', 'openpyxl' and ‘time’. ‘Requests’ was needed to extract the data from the JSON files that can be accessed by compiling URLs. The ‘openpyxl’ library is required for putting data from Python in Excel and turning all the data into a table. The ‘time’ library is needed for pausing the code once the maximum amount of API requests of the used API key have been reached in 2 minutes.

Input data

The Python tool starts off by defining functions which use the ‘requests’ library. After the functions, parameters are initialised and values which should be inputted by the user are asked. The input data which the user has to write down is the region of the account which ought to be analysed, the current summoner name of the account, the API key and the amount of matches which the user chooses to have scanned. These data will function as input for functions.

URL/JSON and summoner data

The tool uses multiple functions. Each function uses the API key to access a URL with a corresponding JSON file to extract information from. An example of such a function is requestSummonerData(region, summonerName, APIKey). It accesses information about the account itself, such as the account ID. The account ID is used to access another function that was made in Python, such as requestMatchHistory. The JSON file which is needed to access the account ID also contains other information besides this account ID. Other listed information at this page are, for example, the summoner name which is shown in the original way of writing, which is with capital letters and spaces included, and also the date when the summoner name was last changed and the summoner level too. The tool extracts only the summoner name and account ID from this JSON file.

Match history scanning

After summoner data is collected, the tool uses this data to access the match history of the summoner. This is based on the user input data, because the user had the option to choose the amount of matches to have scanned. For example, if the user chose 200 matches, then the tool will scan all previous matches starting from the last played match, to the last 200th match played. However, it is possible that not all matches will be analysed. The tool filters out matches which are not ranked matches. Only solo queue ranked matches and ranked flex matches will be used. The outcome of a ranked match directly influences the rank of a player, and thus is deemed as the most serious game modes. The tool filters out all the non-ranked matches of the previous 200 games played. The IDs of the matches which do fit the criteria are placed in a set called ‘matchesfiltered’. Every match has a unique match ID, which is used for analysis of the matches themselves in another function called requestMatchTimeline. Furthermore, the tool filters out matches which are deemed unbalanced. Examples of such matches are remakes or early surrenders. All games within the set ‘matchesfiltered’ which have a match duration of shorter than 22 minutes will be filtered out. At 20 minutes is the first time when a team can surrender without voting unanimously. Since it is possible that a team does not immediately know that 20 minutes have passed, 2 extra minutes are added as slack. In the tool itself, it is possible to change the preferred queue types to analyse by manually changing the ‘desiredqueuetypes’ set, as explained with comments in the code. Initially, the set contains the numbers 420 and 440, which indicate solo queue ranked matches and ranked flex matches respectively. By adding or removing queue type ID’s, the tool will also scan other types of matches. The ID’s for queue types, such as normal queues, are also explained shortly in the code itself.

Scanning the matches itself

For each analysed match, the outcome of the match needs to be registered. The tool checks whether the chosen summoner to be analysed won or lost the match. The players in a match are referred to by their participant ID, which differs per match. The Riot API registers which participant ID is on which team, and registers which team won the match. The Python code determines which participant ID fits the analysed summoner and then checks whether the player is on blue side or red side, and which side won or lost to determine what the match outcome for the chosen summoner is. After this has been determined, the amount of minutes of a match are also registered, which is required to check every minute of the match on whether any team has taken an objective.

The tool then calls the function requestMatchTimeline. The JSON file which it returns contains information on what happened every minute of the match. The tool analyses each minute. Since we categorised the period of each minute in three time-periods: the early, mid and late game, we need to determine in which time-period the appropriate minute and its objective events occur. The tool determines with the help of an if-statement the time-period of such an event. Every minute consists of multiple events. The tool checks every event that happened in a minute and it saves the events which are about an objective being taken by a team. It counts the amount of events in each minute and then uses a triple nested for loop for every match to check every event in every minute.

Every event is checked on whether it is an objective that is being taken, with the help of if- and else-if statements. One if-statement checks whether the concerning event is a building that is being taken and one if-statement checks whether an epic monster is being taken.

The building if-statement looks at towers. However, inhibitor buildings destroyed are also registered. We will refer to these building as turrets also. The tower if-statement checks the lane in which the concerning destroyed turret stood, which is either top lane, middle lane or bottom lane. The tool also checks the position of the turret in the lane itself. This can be an outer turret, inner turret, base turret or nexus turret. The data of the turret events are written down in the Excel data storage sheet.

Another if-statement checks whether an event is about an epic monster that has been slain. If this would be the case, the code determines whether the enemy or friendly team of the analysed player took it by checking the participant ID of the killer in the JSON file. The type of monster slain is also registered. If this monster is a drake, the program checks what type of drake this is and registers it. Furthermore, the tool keeps track of how many drakes each team has taken. When either team takes their fourth drake, they instantly gain the dragon soul buff which corresponds to the last drake taken. The tool will register that dragon soul for the corresponding team and this dragon soul value will remain the same for the rest of the match.

After every event of every minute of every match in the ‘matchesfiltered’ set has been analysed, the tool starts to fill the Excel data storage file, which is the document to register all the in-game objective data. The headers are added and also the amount of rows in this data document are registered. The name of the table with all the data is ExcelDataTable. The amount of rows is used in another Excel/VBA document which will be the dashboard document. After every run, the table is cleared of data and data of the new run is put into it.

