Markovljev proces odlučivanja i njegova primjena u kontekstu računalnih igara

Cerovec, Sven

UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN

Sven Cerovec

MARKOV DECISION PROCESS AND ITS APPLICATION IN THE CONTEXT OF COMPUTER GAMES

BACHELOR'S THESIS

Varaždin, 2021

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Sven Cerovec

Student ID: 0016136668

Programme: Information Systems

MARKOV DECISION PROCESS AND ITS APPLICATION IN THE CONTEXT OF COMPUTER GAMES

BACHELOR'S THESIS

Mentor:

Bogdan Okreša Đurić, PhD

Varaždin, September 2021

Sven Cerovec

Statement of Authenticity

Hereby I state that this document, my Bachelor's Thesis, is authentic, authored by me, and that, for the purposes of writing it, I have not used any sources other than those stated in this thesis. Ethically adequate and acceptable methods and techniques were used while preparing and writing this thesis.

The author acknowledges the above by accepting the statement in FOI Radovi online system.

Abstract

Creating programs that play games has been around for some time, a while back programmers were competing to create a program that plays chess perfectly, and they are still refining these programs today. Classic games like chess are not the only games being affected, with the increase in popularity of online games, arose a growing interest in gaming artificial intelligence. Using information and programs made by other people on the internet I built my own that plays the most popular game in the last decade; League of Legends. What helped make this program was theory on Markov decision processes which have been used to solve real world problems in the past. My findings confirmed that with the use of a Markov decision process model for the creation of my artificial intelligence, it greatly improved the logic behind the bot I was creating.

Keywords: artificial intelligence, Markov chain, Markov decision process, League of Legends, gaming, stochastic game, bot

Table of Contents

1. Introduction

When IBMs supercomputer Deep Blue defeated Russian grandmaster Gary Kasparov back in 1997 in a gentleman's game of chess everyone was dumbfounded, at the time it was believed it would take decades for a computer to accomplish a feat as impressive as beating the world champion of chess. But if artificial intelligence could conquer the best of us it could conquer all of us, and so the topic of hostile AI had been made relevant once again.

The term artificial intelligence or the more commonly used acronym AI was coined in 1956 when the search for a solution to intelligence began, but many approached the problem cautiously and questioned what would happen if AI would turn against us, if everything went wrong. The idea of a post-apocalyptic future ruled by AI was born long before Deep Blue, in 1984 AI was all the rave with the release of the science fiction action film The Terminator.

Today, AI is a common conversation starter thanks to SpaceX owner and Tesla CEO Elon Musk repeatedly warning us that AI would soon become just as smart as humans, the main culprit being DeepMind, a research lab best known for developing AI systems that 'crush all humans at all games'. Before AI can go about exterminating all human life it first must learn how to play League of Legends, a modern game and a mainstay Esport of the 21st century gaming zeitgeist.

This paper will focus on a theoretical breakdown and explanation of Markov chains, how they are intertwined with AI sciences and my work on a simple League of Legends AI system that applies Markov chains, although I won't be pitting it against the world champions in League of Legends.

2. Methods and work techniques

For this paper I have gathered information mostly online trough services like Google Scholar and Wikipedia. I have also used my own experiences and observations about the topic in hopes of elevating it.

To develop the AI system for League of Legends I used Python, a programming language that includes a module used to control mouse and keyboard inputs called PyAutoGUI, another library I used was OpenCV, which is an advanced version of PyAutoGUI that is used for object detection in images.

The most useful information always came from watching videos on YouTube, specifically the YouTube channel 'Learn Code By Gaming' [1] and their video series 'How To Bot with OpenCV'. A chunk of the code I made regarding object detection is a modified version of theirs. About PyAutoGUI, making bots and Python code in general I learned from the YouTube channel Kian Brose [2] and their series on making bots for different games.

Testing the bot I made was as simple as running a game and the bot at the same time. It went trough different test phases, such as testing if the bot can do a specific thing I wanted it to, or at the end seeing if the bot could win a game against other bots all on it's own, which was the final goal I set for the bot to achieve.

Testing against human opponents wasn't as successful, because the bot is using image processing to detect what is happening on screen, it isn't as fast as a human at reaction speeds. Human opponents were quick to figure out the bot's drawbacks and negatives, they were able to abuse them and get advantages. I never intended the bot to win against a human, so this didn't come as a surprise.

To create and edit this paper in the LaTeX system of document preparation I used an online cloud-based service called Overleaf, in a template designed to help create scientific works for the Faculty of Organisation and Informatics provided by my mentor Bogdan Okreša Đurić.

3. Topic Elaboration

I have divided this paper into 4 chapters. The first is about artificial intelligence, it's definition, history and modern application. The second chapter is an introduction to the Markov decision process. The third is about League of Legends in general, the game I built an AI system for, the AI system that is the topic of the fourth and final chapter.

3.1. Artificial Intelligence

What is artificial intelligence? How did it come to be? What can we do with it? These questions will be the topic of this chapter. I would find it hard to believe if you told me you haven't heard of the term 'artificial intelligence' before, as it is an ever-present buzzword occasionally popping up on social media and in news articles. Nevertheless, let's first look at how to define artificial intelligence, before moving on to other the questions in need of answering.

3.1.1. The umbrella term 'Artificial intelligence'

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. [3] AI can also be thought of as a branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. [4] To the question 'What is artificial intelligence?' John McCarthy of Stanford University [5], a founder of the discipline of artificial intelligence answers with: "It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."

What even is intelligence? Oxford Dictionary [6] defines intelligence as "the ability to acquire and apply knowledge and skills as well as to learn, understand and think in a logical way about things". According to McCarthy varying kinds and degrees of intelligence occur in people, many animals and some machines, but at what point can a machine be deemed intelligent?

Human intelligence is often assessed with an intelligence quotient, a score derived from a test that measures the ratio of the age at which a person normally makes a certain score to the person's actual age. You can certainly make a machine that solves IQ tests, but that machine wouldn't be very useful in other ways meaning IQ test can't really be applied to AI. As of today, we don't really know how to assess whether or not something embodies artificial intelligence and benchmarks that a machine would need to pass to be deemed AI change day by day as technology advances. Machines and programs that were considered intelligent 10 years ago are being disregarded today as inherent computer functions.

One way to distinguish between AI and non-AI is the Turing test named after English scientist Alan Turing considered by many to be the father of AI. To Turing the question of whether

a machine can think is meaningless, he proposes a somewhat related question of whether a machine can do well in game he called the Imitation Game. What follows is an explanation of 'The Imitation Game'.

> "Suppose that we have a person, a machine, and an interrogator. The interrogator is in a room separated from the other person and the machine. The object of the game is for the interrogator to determine which of the other two is the person, and which is the machine. The interrogator knows the other person and the machine by the labels 'X' and 'Y' and at the end of the game says either 'X is the person and Y is the machine' or 'X is the machine and Y is the person'. The interrogator is allowed to put questions to the person and the machine of the following kind: "Will X please tell me whether X plays chess?" Whichever of the machine and the other person is X must answer questions that are addressed to X. The goal of the machine is to try to cause the interrogator to mistakenly conclude that the machine is the other person; the goal of the other person is to try to help the interrogator to correctly identify the machine." [7, ch. 1]

In recent years there have been computer programs capable of deceiving interrogators in five minutes of questioning, none of them are convincing enough for what Turing envisioned we would have, in fact it is safe to say Turing would be fairly disappointed as he predicted in 1950 [7] that in fifty years from the time of his writing, it will be possible to program computers to play the imitation game so well that an average interrogator will have no more than a seventy percent chance of making the right identification. Although we might need 50 more years to have a convincing candidate that Turing would approve of, we are slowly getting there and might soon be facing a breakthrough in AI sciences.

The Turing test is the most popular, but not the only test proposed for declaring if something can be considered artificial intelligence. Research on the measurement of intelligence has been done and is still being done by the day because intelligence as of yet has not been solved.

3.1.2. Talking mechanical handmaidens (A brief history of AI)

The idea of an inanimate object mimicking human behaviour is not only relevant today but it is also an idea discussed by ancient Greek philosophers whose god Hephaestus was said to have built talking mechanical handmaidens out of pure gold. Talking mechanical handmaidens aren't exclusively built by gods, people have also been creating robots for a while now, from Egyptian statues with moving heads to factory machines and industrial robots to AI robots we have today. Automating things is something we've been troubled with for the last few centuries, and what better way to automate things than create a thing that automates things for you, hence our obsession with artificial intelligence.