After the data has been put into the LoLPMData Excel data storage file, the file is saved by the Python code. It is crucial that this Excel file is closed while the tool is running, because the Excel data file cannot be saved with new information if the file is still open.

The program has multiple ways of dealing with situations like errors or when the API call rate limit has been exceeded. The API call rate is at maximum 100 requests per 2 minutes, which means that the program should pause and wait for 2 minutes if the API call rate gets close to 100 requests. The tool uses a threshold of 98 or more requests for this. The program will wait and after 2 minutes it will continue scanning matches until all matches in the set ‘matchesfiltered’ are scanned.

For every request function, there is an insignificant probability that the URL responds in an error. These types of errors are originating from the Riot servers itself, and not from the tool. When this happens, the corresponding match is simply skipped, and the tool proceeds scanning the other matches as usual.

Excel

There are 2 Excel files, the Excel data storage file and the Excel/VBA dashboard file. The data file registers the data which it receives from the Python code and the dashboard file processes information from the data file and summarises it to show tables and graphs. The name of the Excel data file is “LoLPMData.xlsx”, and the name of the Excel dashboard file is “”LoLPMDashboard.xlsm”. Both files have to be opened when the user is going to analyse the results, where the “LoLPMData.xlsx” file should be opened first. Pop-ups should be enabled when they show up when starting the dashboard file, which is required for VBA functioning.

Excel data storage file

The Excel data storage file is the collector of the data which it receives from the Python script. Each row consists of an event which signifies an in-game objective that has been taken. The headers are as follows:

1. Summoner
2. MatchID
3. Outcome
4. Minute
5. Period
6. Tower
7. Monster
8. Dragon Soul

The file shows which summoner is analysed, which is the chosen summoner of the Python code. This was inputted by the user when starting the program. The MatchID, minute and period all signify data which tell at what moment in which scenario the associated objective was taken. The period refers to the corresponding time-period of the appropriate event, which can refer to the early game for example. The abbreviation for this is 'EARLY'. 'MIDDLE' and 'LATE' are used for the other time periods. The Outcome column shows what team won the match. A 0 means that the enemy team won and a 1 means the friendly team won which the analysed summoner was on during that match. The Minute column tells when the event happened in the match. If an event took place at time 22, then it means that the event took place in the 22th minute of the match, which means that the event took place between 21:00 and 22:00.

The headings Tower and Monster both refer to what building or epic monster was taken respectively. For example, if a building is taken in a certain minute, then the Tower cell will be filled with information about what team took which turret at what location, while the Monster cell will be left empty. This also works vice versa. Once a team is the first to acquire 4 dragons, they will gain the dragon soul buff and the type of dragon soul will be written down in the Dragon Soul column until the end of the match, because this buff is permanent.

The notation for the objectives, which are the Tower, Monster and Dragon Soul columns, will be explained.

The team who took the objective is written down. A hashtag '#' means that the friendly team of the analysed player took it, and the at sign '@' signifies that the enemy team took it. For the Tower column, this means the written down team took a turret from the other team. Furthermore, the lane of the concerning turret is written down after the team sign, and then the type or position of the turret is written down. For example, if a cell in column Tower says '# Top OUTER_TURRET', then it means that the friendly team destroyed a turret from the enemy team on top lane, and the destroyed type of turret is an outer turret. Nexus turrets are abbreviated as 'NexusT'.

Since each epic monster has its own set location on the map, the location does not have to be written down when registering data about the epic monster objective. In the Monster column, only the team sign and the slain monster are written down. First the team sign, and then the corresponding monster. The same principle holds for the Dragon Soul column. The only difference is that the word 'Soul' is added.

In the Excel data storage document, the amount of total rows are counted and registered in a cell next to the data table. This cell value is used in the Excel dashboard file.

Excel dashboard file

The Excel dashboard file utilises Excel functions and VBA coding to summarise and visualise the data in the Excel data file, which is done by creating two tables and a graph. The tables are a normal Excel table and a Pivot table. The Pivot table uses the data of the normal left table as input, and creates a graph of it for visualisation purposes. The normal table, which is called 'SituTable', on the very left of the file is not meant to be looked at by the user, and its purpose is to prepare the data for the second Pivot table on the right. The ‘SituTable’ combines for each row in the Excel data file the 3 columns about the objectives, and then puts them into one single column. This single combination column is the Situation column. The benefit of this is that the permanent dragon soul buff and the other objectives that occur after it is gained, can be written down together as one cell value, and this makes it possible to put the full objective situation information of an event on the x-axis of the Pivot table. If this would not have been implemented, then the tool would see the dragon soul buff and other objectives taken as separate events.

Pivot table can categorise every situation and its corresponding time period in the ‘SituTable’, and calculate the average win rate when such a situation occurred in a match. This average win rate indicates whether achieving a certain situation in-game is favourable. The win rate of every situation is put in the Pivot table and also in the Pivot graph. In the Pivot table, the amount of situations which occurred in the data set are counted. This is called the number of situations. The higher the number of situations, the more accurate the corresponding win rate of the situation is. By default, the tool filters out all cases which happened 2 times or fewer, which is a suitable threshold when 100 matches are analysed to avoid exceptional situations. However, this threshold is still based on the personal preference of the user, and can be adjusted by clicking the ‘Situation’ button in the graph and input different filter values. The user can also filter on situations and periods by using the button.