In the 1940s advances in psychology and our understanding of neural networks alongside the rise of computers paved the way for the 1950s boom in AI technology. In 1950 Alan

Turing published his 'Computer Machinery and Intelligence' which included the aforementioned Turing Test, Claude Shannon published the paper 'Programming a Computer for Playing Chess' and Isaac Asimov published the 'Three laws of robotics'. The phrase artificial intelligence was first coined in 1956 at the "Dartmouth Summer Research Project on Artificial Intelligence". Led by John McCarthy, the conference, which defined the scope and goals of AI, is widely considered to be the birth of artificial intelligence as we know it today. McCarthy would later go on to establish the AI lab at Stanford in 1963. [4]

The expectations during the 1970s were too high for AI to achieve, research was being backed by national governments and many believed we were on the verge of a breakthrough. One of them was Marvin Minsky, co-founder of the MIT Artificial Intelligence Project alongside McCarthy, who in 1970 told Life Magazine, "from three to eight years we will have a machine with the general intelligence of an average human being." It would soon be revealed that there wasn't enough computational power for a machine to even remotely exhibit intelligence. Frustration with the progress of AI development (or lack thereof) dwindled the funds of academic research. The period from 1974 to 1980 would henceforth be known as the First AI Winter. [8]

The term *expert systems* was introduced to the industry in the 1980s . An expert system would learn how to respond to virtually every situation in a field from a human expert. Amateurs and newcomers to a field could learn from these systems, it was in it's essence an educational tool. The Japanese government invested 400 million dollars into these systems as part of their Fifth Generation Computer Project, but most of their goals were not met, AI once again fell out of the limelight. [8]

In the 1990s and 2000s many of the goals had been achieved despite AI research receiving less funding and publicity. In 1997 Gary Kasparov was defeated by Deep Blue, the same year speech recognition software was implemented on the Windows Operating System.

The Chinese game called Go is a two player strategy board game in which the objective is to capture more territory than the opponent, but compared to chess it is considerably more complex and therefore harder to play for people as well as AI. The complexity of the ancient game was seen as a major hurdle to clear for AI, but in 2016 for the first time Alpha Go, a computer program developed by DeepMind won against world champion Lee Sedol. Later at the 2017 Future of Go Summit AlphaGo beat Ke Jie, the number one ranked player in Go at the time. [8] [4]

Programs that play games on their own are a hot topic in the E-sport community, as of yet there aren't any programs or more commonly known as *bots* (short for robot) that could convincingly beat world champions in any single MOBA (Multiplayer online battle arena) but things could change rapidly for the casual player for the worse. More on bots and how they affect the gaming industry in a later chapter.

3.1.3. Virtual assistants, autonomous robotic vacuum cleaners and self-driving cars

A beautiful morning. I get out of bed as my automated curtains slowly start raising, letting in only a fraction of sunlight at a time. As I'm walking down the hallway to the kitchen, the room temperature slightly increases with each step.

'Good morning Alexa' I say, after a short yawn.

'Good morning. How was your sleep?' asks Alexa with morbid curiosity in her voice before patiently awaiting my answer.

'It was okay.' I reply. 'I'll get some more sleep in the Tesla.'

As I'm getting a beverage from my Samsung smart fridge, I feel a rigid object hitting my ankle. It's my Roomba. I stare wide-eyed at it persistently trying to get past my feet when I realized my eyes hadn't been opened wide, they were closed this entire time. I wake up in my room, with no virtual assistant, no autonomous robotic vacuum cleaner and no self-driving car. I sigh in utter disappointment.

Today, AI is everywhere. AI is in our computers, phones, cars, watches, fridges, doors, stove tops, microwaves, supermarkets, cinemas. It is a prevalent topic in sciences such as information technologies, psychology, ethics, philosophy, robotics and so on. We can apply AI to just about anything. Although most of us are yet to own a Roomba, a smart home is not that far from being the standard in the 1st world. AI is not exclusively a privilege of the wealthy, according to David Mhlanga's research [9] because the way AI technology is changing education, agriculture and data collection in general it is also having a positive influence on poverty reduction.

The first "robot citizen", a humanoid robot named [4] was created in 2016 by Hanson Robotics and is capable of facial recognition, verbal communication and facial expression. In 2021 Tesla announced the development of what they [10] describe as "a general purpose, bi-pedal, humanoid robot capable of performing tasks that are unsafe, repetitive or boring". It seems that not only do we want something to think like us, we also want it to look like us.

Language recognition and translation has always been one of the main drives of AI research. Any modern phone has speech recognition and controls for situations where you would rather control your phone with speech instead of holding it in your hands. BERT was released in 2018 by Google, it is a natural language processing engine that achieved outstanding performance on a number of standardised language understanding tasks.[4]

Self-driving cars are a benchmark yet to be perfected and fully utilised. Self-driving cars use a variety of sensors to move safely, avoid obstacles and recognise relevant road signs with very little or no human input. Waymo launched its Waymo One service in 2018, offering people in Phoenix, Arizona rides trough the city in their autonomous vehicles. A handful of companies have been given the green light to use self-driving cars for their operations and even racing events and series have been launched for driverless cars.[4] [11]

Since the conception of COVID-19, AI has been used to fight the pandemic by speeding

up genome sequencing, making faster diagnoses, carrying out scanner analyses and occasionally handling maintenance and delivery robots, with China being the main investor in these AI technologies. [12] Baidu released it's LinearFold AI algorithm [4] to vaccine developing teams during the early stages of the pandemic. The algorithm was built for predicting the RNA sequence of the virus and it does so in seconds.

3.2. Markov Decision Process

Markov decision processes have been used to solve real world problems with AI, this chapter talks about Markov chains, Markov decision processes and stochastic games.

3.2.1. Introduction to Markov chains

To explain what a Markov chain is, we first have to look at a certain type of process called a Markov process. A Markov process has no memory of it's past state, it's past state is irrelevant to what state the process will be in next. The current state of a Markov process is the only information that influences where the process is going to go, this property is commonly known as the Markov property. A Markov chain is the case of a Markov process having a countable set of possible states. [13]

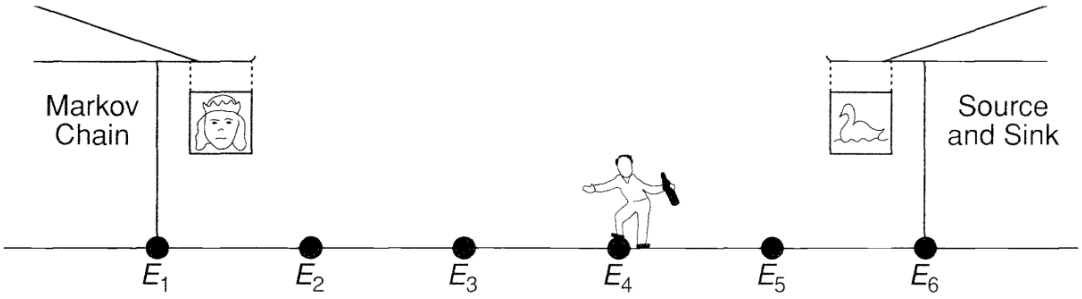

Figure 1: The drunkard problem [14]

Let's first look at a more trivial example of a Markov chain, it is the problem of a drunkard standing between two pubs [14], 'The Markov Chain' and 'The Source and Sink' as shown on figure 1. The drunkard either moves 10 meters towards 'The Markov Chain' with probability $\frac{1}{2}$ or he moves 10 meters towards 'The Source and Sink' with probability $\frac{1}{3}$. He can also stay where he is with the remaining probability of $\frac{1}{6}$. This random movement of the drunkard depicted trough probabilities is called a *one-dimensional random walk*. We assume that once he arrives at a pub the movement is done, we also assume that the pubs are 50 meters apart and that the drunkard is initially standing 30 meters from 'The Markov Chain' and 20 meters from 'The Source and Sink'. As indicated on figure 1 the places he can stop by are $E_1, ..., E_6$, we can think of these positions as states. The drunkard's initial position can be described as the vector $(0, 0, 0, 1, 0, 0)$. After one move the probabilities of his position give us the vector $(0, 0, \frac{1}{2}, \frac{1}{6}, \frac{1}{3}, 0)$, after two moves the vector $(0, \frac{1}{4}, \frac{1}{6}, \frac{13}{36}, \frac{1}{9}, \frac{1}{9})$. In our problem states E_1 and E_6 are considered *persistent* states as the process is done after arriving at them. The other states are considered

transient because we would still move after arriving at these states.

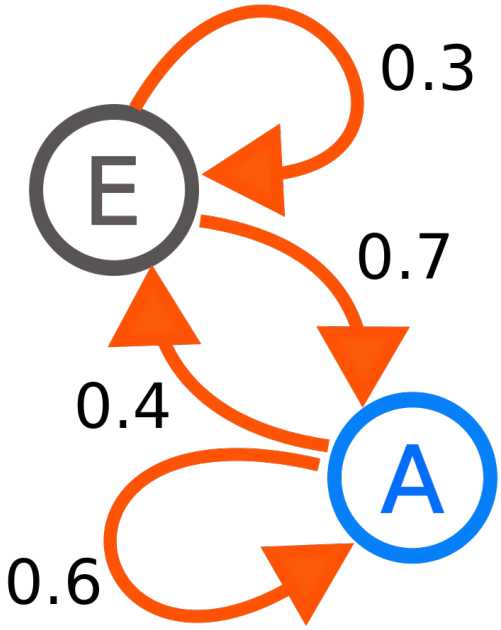

Figure 2: An example of a Markov chain [15]

Let's move on to Markov chains depicted on a simple graph. On figure 1 we can see two states: A and E. When on state A, we could stay on state A or move to state E, indicated by arrows. We have a sixty percent chance to stay on A, as indicated on the arrow originating on A and looping back to it. We have a forty percent chance to move from state A to state E as indicated on the arrow originating in A and pointing to E. We would have the same information without the arrows looping into the same state because the sum of the values on the arrows has to be equal to 1. For the purpose of demonstration these arrows have been kept, but we usually would not indicate them as they are already implied.