Buttons

The Excel dashboard file contains three buttons. These buttons should be clicked in the indicated order in the tool itself to properly update the file. Furthermore, it is essential that the Excel data storage file should be opened and loaded before clicking the buttons. The buttons are connected to VBA modules and thus the user should allow macros when the Excel dashboard file asks whether they should be allowed. Another notification might pop up which asks whether data transfer should be allowed or enabled. This should be enabled, because otherwise, the dashboard file cannot analyse the data from the data file.

The first button, called ‘Update Table’ activates a VBA macro which will update the very left table ‘SituTable’ in the document. This makes it possible for the Pivot table to analyse the data in the ‘SituTable’. The macro checks the amount of rows in the Excel data storage file and makes ‘SituTable’ the same size as the original data table situated in the Excel data storage file, which makes it possible to copy and process the values to the dashboard file.

The second button, called ‘Refresh’, refreshes and updates that every table and every graph in the dashboard file.

The third and final button, called ‘Hide Nr. Bars’, makes sure that the Pivot graph only shows the win rate, and not the number of situations, because the win rate is the focus of the graph. The Pivot graph contains both variables because the Pivot table also contains the number of situations, next to the win rate. The button turns the bars of the counted number of situations in the graph invisible.

Conclusion

The analysis of the tool will show a summary of the analysed player's playstyle with relation to in-game objectives. The Pivot table and graph can function as a self-analysis and will show the expected winning probability of decisions with relation to objectives. Furthermore, the results of the tool makes evident which enemy objectives have the largest impact on resulting in a defeat for the player, which would yield the lowest win rate, and thus insight is provided in the analysed player's weaknesses. Certain objectives have a larger impact on the outcome of the match than other objectives. This impact rate can be concluded by how much the win rate of an objective deviates from a 50% win rate. For example, objectives with a win rate of around 50% do not seem to have an outspoken impact, while objectives with a win rate of 1% or 99% do.