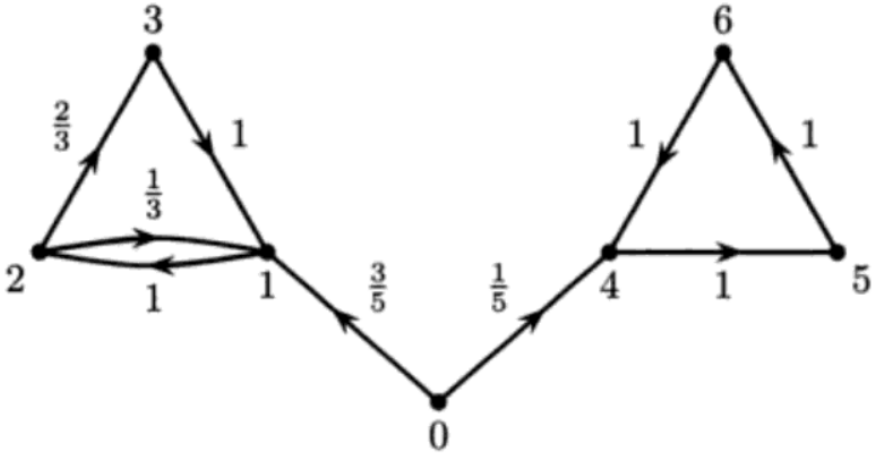

Figure 3: Discrete time Markov chain with closed classes [13]

Figure 2 represents what is called a *discrete time* Markov chain [13]. In this type of Markov chain, time it takes to move from one state to another is irrelevant, what is relevant is the probability of jumping from one state to another. Let's look at another example of a discrete time Markov chain.

On figure 3 we have a Markov chain with states indicated by numbers one trough six, it also doesn't include the loops to the same state which are implied. What we can notice that after moving from the starting point which is state 0, we would have no way to return back to state 0 as neither state 1 or state 4 have arrows pointing back to state 0. This is what is called a *closed class* or in other words a class that you can't escape. There are three classes in this example: {0},{1, 2, 3} and {4, 5, 6}, and the closed classes in this case would be {1, 2, 3} and {4, 5, 6}. In these examples the closed classes are also *recurrent*, which means you move to each state, returning again and again to every state. Class {4, 5, 6} is *periodic* as it always goes in this order: 4 -> 5 -> 6 and returns back to state 4. [13]

Figure 4 shows a different type of Markov Chain called a *continuous time* Markov chain [15]. In a continuous time chain the process makes a transition after the amount of time specified by the holding time — an exponential random variable, as opposed to a discrete time chain where moving states is taken in time steps.

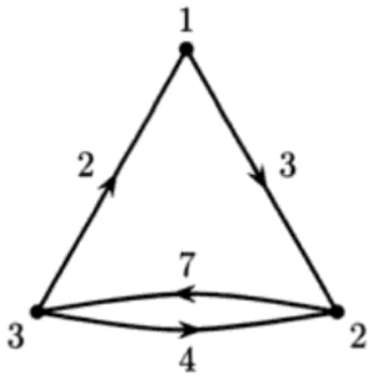

Figure 4: Continuous time Markov chain [13]

3.2.2. Markov decision processes

The Markov decision process expands what was defined with Markov chains as it satisfies the Markov property; what matters when moving states is only the current state. It is used to model decision making where the outcome is partly random and partly under the control of a decision maker. A Markov decision process contains the following [16]:

- A set of possible world states S,
- A set of possible actions A,
- A real valued reward function R(s,a).

The following text is an excerpt from page 10 of Jerzy Filar's and Koos Vierzes book 'Competitive Markov Decision Processes' that describes these three characteristics.

> "We shall consider a process P that is observed at discrete time points $t = 0, 1, 2, 3, ...$ that will sometimes be called stages. At each time point t, the state of the process will be denoted by S_e. We shall assume that S_t is a random variable that can take on values from the finite set $S = \{1, 2, ..., N\}$ which from now on will be called the state space. The phrase "the process is in state s at time t" will be synonymous with the event $St = s$. We shall assume that the process is controlled by a controller or a decision-maker who chooses an action $a \in A(8) = \{1, 2, ..., m(s)\}$ at time t if the process is in state s at that time. We may regard the action chosen as a realization of a random variable A_t denoting the controller's choice at time t. Furthermore, we shall assume that the choice of $a \in A(s)$ in state s results in an immediate reward or output $r(s, a)$, and in a probabilistic transition to a new state $s' \in S$." [17]

The difference between a Markov decision process and a Markov chain is the addition of actions and rewards. Actions allow the decision maker to make a choice what to do next, rewarding them with the appropriate reward after changing states. If each state would have only one action and the reward for making each action was the same, a Markov decision process would reduce to a Markov chain. [16]

Markov decision processes are closely tied to stochastic games. According to Eilon Solan a stochastic game "is played by a set of players. In each stage of the game, the play is in a given state, taken from a set of states, and every player chooses an action from a set of available actions. The collection of actions that the players choose, together with the current state, determine the stage payoff that each player receives, as well as a probability distribution according to which the new state that the play will visit." [18] Sounds familiar? Stochastic games extend Markov decision process' with competitive situations where more than one decision maker is involved. It may not sound like it, but many of the games we play today on computers and phones could be considered stochastic or at the very least to have stochastic properties. For example, chess is the most commonly known stochastic game in the world.

3.2.3. Markov Decision Processes and Artificial Intelligence

Markov decision processes and stochastic games were initially used to model real-life situations that arise in economics, political science, operation research and so on. Analysing the game modeled after these situations can provide with crucial and decisive predictions and recommended strategies.

Overexploitation means to harvest a renewable resource by multiple parties to the point of diminishing returns, in economics the overexploitation of a common resource between two or more parties is reffered to as 'the tragedy of the commons'. Solan and Vieille [18] mention an example of a fishery war between two countries. Fish would thrive without unnatural intervention, but the two countries were depleting the population of fish, it was approaching a

point where the fish would no longer be able to reproduce fast enough and would slowly die out. In any given area the amount of fish living there can be considered a state in terms of a Markov decision process. The countries can determine how many fish the fishermen can catch, this would be considered an action. If the countries cooperate, they would be rewarded with a larger population of fish and therefore a long-lasting food resource. This example has extensively been analysed and studied as a stochastic game as it has a clear definition of state-space, actions, rewards and it has two players.

Another example Solan and Vieille [18] mention is that of market games with money, namely a problem regarding inflation. Inflation is the rate at which a cost of something increases, typically the cost of living in a country. Study has been done on the origin of inflation trough a stochastic game model. At every stage of the game, each player or so called agent receives a random endowment of a perishable commodity. The agents then decide how much they want to lend to or borrow from the central bank and after the transaction they consume the amount of the commodity that they have. The conclusion of analysing this game was that to make inflation appear it is enough to have uncertainty regarding the random endowments at the start of each stage.

When dealing with a patient, healthcare workers have to make a decision from a myriad of different treatment options taking into account every bit of information they can, pertaining to their patient. Casey C. Bennett and Kris Hauser [19] approached making an AI framework that 'thinks like a doctor'. A computer is able to process large streams of information at once, something that humans can't do so well which is extremely useful for doctors when having to deal with a lot of patients at once, especially during the COVID-19 pandemic. Bennet's and Hauser's approach used Markov decision process' as a foundation, each patient has their own state, needs an assigned action performed and the reward for the decision maker, in this case the AI, would be helping a patient heal fully or at the very least reach better state. The results of this research demonstrated that this approach worked and that such an AI would reduce the costs of treatment and increase in the quality of choice of patient treatments.

3.3. League of Legends

Video games have dominated the entertainment industry for the last two decades, especially competitive multiplayer games. From the bunch of games spawned in the 2000s one thrived and stayed on top of the industry since 2009. Let's take a look at the biggest Esport of all time.