Appendix C: Python script

```
   ##PRESS F5 TO RUN CODE


#last time code is customised: 22/7/2021


import requests     #requests has to be installed (via pip installation)

import openpyxl     #openpyxl has to be installed (via pip installation)

from openpyxl import load_workbook

from openpyxl.worksheet.table import Table, TableStyleInfo

import time




#FUNCTIONS

def requestSummonerData(region, summonerName, APIKey):

   URL = "https://" + region + ".api.riotgames.com/lol/summoner/v4/summoners/by-name/"
+ summonerName + "?api_key=" + APIKey

   response = requests.get(URL)

   print("SummonerData =", URL)

   return response.json()


def requestMatchHistory(region, AccID, APIKey, beginIndex, endIndex):

   URL = "https://" + region + ".api.riotgames.com/lol/match/v4/matchlists/by-account/" +
AccID + "?endIndex=" + f"{endIndex}" + "&beginIndex=" + f"{beginIndex}"  + "&api_key=" +
APIKey

   response = requests.get(URL)

   print("MatchHistory =", URL)

   return response.json()


def requestMatchDetails(region, matchId, APIKey):

   URL = "https://" + region + ".api.riotgames.com/lol/match/v4/matches/" + matchId +
"?api_key=" + APIKey
```

```
    response = requests.get(URL)
    print("MatchDetails =", URL)
    return response.json()


def requestMatchTimeline(region, matchId, APIKey):
    URL = "https://" + region + ".api.riotgames.com/lol/match/v4/timelines/by-match/" +
matchId + "?api_key=" + APIKey
    response = requests.get(URL)
    print("MatchTimeline =", URL)
    return response.json()


#MAIN CODE STARTS HERE
APIcallrate = 0
TotalAPIcallrate = 0
beginIndex = 0
matcheschosen = 0
excelrowcounter = 1
matchesscanned = 0
erroramount = 0
unbalancedamount = 0


#INPUT DATA
region = (str)(input('Choose and type one region and press Enter: BR1 EUN1 EUW1 JP1 KR
LA1 LA2 NA1 OC1 TR1 RU: '))
summonerName = (str)(input('Type your League of Legends in-game Summoner name: '))
print(); print("IMPORTANT: Please make sure you did not use this API Key in the last 2
minutes, and do not use it for something else while the program is running."); print()
APIKey = (str)(input('Copy and paste your API Key here: ')); print()
desiredmatches = int(input('Type desired amount of previous matches scanned. For each
100 after 100, the program runs a maximum of 4 minutes longer: '))
#multiples of 100 are recommended if the code somehow fails
```

```
#customisable queue types by user:

desiredqueuetypes = []                    #400 is normal draft; 420 is ranked solo queue; 430
normal blind; 440 is ranked flex (seperate codes by comma's)


print()


#EXCEL SET-UP

filename = 'LoLPMData'                          #the Excel files should be in the same folder as
this Python code

wb = load_workbook(f"{filename}.xlsx")               #both Excel files should be closed before
and while running the Python code

ws = wb.worksheets[0]

oldfilledrows = len(ws['A'])

ws.delete_rows(2, oldfilledrows)


#ID's

responseJSON  = requestSummonerData(region, summonerName, APIKey)#

APIcallrate += 1

print("APIcallrate =", APIcallrate), print()


   #try: back-up plan when API request failed

try:

   AccID = responseJSON['accountId']

except:

   print(responseJSON)

   erroramount += 1

   print("Please restart run. An unexpected error occurred.")


name = responseJSON['name']
```

```
AccID = str(AccID)
print('AccID: "',AccID,'"'), print()

#COLLECT AND FILTER matchId's
matchesfiltered = []
print("desiredqueuetypes =", desiredqueuetypes)
print("(See Python code for more details about queue types)"), print()



while beginIndex < desiredmatches:

    endIndex = beginIndex + 100

    if endIndex > desiredmatches:
        endIndex = desiredmatches

    responseJSONhistory = requestMatchHistory(region, AccID, APIKey, beginIndex,
endIndex)#
    APIcallrate += 1
    print("APIcallrate =", APIcallrate), print()

    try:
        matcheswindow = len(responseJSONhistory['matches'])
    except:
        print(responseJSONhistory)
        matcheschosen -= 1
        matchesfiltered.remove(matchId)
        erroramount += 1
        print("Retry due to unexpected error")
        continue
    for match in range(matcheswindow):
        queuetype = responseJSONhistory['matches'][match]['queue']
        if queuetype in desiredqueuetypes:
            matcheschosen += 1
            matchesfiltered.append(f"{responseJSONhistory['matches'][match]['gameId']}")

    beginIndex += 100



print("amount of matchesfiltered =", len(matchesfiltered))
print("matchesfiltered =", matchesfiltered), print()



#SCANNING matcheschosen
for matchId in matchesfiltered:


    if APIcallrate > 97:                        #this part puts a pause on the code in order to not
exceed the API request limit
        print(); print("Matches analysed until now =", matchesscanned); print()
        print("Program will continue after 2 minutes, please wait. This is supposed to happen.
Don't use the corresponding API Key to access other information in the meantime or it will
screw up the running program.")
        print()
        time.sleep(121)
        TotalAPIcallrate += APIcallrate
        APIcallrate = 0


    #ELIMINATE EARLY SURRENDERS AND REMAKES
    responseJSONdetails = requestMatchDetails(region, matchId, APIKey)#
    APIcallrate += 1
    print("APIcallrate =", APIcallrate); print()


    try:
        gameDuration = responseJSONdetails['gameDuration']
    except:
        print(responseJSONdetails)
        matcheschosen -= 1
        erroramount += 1
        print("Match skipped due to unexpected error")
        continue


    if responseJSONdetails['gameDuration'] < 1320:                  #Duration is in seconds, 1320
is 22 mins; early surrendering may also happen a few minutes after it is possible;
surrendering also takes some time
        matcheschosen -= 1
        unbalancedamount += 1
        print("Unbalanced match removed"), print()
        continue


    matchesscanned += 1
    print("matchId =", matchId)


    #MATCH VALUE RESET
    dragoncounterF = 0
    dragoncounterE = 0
    dragonsoul = ""
    dragonsoulteam = ""
    matchoutcome = ""
    #DETERMINE FRIENDLY TEAM AND OUTCOME
    for x in range(10):
        if responseJSONdetails['participantIdentities'][x]['player']['currentAccountId'] == AccID:
            participantId = responseJSONdetails['participantIdentities'][x]['participantId']
            teamfriendlyId = responseJSONdetails['participants'][x]['teamId']
            print("teamfriendlyId =", teamfriendlyId)

    if (responseJSONdetails['teams'][0]['win'] == "Win") and