3.3.1. The biggest Esport of all time

League of Legends, also reffered to as *League* or just *LoL*, is a MOBA (Multiplayer Online Battle Arena) developed by Riot Games and publised in 2009 [20]. In a game of League two teams of five players battle it out to try and push trough the enemy base and destroy their *nexus*, a structure located inside the base which when destroyed, much like a check mate, ends the game and secures the victory. With low system requirements, availability and it's free-to-

play trait it has become not only the most played game in the world but also the most played Esport. 115 million people play the game in 2021 on a monthly basis, statistically it has at least one game started per second. [21]

Just as the Romans were inspired by ancient Greece, so were the founders of Riot Games, Brandon Beck and Mark Merill, inspired by a game mode for Warcraft III: Reign of Chaos. Warcraft III is a fantasy strategy game developed by Blizzard Entertainment, today Blizzard is Riot's competitor. Beck and Merill had an idea for a spiritual successor to one of the game's modes, called Defense of the Ancients (DotA), and just like the Romans, not only were they inspired, but they also refined the defining characteristics of their predecessors. They wanted to make a standalone game that would be supported over a longer period and one that didn't hold a price tag, so they hired people involved with DotA and went on to design League of Legends. [20] [22]

League of Legends was released in North America on October 27, 2009 instantly receiving praise and gaining popularity and player numbers, so much that in 2011 the first World Championship, or Worlds for short, was held at Dreamhack with a 100 000 USD prize pool. From then, it would only get more viewers, players and the prize pool each year was getting larger.

On League's viewership and franchising Wikipedia writes:

> "Online viewership and in-person attendance for the game's esports events outperformed those of the National Basketball Association, the World Series, and the Stanley Cup in 2016. For the 2019 and 2020 League of Legends World Championship finals, Riot Games reported 44 and 45 peak million concurrent viewers respectively. Harvard Business Review said that League of Legends epitomized the birth of the esports industry. As of April 2021, Riot Games operates 12 regional leagues internationally, four of which – China, Europe, Korea, and North America – have franchised systems. League games are typically livestreamed on platforms such as Twitch and YouTube. The company sells streaming rights to the game; the North American league play-off is broadcast on cable television by sports network ESPN." [20]

3.3.2. League of Legends and me

Like most things in the entertainment industry, I heard about League through a friend. I was eleven years old when I played my first game and for the last ten years I've been infatuated with it. Having played chess throughout middle school, the combination of strategy and competitiveness was something that instantly caught my attention, but unlike chess League had colourful graphics, fantasy and was played using a computer instead of an old-fashioned wooden board and pieces. It was at a time when League was only starting it's long rise to the top of the gaming industry but all of my friends had already been playing it and we would plan weekend nights to play together. I was at the top of my class in middle school with only a few A's out of reach from becoming a straight A student, but I wasn't very good at League just yet.

Freshman year of high school didn't bring many changes, my class was comprised of mostly girls and out of the six boys four of us played League. This game brought us together, during recess we would talk about the different combinations of characters, tactics and play styles. During the coming years I would study less and play more, my grades were dropping to mostly B-s but my League skills were slowly increasing. Motivation stemmed mostly from pure competition but our high school was holding a League tournament at the end of each year, we didn't win a single one but it was always something to look forward to. In the final year of high school I was able to reach the top 200 players on the smaller of two European servers and by sheer chance I got noticed by a few top ranked players in my region in Croatia.

We formed our own team and started playing in amateur leagues and tournaments, we were getting good enough to hold our ground against even the best teams in the Balkan region. Somewhere along the way we lost the will to continue playing and would soon part ways. University came along, it was holding it's own League tournament and once again somehow I found myself playing competitively for the Faculty of Organization and Informatics League team. This time, we were consistently winning the university tournament which lead to a dean's award in sports despite many not considering Esport to be a real sport.

Gaming addiction is a real issue, but for me playing League was always something I did and if I wasn't doing it I would be doing something similar. To get good at something you have to practice it, and to stay good you have to practice regularly. I don't regret my time playing League but too often I've been told that I am either wasting my life or that I was addicted to it. League taught me skills that transcend the gaming world, to win the most games you sometimes have to calm somebody on your team down trough only chat messages, your mind and body should be in good condition if you want the highest chance of winning, starting a game usually means committing anywhere from twenty minutes to an hour of free undisturbed time and above all that you have to make many small calculations in a short amount of time and react quickly to what is happening on the screen. It has become increasingly important to fight gaming addiction in children and in some adults, it is also important to distinguish between addiction and commitment to a hobby. Playing video games all day hurts the body especially the sitting and staring at the screen part, so it is essential to take breaks and to exercise. It was obvious to me even as a kid that I would at some point either have to address these issues or face them and luckily I addressed them fairly quickly.

Using my knowledge about the game I have created a program using the programming language Python that plays League, my own League of Legends bot. The difference between League and games like chess and Go is that in League you have to not only make a good decision you also have to react to what is happening in game, there is also no taking turns but you have to constantly change what you're doing according to all the given information.

3.3.3. Welcome to Summoner's Rift

The flagship game mode in League of Legends is played on a square map called Summoner's Rift. The map is split diagonally so that on the bottom left is one team's territory, or the blue side and in the top right is the other team's territory, or the red side. Because the map is

symmetrical there is no important difference that would tip the scales in favor of one side, but there are minor differences. The map contains three lanes, two of the three follow the edge of the map and are known as the top and bottom lane. The third lane is the fastest way from one base to the other and follows the second diagonal, it is known as the middle lane. The areas between the lanes are under one name called the jungle.

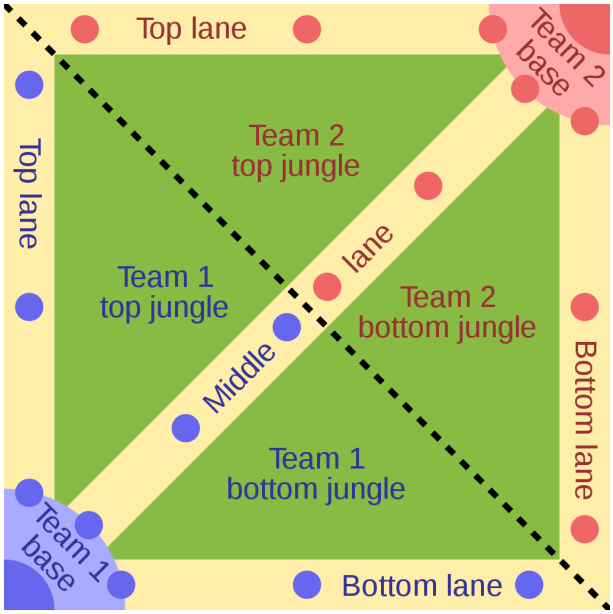

Figure 5: Layout of Summoner's Rift; source: [20]

Each team consists of five players, each controlling a unique character called a *champion*. There are more than a hundred different champions in League, all with different abilities and effects. Champions usually have two main resources. One is standard across all champions and it is referred to as HP (short for hit points) or just health. When a champion's health reaches zero they die and a short amount of time has to pass before the player can use them again. This reappearance effect after dying is commonly known in video games as respawning. The other resource depends on the design of the champion, but the most common one is mana. Mana is used to cast abilities, when you're out of mana you don't die, but you can't cast any abilities. Both health and mana can be refilled at the corner of the allied base called the fountain where you initially start the game from and where all allied champions respawn. Champions passively also regenerate a small amount of health and mana wherever they are, although it is not enough to be able to stay on the map without returning to the fountain indefinitely.

All champions start the game at level 1. Champions get stronger by leveling up which they achieve trough experience points, or XP for short. Level 18 is the maximum a champion can be. XP is gained trough a few ways, the first of which is killing enemy champions. The second way is by being around minions when they die, dealing the killing blow to a minion is not required to get XP but that will come into play later. Minions are computer controlled units that spawn in waves and walk down each of the lanes towards the enemy base until they meet and attack enemy minions, structures or champions along their path. Minions have health and just like champions will die if it reaches zero. The third way of getting XP is by killing monsters also known as neutral camps inside the jungle. Neutral camps respawn at designated locations on

the map after a set amount of time. There are also two neutral objectives; Baron Nashor and Dragon. These objectives take longer to kill but are worth the effort because of their impactful rewards.

Figure 6: Minions fighting in the middle lane; source: [23]

Another important aspect of League is gold. All players get some gold at the start of the game, and generate a small amount per second. Like XP, gold is gained trough killing enemy champions and neutral camps but you can also gain gold from destroying enemy structures. Minions also give gold but only upon dealing the killing blow, unlike XP for which you only need to be in close proximity. Gold can be used to buy items in the store, which is located next to each fountain. Items, when purchased, stay in your inventory and are an effective way to power up your champion, a champion's strength inside a game is usually approximated by looking at their level and the items they have purchased.

Three types of structures exist in League: turrets, inhibitors and the nexus. There are eleven turrets on each side of the map, represented on Figure 1 as small circles, there are two turrets defending each nexus. Turrets target enemy minions and champions with a hard hitting periodic attack and are stronger than champions for the entirety of the game, therefore they need to be taken down with the help of minions, your own turrets can be used for safety because of their inherent strength. Inhibitors are located behind each of the inner turrets inside each base, there are six of them in total. When an enemy inhibitor is destroyed, for the next few minutes a stronger minion called a super minion spawns with each allied minion wave in that lane, accelerating the push to victory. The final structure, the nexus, when destroyed ends the game with a victory for the team that managed to destroy it.