(responseJSONdetails['teams'][0]['teamId'] == teamfriendlyId):
        matchoutcome = 1
    elif (responseJSONdetails['teams'][0]['win'] != "Win") and
(responseJSONdetails['teams'][0]['teamId'] != teamfriendlyId):
        matchoutcome = 1
    else:
        matchoutcome = 0
    print("matchoutcome =", matchoutcome), print()

    #REGISTER DATA ABOUT OBJECTIVES
    responseJSONtimeline = requestMatchTimeline(region, matchId, APIKey)#
    APIcallrate += 1
    print("APIcallrate =",APIcallrate); print()

    try:
        matchframes = responseJSONtimeline['frames']
    except:
        print(responseJSONtimeline)
        matcheschosen -= 1
        erroramount += 1
        print("Match skipped due to unexpected error")
        continue
    matchframes = len(responseJSONtimeline['frames'])
    print("totalminutes =", matchframes - 1), print()

    #MINUTES
    for minute in range(matchframes):
        minuteevents = len(responseJSONtimeline['frames'][minute]['events'])
        if minuteevents != 0:

            #GAME PERIOD
            if minute < 15:
                period = "EARLY"
            elif 15 <= minute <= 25:
                period = "MIDDLE"
            elif minute > 25:
                period = "LATE"

            #EVENTS
            for eventnr in range(minuteevents):

                #RESET OBJECTIVES
                tower = ""
                monster = ""

                #TOWER/INHIBITOR
                if responseJSONtimeline['frames'][minute]['events'][eventnr]['type'] ==
"BUILDING_KILL":
                    towerLane =
responseJSONtimeline['frames'][minute]['events'][eventnr]['laneType']
                    if towerLane == "TOP_LANE":
```

```
                towerLane = "Top"
            elif towerLane == "MID_LANE":
                towerLane = "Mid"
            elif towerLane == "BOT_LANE":
                towerLane = "Bot"
            towerType =
responseJSONtimeline['frames'][minute]['events'][eventnr]['towerType']
            teamId = responseJSONtimeline['frames'][minute]['events'][eventnr]['teamId']
            if teamId != teamfriendlyId:
                team = "#"
            else: team = "@"
            if towerType == "NEXUS_TURRET":
                towerLane = ""
                towerType = "NexusT"
            if towerType == "UNDEFINED_TURRET":
                towerType = "INHIB"

            tower = f"{team} {towerLane} {towerType}"
            print(f"In the {minute}th minute: {tower}")

            excelrowcounter += 1
            ws[f"A{excelrowcounter}"] = name
            ws[f"B{excelrowcounter}"] = matchId
            ws[f"C{excelrowcounter}"] = matchoutcome
            ws[f"D{excelrowcounter}"] = minute
            ws[f"E{excelrowcounter}"] = period
            ws[f"F{excelrowcounter}"] = tower
            ws[f"G{excelrowcounter}"] = monster
            ws[f"H{excelrowcounter}"] = dragonsoul
```

```
        #DRAKE/BARON/HERALD
            if responseJSONtimeline['frames'][minute]['events'][eventnr]['type'] ==
"ELITE_MONSTER_KILL":
                if (0 < responseJSONtimeline['frames'][minute]['events'][eventnr]['killerId'] <= 5)
and (teamfriendlyId == 100):
                    team = "#"
                elif (5 < responseJSONtimeline['frames'][minute]['events'][eventnr]['killerId'] <=
10) and (teamfriendlyId == 200):
                    team = "#"
                else: team = "@"
                monsterType =
responseJSONtimeline['frames'][minute]['events'][eventnr]['monsterType']
                if monsterType == "DRAGON":
                    if team == "#":
                        dragoncounterF += 1
                    else: dragoncounterE += 1
                    monsterType =
responseJSONtimeline['frames'][minute]['events'][eventnr]['monsterSubType']
                    if monsterType == "EARTH_DRAGON":
                        monsterType = "Mountain_Drake"
                    elif monsterType == "WATER_DRAGON":
                        monsterType = "Ocean_Drake"
                    elif monsterType == "AIR_DRAGON":
                        monsterType = "Cloud_Drake"
                    elif monsterType == "FIRE_DRAGON":
                        monsterType = "Infernal_Drake"
                monster = f"{team} {monsterType}"
                print(f"In the {minute}th minute: {monster}")


                excelrowcounter += 1
                ws[f"A{excelrowcounter}"] = name
                ws[f"B{excelrowcounter}"] = matchId
                ws[f"C{excelrowcounter}"] = matchoutcome
                ws[f"D{excelrowcounter}"] = minute
                ws[f"E{excelrowcounter}"] = period
                ws[f"F{excelrowcounter}"] = tower
                ws[f"G{excelrowcounter}"] = monster

                #DRAGON SOUL
                if dragonsoulteam == "":
                    if dragoncounterF >= 4:
                        dragonsoulteam = "#"
                        dragonsoul = f"{dragonsoulteam} {monsterType}Soul"
                    elif dragoncounterE >= 4:
                        dragonsoulteam = "@"
                        dragonsoul = f"{dragonsoulteam} {monsterType}Soul"
                ws[f"H{excelrowcounter}"] = dragonsoul

        print()
        print("dragoncounterE =", dragoncounterE)
        print("dragoncounterF =", dragoncounterF)
        print("dragonsoul =", dragonsoul)
        print("dragonsoulteam =", dragonsoulteam), print()

TotalAPIcallrate += APIcallrate
print("Matches analysed =", matchesscanned)
print("Final APIcallrate =", TotalAPIcallrate)
print("Amount of matches skipped due to unexpected unimportant errors during run =",
erroramount)
print("Amount of heavily unbalanced matches removed =", unbalancedamount)
```

```
#EXCEL PRINTING
newfilledrows = len(ws['A'])
ws["A1"] = "Summoner"
ws["B1"] = "MatchID"
ws["C1"] = "Outcome"
ws["D1"] = "Minute"
ws["E1"] = "Period"
ws["F1"] = "Tower"
ws["G1"] = "Monster"
ws["H1"] = "Dragon Soul"

ws["K1"] = "Amount of rows of data"
ws["K2"] = newfilledrows

exceldatatable = openpyxl.worksheet.table.Table(ref = f"A1:H{newfilledrows}", displayName
= "ExcelDataTable")
exceldatatable.ref = f"A1:H{newfilledrows}"
print("Excel raw data table size =", exceldatatable.ref)

if exceldatatable.name not in ws.tables:
   ws.add_table(exceldatatable)
elif exceldatatable.name in ws.tables:
   ws.tables["ExcelDataTable"].ref = exceldatatable.ref

wb.save(f"{filename}.xlsx")

print(), print()
print(); print('Program finished successfully! First open:'); print(f"{filename}.xlsx"); print()
print('After it has successfully loaded, open:'); print('LoLPMDashboard.xlsm')
```

Appendix D: VBA macros

This file contains the coding for the VBA macros connected to the clickable buttons of the Excel dashboard file.

The VBA macro of button 1, which update the SituTable on the left of the Excel dashboard file, which is used as input for the Pivot table and Pivot chart

```
Sub UpdateTableLength()


' Measure rowlength of new data table and put it in situation table


  SituOldlength = Range("A" & Rows.Count).End(xlUp).Row
  OGDatalength = Workbooks("LoLPMData.xlsx").Worksheets("Data").Range("K2")
  If IsEmpty(OGDatalength) Then
  OGDatalength = 2
  End If
  SituNewrange = "$A$1:$C$" & OGDatalength
  ActiveSheet.ListObjects("SituTable").Resize Range(SituNewrange)
  If OGDatalength < SituOldlength Then
    SituDeleteRange = "$A$" & (OGDatalength + 1) & ":$C$" & SituOldlength
    Range(SituDeleteRange).ClearContents
  End If


End Sub
```

The VBA macro of button 2, which refreshes the tables and graphs

```
Sub RefreshPivotTable()

' RefreshPivotTable Macro

    ActiveWorkbook.RefreshAll

End Sub
```

The VBA macro of button 3, which hides the bars in the graph which show the number of occurrences of each objective event and their timing

```
Sub HideCountBars()

' HideCountBars Macro

    On Error GoTo Error

    ActiveSheet.ChartObjects("Average Winrate").Activate
    ActiveChart.FullSeriesCollection(2).Select
    Selection.Format.Fill.Visible = msoFalse
    Range("AF8").Select

Exit Sub
Error:
Range("AE8").Select
Exit Sub
End Sub
```

Appendix E: External validation survey

Introduction

This file contains questions about the Excel dashboard file which shows the expected win rate of each objective taken by either team in either the early, middle, or late game. The middle game is assumed to happen between 15 and 25 minutes. The early game happens before that and the late game afterwards. The data gathering is done by scanning the last 60 matches played by the inputted player (which should be your account), and by checking which objectives were taken in a match and whether the player won. Only draft pick matches are included, both normal and ranked games. Objectives are defined as: Rift Heralds, Baron Nashors, Elemental Drakes, and turrets destroyed. The data gathering process is done by using a Python tool which automatically extracts data from the Riot API.

It might seem obvious that taking Nexus Turrets will often result in a 100% expected win rate, but this is not the focus of the tool. The value of the analysis lies in the fact that whether choosing to take a certain objective over another will more likely result in a victory, and thus this could possibly show which objectives are more important (for the inputted player). For example, the tool could show that taking Infernal drakes results in a higher expected win rate than taking Cloud drakes. Gathering feedback would help to discover possible improvement points according to Esports players, and whether the tool indeed gives insight in the importance of objectives. This is the reason why I made this questionnaire.

The filtering of data is possible by adjusting the bottom left filter buttons on the Pivot chart. Please use these buttons to filter information, instead of adjusting the Pivot table left of the graph. You may ignore the “Update Table”, “Refresh”, and “Hide Nr. Bars” buttons, as these are used for the initial data gathering process, which is already done. If the Excel dashboard file is somehow not working, then please check the figures at the end of this survey to see the data of one of the accounts of the Esports player Rekkles.

The questions to answer are on the next page. I plan to summarise the responses to this questionnaire and use it to write about the external validation part of my thesis.

Thank you for filling in the questionnaire!

Questions

1. *Do you think this tool could help you gain insight in your own gameplay and could help improve the strategy of players? If so, how?*

2. *Did you think you could draw any conclusions from your data? Please explain why or why not. What do you think is required to make such conclusions more accurate?*

3. *Do you think the dashboard data can help to create strategies to counter opponents? If so, how?*