The five players in a team each serve their own unique purpose. The game has evolved in a way that there is not much deviation from five roles in League, which are Top, Jungle, Mid,

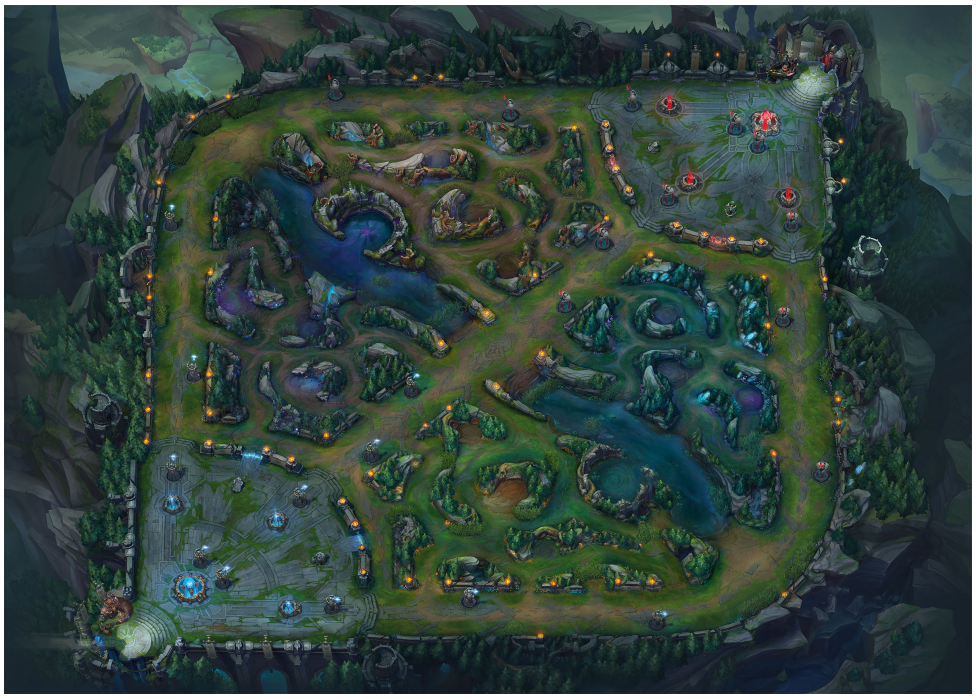

Figure 7: Summoner's Rift in game; source: [23]

ADC and Support. Top lane is mostly reserved for champions who can sustain damage and champions who can fight on their own. Mid lane is where long range artillery mages and front loaded burst champions meet. On the bottom lane reside two champions; an Attack Damage Carry (ADC) or Marksman who relies on their basic attacks for damage that gets better the longer the game goes on and a supportive champion that protects the ADC or a champion able to lock opponents down and make plays. Lastly, the jungle role is tasked with collecting XP and gold from the neutral camps in the jungle and help out the other lanes or try to outnumber the enemy in a lane to score kills and take objectives.

3.3.4. League of Bots

To play League, you would have to create an account. Every account starts at level 1 (this is different from the in game level of a champion). As a requirement for playing League's ranked mode the account has to be at least level 30. Because of the in game chat feature, almost every game there is at least one person on each team who will vent their frustrations and lash out in the form of toxic chat messages. Toxicity is a defining characteristic of the League of Legends player base and Riot has systems in place that detect toxic behavior and give out two week bans although the systems are criticised by many for being insufficient.

Continuous toxic behaviour can get an account banned permanently, when that happens the toxic player will have to create another account but that would mean playing about fifty games that last 30 minutes on average until the player can enter the ranked mode again. For this player these fifty games are a waste of time, which is why they look to buy a level 30 account online. Buying and selling League accounts is against the Riot Games Terms of Service but as long as there is a market for it, leveling accounts with bots or *botting* will continue to

exist.

3.4. S.A.M.F.I.E. (Smart Artificial Miss Fortune Intelligence Emulator)

League of Legends is daunting to new players because of it's complexity, but not to S.A.M.F.I.E. who conquered it. S.A.M.F.I.E. is my own League bot, and the topic of this chapter, along with how a Markov decision process model was used to improve it.

3.4.1. Miss Fortune

Before we can move to how S.A.M.F.I.E. actually works, we first need to look at the character I decided it would be playing. This character is an ADC, meaning she excels at dealing lots of damage from further away than most characters. Her name is Miss Fortune, and yes, she is a pirate.

Figure 8: Screenshot of Miss Fortune in game;

Miss Fortune's passive ability is called Love Tap and states that every time she attacks a target that is different from her previous target the attack will deal extra damage. This means that switching between targets when playing Miss Fortune will net more damage as opposed to only attacking one target at a time. Even if you don't switch targets, the extra damage will apply at least once, so even a less experienced player can use it somewhat reliably.

The official text of Love Tap, Miss Fortune's passive ability:

"INNATE: Miss Fortune's next basic attack against an enemy that was not the target of her previous basic attack will deal 50% - 100% (based on level) AD bonus physical damage, halved to 25% - 50% (based on level) AD against minions." [24]

Miss Fortune's first basic ability or Q ability is called Double Up (abilities are commonly known by the key that needs to be pressed in order to use it). Double Up is a targeted ability, when used it fires a shot at the target and then looks for a second target behind the first to bounce to. It synergises with the passive ability as it applies the extra damage to both targets, after it hits the second target the first one will again be eligible for the passive damage. This ability has a low mana cost and is on a low cooldown (you can use it often), which means you can use it almost whenever it is available, without being worried about running out of mana (Miss Fortune's secondary resource).

The official text of Double Up, Miss Fortune's first basic ability:

"ACTIVE: Miss Fortune fires a shot at the target enemy that deals physical damage upon its arrival, which then bounces to hit another enemy behind it. The shot applies on-hit effects at 100% effectiveness to both enemies hit, and applies on-attack effects to the first enemy hit." [24]

Miss Fortune's second basic ability or W ability is called Strut, it has a passive, and an active component. Passively Miss Fortune moves faster if she hasn't taken any damage in the last five seconds. When the ability is activated, she will gain this movement speed even if she took damage in the last five seconds. Additionally she attacks faster for the next four seconds.

The official text of Strut, Miss Fortune's second basic ability:

"PASSIVE: Miss Fortune gains 25 bonus movement speed if she has not taken persistent damage in the last 5 seconds. This bonus is increased to a greater amount after another 5 seconds.

ACTIVE: Miss Fortune gains bonus attack speed for 4 seconds and brings Strut's passive effect to full power." [24]

Marking a new target with Love Tap reduces Strut's cooldown by 2 seconds.

Miss Fortune's third and final basic ability, or her E ability, is called Make It Rain. In a target location Miss Fortune fires a rain of bullets that periodically deal damage and slow enemies. This is useful when chasing enemies down in combination with the Strut passive movement speed.

The official text of Make It Rain, Miss Fortune's third basic ability:

"ACTIVE: Miss Fortune rains down hundreds of bullets onto the target location for 2 seconds, granting sight of the area and dealing magic damage to all enemies within every 0.25 seconds and slowing them." [24]

Miss Fortune's ultimate ability or R ability is called Bullet Time, and it is one of League's most damaging abilities. Miss Fortune fires waves of bullets in a cone in front of her for three seconds, dealing tons of damage. It's drawback is that without the help of other abilities, enemies can move out of the way, but when used against other bots that don't register it immediately, it is extremely impacting.

The official text of Bullet Time, Miss Fortune's ultimate ability:

> "ACTIVE: Miss Fortune channels for up to 3 seconds, firing several waves of bullets in a spread of 6 projectiles per wave in the target direction, with each wave dealing 75% AD (+ 20% AP) physical damage to enemies hit." [24]

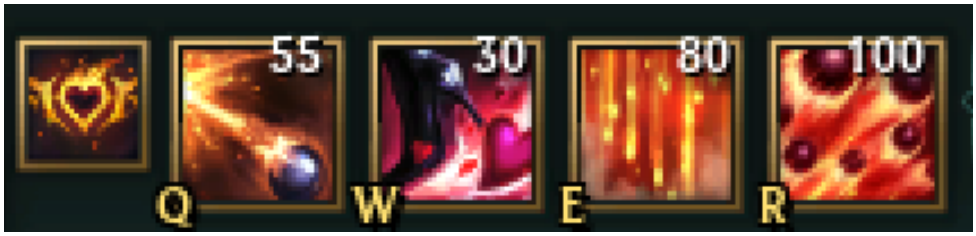

Figure 9: Screenshot of Miss Fortune's abilities, as shown in game;

Miss Fortune is considered a simple champion. Having two passive abilities helps, Make It Rain and Bullet Time are also lethal when used in conjunction for anything that stays inside the abilities' reach, and the combo can devastate an entire team. Her W is in almost all cases used whenever she enters battle with a champion. Her Q ability is arguably the most complex thing about her, it has to be used in an innovative way to be most effective because of it's unique bounce property and it's synergy with her passive. Use of her Q is where good Miss Fortune players shine. My bot is nowhere near a good Miss Fortune player.

The reasons behind choosing Miss Fortune are her simplicity, her basic ability combinations and her being a ranged character. As stated above, three of her abilities need only to be pointed at the enemy, the other ability, while her W, needs only to be activated. Bullet Time, if used properly, can kill an enemy from full health if used at the right angle. Her range is very useful, she can stay away from danger while slowly dwindling enemy resources. All of these properties made her an appropriate choice for a League AI. Having tested with many other champions, I found that Miss Fortune was the most consistent choice.

Runes are a set of power-ups a player can choose before going into a game. Figure 10 shows the runes I chose for the AI to use with short descriptions. Without going into too much detail on the runes, the ones I chose are simple to use, and most of them help deal damage, and are useful for the attack patterns I had my bot use.

Items are also an important factor. Figure 11 shows the items I had S.A.M.F.I.E buy, they are a combination of common items purchased when playing Miss Fortune and items that have very similar build paths. Expensive items are made up of smaller, less expensive items that combine into the expensive item. The build path of an item is essentially all the smaller items needed to combine into that specific item. You can of course purchase the item without buying it's components if you have enough gold for the full cost.

Figure 10: Runes I had S.A.M.F.I.E. use;

Figure 11: Items I had S.A.M.F.I.E. buy;

3.4.2. The first version

I started developing S.A.M.F.I.E. before starting to work on this paper. The first thing to figure out was: how will my program detect what is on the screen. Consulting with the internet, I found that a Python library exists that does the detection automatically trough a few different functions. Python is an interpreted, object-oriented, high-level programming language and the library in question is pyautogui. Pyautogui allowed me to not only detect images on the screen but also check the RGB values of any specific pixel on the screen. Next, I needed to determine which images I needed to detect, so that my bot can find and click on minions, champions and turrets. As these things were always in motion relative to the camera, it wasn't as simple as taking a picture of these objects and trying to find them using pyautogui. Thankfully, I could

take images of the corners of the health bars of these objects and detect them trough those. Health bars always stay on top of everything else on screen, so it was a nice solution to my detection problem.

Figure 12: Some images I used to detect various things on the screen, from the left: ally champions, ally minions, enemy champions, enemy minions, enemy structures, enemy plated turrets, store button, ultimate ability button;

Next, I had to define the logic my bot would work by. I worked on the logic part directly inside the code, which in hindsight wasn't efficient and left me with a less 'smart' bot. If the program detected a minion it would attack the minion, if it detected a turret it attacked the turret and so on. This worked fine in games without enemy champions, it would win the game, but it was losing when it had an opponent. There were some problems that pyautogui couldn't solve. For example, pyautogui would detect the first image from the top of the screen, but there were lots of minions on the screen and it couldn't tell me how many minions there were. The detection was also extremely slow, League is a fast paced game and if the bot had too many things to detect, it would make it react slowly to what was happening on screen. The bot stayed in this initial version, and at the time I started working on this thesis and learning about Markov chains and Markov decision processes.

3.4.3. Applying a Markov Decision Process model to S.A.M.F.I.E.

When I came back to working on the bot, I was equipped with more knowledge and had more ideas to make the bot work better. Instead of using pyautogui for screen detection I used OpenCV, OpenCV is another Python library, but using it allowed me to detect multiple instances of the same image on one screenshot, meaning I could detect all the minions on the screen as opposed to only seeing one minion with pyautogui. I found out about OpenCV trough the YouTube channel "Learn Code by Gaming" [1], and I modified their code to suit what I needed to detect with S.A.M.F.I.E. First thing I did was write up a pseudocode for the logic I wanted my bot to work on. Writing this pseudocode proved to be far better than working by heart and hence it yielded better results.

Listing 1: Pseudocode I based S.A.M.F.I.E. on

```
1 lvl up if can
2 buy items if can
3 if low hp back
4
5 if see turret
6         if no friendly minions
7                 retreat
8         if no enemy minions and no enemy champions
```

```
                attack enemy turret
        if enemy minions and no enemy champions
                attack enemy minions
        if enemy minions and enemy champions
                retreat

else
        if no enemy minions and no enemy champions
                move forward
        if enemy minions and no enemy champions
                walk behind my wave
                clear enemy minions
        if there are friendly minions
                if enemy minions and enemy champions
                        if num of friendly minions > num of enemy minions + 1
                                attack enemy champion
                        else
                                attack enemy minions
        else
                retreat until friendly minions
```

Another thing I did was use newfound knowledge of Markov decision processes and stochastic games to determine what states S.A.M.F.I.E. would be moving to and from, as well as the actions it would be taking and rewards it would gain. The states I landed on were as follows: attacking turrets, attacking champions, farming, going back, moving forward and retreating. As the randomness of moving from one state to the other is determined by the screenshot the bot takes, it is hard to quantify the probabilities of moving from one state to another without historic data. Rewards are also random and determined trough the game, not every action will lead to a reward. Killing minions, champions and destroying turrets gives gold to the champion, and gold can be seen as a reward, but it all depends on whether or not the bot's action works for that situation, so the bot works on a priority system that decides what action the bot will take. Most of the program is contained inside a while loop that repeats itself. What is not inside the while loop are preparations the bot does at the start of the game, like buying the starting item and waiting for the initial minion wave.

The going back state appears at the very start of the while loop, S.A.M.F.I.E. checks if it's health bar is low by checking a specific pixel on the health bar. If it's health is low it will use the movement speed boost on W and it's defensive ability Barrier located on the F key that gives it a shield. S.A.M.F.I.E. then starts moving away, towards it's base, until there are no enemy champions and enemy minions, and meanwhile always checks if it has at any point died. If it hasn't died it presses B, which is the recall button. Recalling returns S.A.M.F.I.E. to the ally base after 8 seconds of waiting, where it waits a bit more to gain health and mana, before continuing to move forward. This means the next state after the going back state is always going to be the move forward state.

Listing 2: Going back state

```
    if pixelLowHP == (1, 13, 7) or time.time() - backTimer >= 300:
        dead = False
```

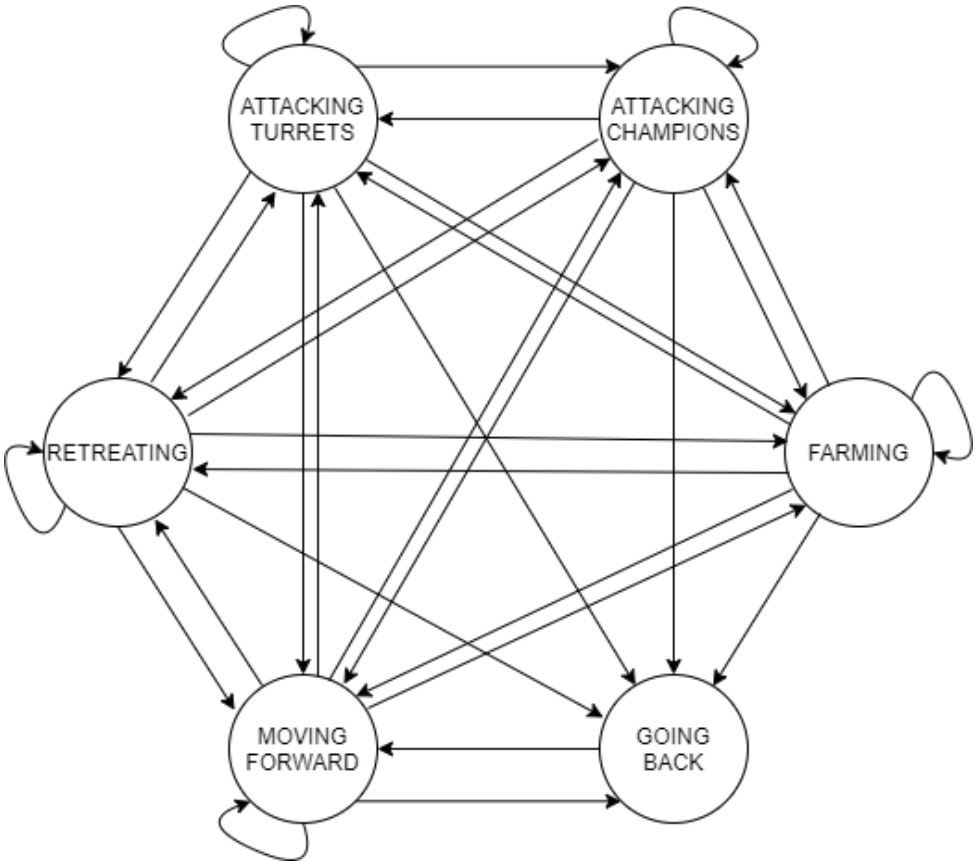

Figure 13: Markov chain modelled for S.A.M.F.I.E;

```
        rightClick(1666, 1056)
        useW()
        if pixelLowHP == (1, 13, 7):
            useF()
        while len(enemy_champions) or len(enemy_minions):
            rightClick(1666, 1056)
            screenshot = wincap.get_screenshot()
            enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
            enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
            pixelDead = pyautogui.screenshot().getpixel((711, 1032))
            if pixelDead == (185,185,185):
                dead = True
            sleep(0.5)
        if dead == False:
            keyboard.press("b")
            sleep(0.01)
            keyboard.release("b")
            sleep(13)
            backTimer = time.time()
            GOING_BACK += 1
        else:
            while pixelDead == (185,185,185):
                pixelDead = pyautogui.screenshot().getpixel((711, 1032))
            backTimer = time.time()
        continue
```

The moving forward state happens when there are no enemy targets on the screenshot S.A.M.F.I.E. takes. Before moving forward, it checks if the shop icon is on the screen and if it locates the shop button it first buys items. The shop icon appears only when S.A.M.F.I.E. is in the ally base.