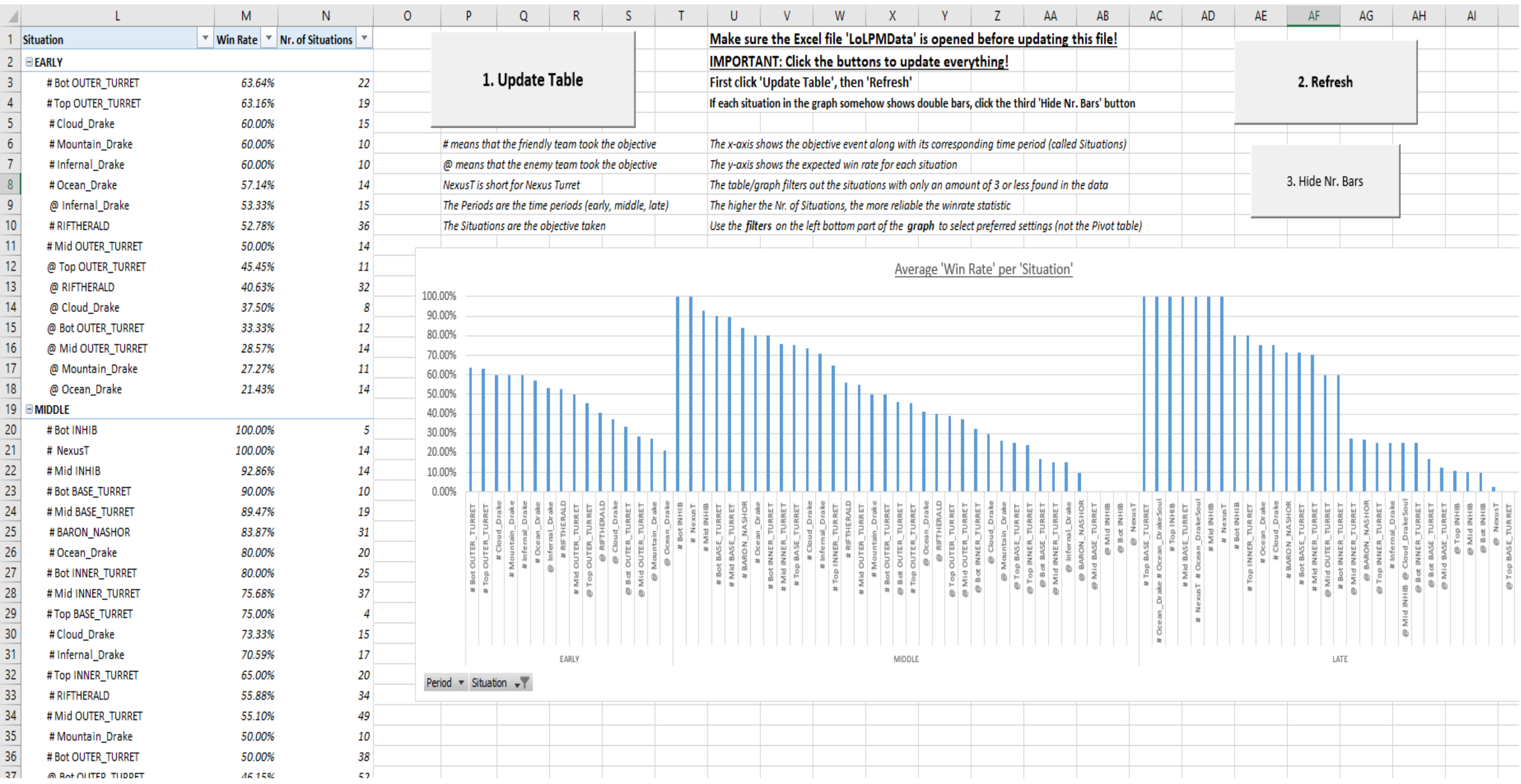
4. *Does the dashboard provide enough user friendliness? (is it easy to use? Are there aspects which seem clunky?)*

5. *Do you think that the idea of analysing the success of taking objectives would be useful for Esports teams?*

6. *Do you think the tool would be more useful if it would focus instead on other events that happen in-game (such as kills, items bought, or amount of wards placed)? If so, why?*

7. *Do you have any additional remarks, suggestions, or improvement points?*

Figures of survey



Situation	Win Rate	Nr. of Situations
EARLY		
# Bot OUTER_TURRET	63.64%	22
# Top OUTER_TURRET	63.16%	19
# Cloud_Drake	60.00%	15
# Mountain_Drake	60.00%	10
# Infernal_Drake	60.00%	10
# Ocean_Drake	57.14%	14
@ Infernal_Drake	53.33%	15
# RIFTHERALD	52.78%	36
# Mid OUTER_TURRET	50.00%	14
@ Top OUTER_TURRET	45.45%	11
@ RIFTHERALD	40.63%	32
@ Cloud_Drake	37.50%	8
@ Bot OUTER_TURRET	33.33%	12
@ Mid OUTER_TURRET	28.57%	14
@ Mountain_Drake	27.27%	11
@ Ocean_Drake	21.43%	14
MIDDLE		
# Bot INHIB	100.00%	5
# NexusT	100.00%	14
# Mid INHIB	92.86%	14
# Bot BASE_TURRET	90.00%	10
# Mid BASE_TURRET	89.47%	19
# BARON_NASHOR	83.87%	31
# Ocean_Drake	80.00%	20
# Bot INNER_TURRET	80.00%	25
# Mid INNER_TURRET	75.68%	37
# Top BASE_TURRET	75.00%	4
# Cloud_Drake	73.33%	15
# Infernal_Drake	70.59%	17
# Top INNER_TURRET	65.00%	20
# RIFTHERALD	55.88%	34
# Mid OUTER_TURRET	55.10%	49
# Mountain_Drake	50.00%	10
# Bot OUTER_TURRET	50.00%	38
@ Bot OUTER_TURRET	46.15%	52

Make sure the Excel file 'LoLPMData' is opened before updating this file!

IMPORTANT: Click the buttons to update everything!