Listing 3: Moving forward state

```
if pyautogui.locateOnScreen('shop.png', confidence=0.95):
    if doransSold == False and gameTimer > 1200:
        doransSold = True
        rightClick(1090,1010)
    buyItem()
    sleep(1)
rightClick(1885, 842)
MOVING_FORWARD += 1
continue
```

The attacking champions state occurs when there are only enemy champions on screen without enemy minions or when S.A.M.F.I.E. has gained a minion advantage, meaning that ally minions outnumber enemy minions. It attacks champions in a pattern, it starts by using E and W to catch up them, following with Q and Ignite, which is it's offensive spell located on D. If after all that the enemy is still alive, S.A.M.F.I.E. checks if it's ultimate ability is available and uses it if it is.

Listing 4: Attacking champions state

```
rightClick(ally_minions[0][0]+30,ally_minions[0][1]+60)
sleep(0.2)
if len(enemy_champions):
    win32api.SetCursorPos((enemy_champions[len(enemy_champions)-1][0]+20,
     enemy_champions[len(enemy_champions)-1][1]+150))
    useE()
    leftClickAA(enemy_champions[len(enemy_champions)-1][0]+40,enemy_champions[len(
     enemy_champions)-1][1]+120)
    useW()
    sleep(1)
keyboard.press("s")
sleep(0.01)
keyboard.release("s")
screenshot = wincap.get_screenshot()

enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
if len(enemy_champions):
    leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,enemy_champions[len(
     enemy_champions)-1][1]+80)
    useQ()
    useD()
    leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,enemy_champions[len(
     enemy_champions)-1][1]+80)
    sleep(1)
screenshot = wincap.get_screenshot()
enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
ultimate_available = vision_ultimate.find(screenshot, 0.98)
```

```
24 if len(enemy_champions):
25     leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,enemy_champions[len(
        enemy_champions)-1][1]+80)
26     if len(ultimate_available):
27         useR()
28         sleep(3)
29 ATTACKING_CHAMPION += 1
30 continue
```

The farming state could also be called the attacking minions state, but farming is League terminology for just that. S.A.M.F.I.E. attacks minions when it hasn't gained a minion advantage and when there are no enemy champions on screen. In between attacks it also moves behind her own minion wave so that it is always in a good position.

Listing 5: Farming state

```
1 rightClick(ally_minions[len(ally_minions)-1][0],ally_minions[len(ally_minions)
   -1][1]+80)
2 sleep(0.1)
3 screenshot = wincap.get_screenshot()
4 enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
5 if len(enemy_minions):
6     rightClick(enemy_minions[len(enemy_minions)-1][0]+30,enemy_minions[len(
       enemy_minions)-1][1]+60)
7     useQ()
8 FARMING += 1
9 continue
```

The retreating state is different from the going back state. Retreating happens when there are enemies and no ally minions. S.A.M.F.I.E. moves back for a short amount of time, but doesn't completely return to base. This happens when the area isn't safe enough to be offensive, but is dangerous enough to warrant a retreat.

Listing 6: Retreating state

```
1 rightClick(1666, 1056)
2 sleep(0.2)
3 RETREATING += 1
4 continue
```

The attacking turrets state occurs when there are no enemy champions, no enemy minions and at least two ally minions. Of course, an enemy turret has to be present as well. If this case happens, S.A.M.F.I.E. will attack the turret until the situation changes.

Listing 7: Attacking turrets state

```
1 screenshot = wincap.get_screenshot()
2 enemy_turret = vision_enemyTurret.find(screenshot, 0.95)
3 enemy_turretPlate = vision_enemyTurretPlate.find(screenshot, 0.95)
4 if len(enemy_turret):
5     rightClick(enemy_turret[len(enemy_turret)-1][0]+75,enemy_turret[len(enemy_turret
       )-1][1]+150)
6
```

```
7  if len(enemy_turretPlate):
8      rightClick(enemy_turretPlate[len(enemy_turretPlate)-1][0]+75,enemy_turretPlate[
       len(enemy_turretPlate)-1][1]+150)
9  ATTACKING_TURRET += 1
10 continue
```

The priority system works this way: if the screenshot the S.A.M.F.I.E. takes has an enemy turret in it, it checks if there are any ally minions because it can't tackle the turret alone, and if there are no friendly minions it will retreat until it sees ally minions. If there are ally minions and enemy minions next to the turret, S.A.M.F.I.E. will prioritise the minions. If there is at least one enemy champion next to the turret, S.A.M.F.I.E. will retreat. This way I minimised the deaths S.A.M.F.I.E. would sometimes have next to turrets (it still sometimes dies to the turret regardless). If there are no turrets, S.A.M.F.I.E. will look to gain a minion advantage on her opponent, before attacking their champion. If there are no turrets and no enemy champions, S.A.M.F.I.E. will just try to kill all enemy minions when there are ally minions nearby. A video of S.A.M.F.I.E. playing is available on YouTube at: https://www.youtube.com/watch?v=KAZ0gmMLlVg.

I had the program count how many times it entered each state. Using this historic data I could make a very rough estimate of what probabilities each state had when moving to other states. I entered this data into a probability matrix that corresponds to the Markov chain in figure 13.

	T	C	R	F	M	B
T	0.40	0.05	0.20	0.15	0.05	0.05
C	0.05	0.50	0.20	0.10	0.05	0.10
R	0.03	0.07	0.30	0.05	0.40	0.05
F	0.05	0.30	0.05	0.50	0.08	0.02
M	0.05	0.05	0.05	0.1	0.70	0.05
B	0.00	0.00	0.00	0.00	1.00	0.00

Figure 14: Probability matrix for S.A.M.F.I.E. states;

How to read the probability matrix on figure 14: the letters on the edge are the aforementioned states (T = attacking turrets, C = attacking champions, R = retreating, F = farming, M = moving forward, B = going back). Let's say I want to look at what probability S.A.M.F.I.E. has when moving from state F to state M. I would look at the row containing state F, that is the fourth row, and would then look for column M. The value corresponding from that movement between states is 0.08, meaning that S.A.M.F.I.E. has an 8 per cent chance to move from state F to state M. These values are again, rough estimates.

3.4.4. Testing and issues

The first goal I set for S.A.M.F.I.E. was to beat a game without any other player or bot inside the game. This was a simple goal, S.A.M.F.I.E. in it's current iteration can beat this anywhere from 15 to 25 minutes, if a human player accomplished this in that amount it would be considered slow, but for a bot it is a satisfying result.

The second goal was for S.A.M.F.I.E. to beat a game of intro bots. Intro bots are the new player friendly version of League's bot, this game mode is littered with botting accounts so it will usually be a game of ten bots fighting each other. Intro bots pretty much let the players win, they are very easy to beat. S.A.M.F.I.E. never lost in this game mode and would most likely win in ninety nine out of a hundred games.

The third goal, and the most challenging, was for S.A.M.F.I.E. to beat a game of beginner bots. S.A.M.F.I.E. suffered great hardships in this mode. Beginner bots are also easy to beat for a human player, but they won't let you win all the time like intro bots. During testing with beginner bots was when most of the tweaking of the attack pattern S.A.M.F.I.E. uses happened and other minor details were being fixed. In it's current iteration, S.A.M.F.I.E. can beat beginner bots in about 80 to 90 per cent, but what I didn't expect was for S.A.M.F.I.E. to make, what is called a pentakill. A pentakill is when a single player kills all five of the players on the opposing team, S.A.M.F.I.E. accomplished this against beginner bots, and it made me incredibly proud. Although I do admit that it was primarily luck that the enemy team of bots stuck together and died from S.A.M.F.I.E.'s ultimate ability.

Figure 15: Screenshot of S.A.M.F.I.E. seconds before scoring a pentakill;

Although S.A.M.F.I.E. works well, there are a couple of issues regarding the image processing it uses. A few times per game S.A.M.F.I.E. would detect champions or turrets when they weren't there. Another issue is when minions or champions walk next to the turret, their health bars will stay on top of the turret's health bar, meaning S.A.M.F.I.E. would play as if there was no enemy turret, and would die from it, as shown on figure 16. These issues are hard or impossible to avoid, so I am fairly satisfied with the results S.A.M.F.I.E. accomplished. The full

code of the bot can be found appended at the bottom of this document.