First click 'Update Table', then 'Refresh'

If each situation in the graph somehow shows double bars, click the third 'Hide Nr. Bars' button

# means that the friendly team took the objective

@ means that the enemy team took the objective

NexusT is short for Nexus Turret

The Periods are the time periods (early, middle, late)

The Situations are the objective taken

The x-axis shows the objective event along with its corresponding time period (called Situations)

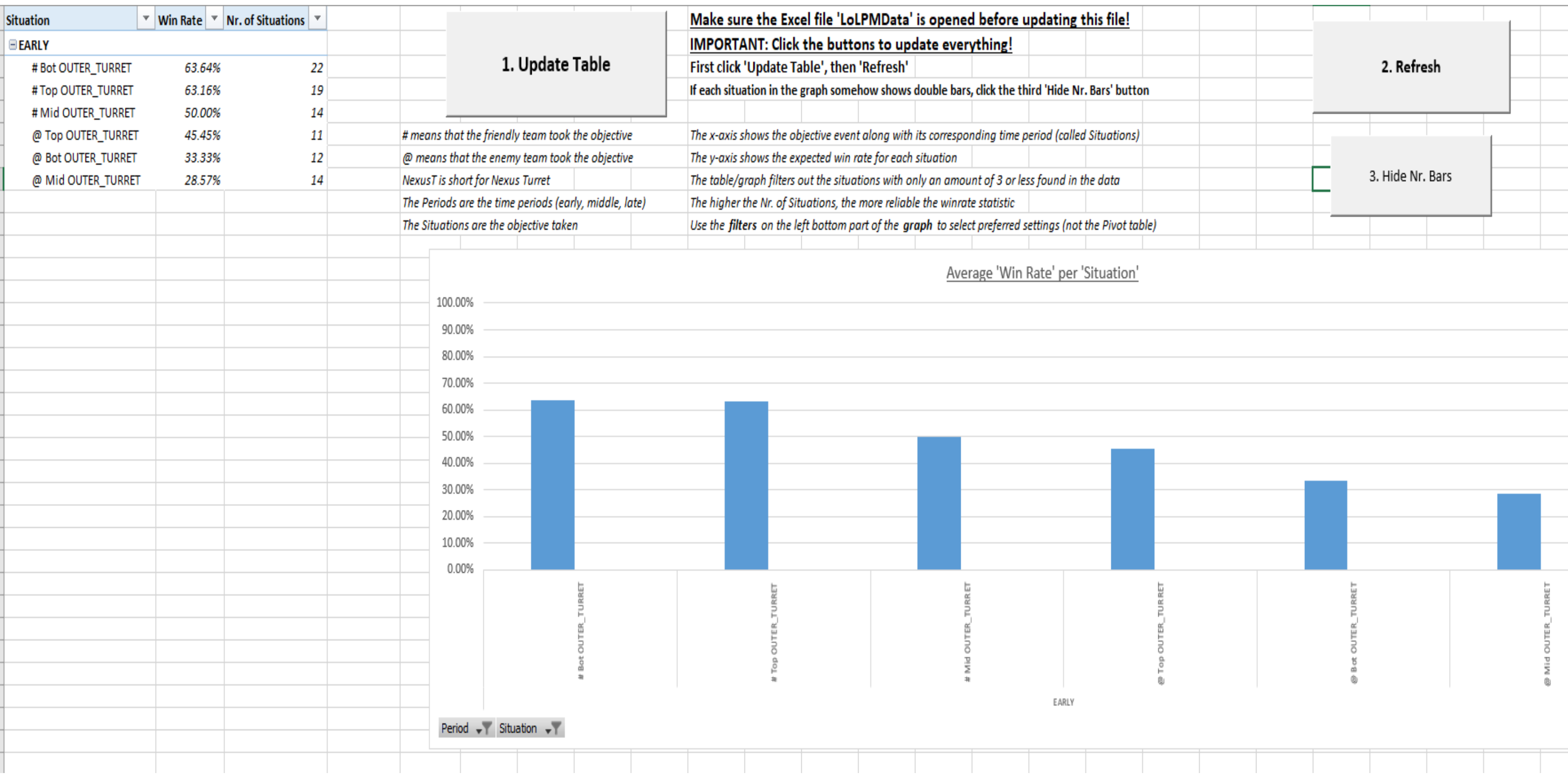
The y-axis shows the expected win rate for each situation

The table/graph filters out the situations with only an amount of 3 or less found in the data

The higher the Nr. of Situations, the more reliable the winrate statistic

*Use the **filters** on the left bottom part of the **graph** to select preferred settings (not the Pivot table)*

Figure 20: The output of the Excel dashboard file



Situation	Win Rate	Nr. of Situations
EARLY		
# Bot OUTER_TURRET	63.64%	22
# Top OUTER_TURRET	63.16%	19
# Mid OUTER_TURRET	50.00%	14
@ Top OUTER_TURRET	45.45%	11
@ Bot OUTER_TURRET	33.33%	12
@ Mid OUTER_TURRET	28.57%	14

Make sure the Excel file 'LoLPMData' is opened before updating this file!

IMPORTANT: Click the buttons to update everything!

First click 'Update Table', then 'Refresh'

If each situation in the graph somehow shows double bars, click the third 'Hide Nr. Bars' button

# means that the friendly team took the objective

@ means that the enemy team took the objective

NexusT is short for Nexus Turret

The Periods are the time periods (early, middle, late)

The Situations are the objective taken

The x-axis shows the objective event along with its corresponding time period (called Situations)

The y-axis shows the expected win rate for each situation

The table/graph filters out the situations with only an amount of 3 or less found in the data

The higher the Nr. of Situations, the more reliable the winrate statistic

*Use the **filters** on the left bottom part of the **graph** to select preferred settings (not the Pivot table)*

Figure 21: Example of filtered settings in the Excel dashboard file with regards to turrets taken in the early game