Figure 16: Screenshot of minion health bars overlapping with the turret health bar;

4. Conclusion

Artificial intelligence is affecting the world now more than ever, and so are computer games. These two industries are intertwined in many ways, some ethical, others not so much. Companies are trying to make AI that play games autonomously, when such a program is developed, human players of the game stop being relevant when talking about optimal, highest level of play. This is especially damaging to online multiplayer games that aren't able to recognise when an AI is playing, cheaters in these games will thrive.

This thesis started with a general exploration into AI science, Markov decision processes and League of Legends. The goal of this project was to apply a Markov decision process model to an autonomous AI player. Through examples of bots made by other people, I made my own bot for my favorite game, League of Legends. I called the bot S.A.M.F.I.E. and made it play against AI in special game modes of the game, to try and see how far I can push my own bot.

Before the final version of S.A.M.F.I.E. was made, I modelled a Markov decision process after the gameplay of League of Legends. A Markov decision process consists of states, in my case an example of a state would be attacking an enemy, it also consists of probabilities the process has when moving to another state. To move to a different state, my bot would need to make an action, another property of Markov decision processes, an example of an action would be when trying to attack an enemy the bot has to right click on the enemy. Rewards also come into play trough in game currency; gold. Killing enemies awards the player with gold, so I set up a priority system for the bot to know who to attack and when it should attack them.

S.A.M.F.I.E. works well and can convincingly beat the AI that usually new players play against in the CO-OP vs AI mode in League of Legends. It was made using Python, a programming language that allowed me to use OpenCV, which is a module for object recognition in images. S.A.M.F.I.E. takes screenshots periodically and based on the image and objects it detects, makes a decision and executes specific commands.

For development of S.A.M.F.I.E., modelling a Markov decision process and defining the states the bot could operate in helped me solve what logic, decisions and priorities I would need to take in account when writing the code. Markov decision processes proved useful enough to use when creating AI that plays games, and in general any real life situation that can be gamified.

Bibliography

[1] (). "Opencv object detection in games python tutorial," [Online]. Available: https://www.youtube.com/watch?v=KecMlLUuiE4&list=PL1m2M8LQlzfKtkKq2lK5xko4X-8EZzFPI (visited on 08/13/2021).

[2] (2020). "How to make advanced image recognition bots using python," [Online]. Available: https://www.youtube.com/watch?v=YRAIUA-Oc1Y (visited on 08/15/2021).

[3] (). "Artificial intelligence," Investopedia, [Online]. Available: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp (visited on 08/15/2021).

[4] (). "Artificial intelligence," Built In, [Online]. Available: https://builtin.com/artificial-intelligence (visited on 08/15/2021).

[5] J. McCarthy, "What is artificial intelligence?," 1998.

[6] (). "Intelligence," Oxford Learner's Dictionaries, [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/intelligence (visited on 08/15/2021).

[7] G. Oppy and D. Dowe, "The Turing Test," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., Winter 2020, Metaphysics Research Lab, Stanford University, 2020.

[8] (2017). "The history of artificial intelligence," Harvard University, [Online]. Available: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ (visited on 08/15/2021).

[9] D. Mhlanga, "Artificial intelligence in the industry 4.0, and its impact on poverty, innovation, infrastructure development, and the sustainable development goals: Lessons from emerging economies?" *Sustainability*, vol. 13, no. 11, 2021, ISSN: 2071-1050. DOI: 10.3390/su13115788.

[10] (2017). "Artificial intelligence autopilot," Tesla, [Online]. Available: https://www.tesla.com/AI (visited on 08/15/2021).

[11] Wikipedia contributors, *Autonomous racing — Wikipedia, the free encyclopedia*, [Online; accessed 30-August-2021], 2021.

[12] (2020). "Ai and control of covid-19 coronavirus," Council of Europe, [Online]. Available: https://www.coe.int/en/web/artificial-intelligence/ai-and-control-of-covid-19-coronavirus (visited on 08/15/2021).

[13] J. R. Norris, *Markov chains*, 2. Cambridge university press, 1998.

[14] R. J. Wilson, *Introduction to Graph Theory*. New York: Prentice Hall/Pearson, 2010, ISBN: 027372889X 9780273728894.

[15] Wikipedia contributors, *Markov chain — Wikipedia, the free encyclopedia*, [Online; accessed 4-September-2021], 2021.

[16] ——, *Markov decision process — Wikipedia, the free encyclopedia*, [Online; accessed 8-September-2021], 2021.

[17] J. Filar and K. Vrieze, *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[18] E. Solan and N. Vieille, "Stochastic games," *Proceedings of the National Academy of Sciences*, vol. 112, no. 45, pp. 13 743–13 746, 2015.

[19] C. C. Bennett and K. Hauser, "Artificial intelligence framework for simulating clinical decision-making: A markov decision process approach," *Artificial intelligence in medicine*, vol. 57, no. 1, pp. 9–19, 2013.

[20] Wikipedia contributors, *League of legends — Wikipedia, the free encyclopedia*, [Online; accessed 31-August-2021], 2021.

[21] (2021). "Total league of legends player count." [Online; accessed 31-August-2021], [Online]. Available: https://leaguefeed.net/did-you-know-total-league-of-legends-player-count-updated/ (visited on 08/31/2021).

[22] ——, *Warcraft iii: Reign of chaos — Wikipedia, the free encyclopedia*, [Online; accessed 1-September-2021], 2021.

[23] (2021). "Map (league of legends)." [Online; accessed 2-September-2021], [Online]. Available: https://leagueoflegends.fandom.com/wiki/Map_(League_of_Legends) (visited on 09/02/2021).

[24] (). "League of legends wiki," [Online]. Available: https://leagueoflegends.fandom.com/wiki/League_of_Legends_Wiki (visited on 09/15/2021).

List of Figures

List of Listings

Appendices

1. Code of ’main.py’

```
from time import sleep
import time

from pyscreeze import pixel
from windowcapture import WindowCapture
from vision import Vision
import win32api, win32con
import keyboard
import pyautogui

def rightClick(x,y):
    win32api.SetCursorPos((x,y))
    win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN,0,0)
    sleep(0.05)
    win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP,0,0)

def leftClickAA(x,y):
    keyboard.press("a")
    sleep(0.01)
    keyboard.release("a")
    win32api.SetCursorPos((x,y))
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
    sleep(0.05)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)


def lvlUp():
    keyboard.press("ctrl+r")
    sleep(0.05)
    keyboard.release("r")
    keyboard.press("ctrl+q")
    sleep(0.05)
    keyboard.release("q")
    keyboard.press("ctrl+e")
    sleep(0.05)
    keyboard.release("e")
    keyboard.press("ctrl+w")
    sleep(0.05)
    keyboard.release("w")
    keyboard.release("ctrl")
    sleep(0.05)

def buyItem():
    keyboard.press("z")
    sleep(0.2)
    keyboard.release("z")
    sleep(0.1)
    rightClick(600,340)
    sleep(0.05)
    rightClick(665,340)
```

```
    sleep(0.05)
    rightClick(720,340)
    sleep(0.05)
    rightClick(775,340)
    sleep(0.05)
    rightClick(830,340)
    sleep(0.05)
    rightClick(885,340)
    sleep(0.05)
    rightClick(940,340)
    sleep(0.05)
    rightClick(600,460)
    sleep(0.05)
    rightClick(665,460)
    sleep(0.05)
    rightClick(720,460)
    sleep(0.05)
    rightClick(775,460)
    sleep(0.05)
    rightClick(830,460)
    keyboard.press("z")
    sleep(0.2)
    keyboard.release("z")

def buyStartingItem():
    keyboard.press("z")
    sleep(0.2)
    keyboard.release("z")
    sleep(0.1)
    rightClick(600,340)
    sleep(0.05)
    win32api.SetCursorPos((1018,156))
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
    sleep(0.05)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)
    keyboard.press("z")
    sleep(0.2)
    keyboard.release("z")

def useQ():
    keyboard.press("q")
    sleep(0.01)
    keyboard.release("q")

def useW():
    keyboard.press("w")
    sleep(0.01)
    keyboard.release("w")

def useE():
    keyboard.press("e")
    sleep(0.01)
    keyboard.release("e")
```

```
104
105 def useR():
106     keyboard.press("r")
107     sleep(0.01)
108     keyboard.release("r")
109
110 def useD():
111     keyboard.press("d")
112     sleep(0.01)
113     keyboard.release("d")
114
115 def useF():
116     keyboard.press("f")
117     sleep(0.01)
118     keyboard.release("f")
119
120 wincap = WindowCapture()
121 screenshot = wincap.get_screenshot()
122
123 vision_enemyMinion = Vision('enemyMinionHP.png')
124 vision_enemyChampion = Vision('enemyHP.png')
125 vision_enemyTurret = Vision('enemyTurretHP.png')
126 vision_enemyTurretPlate = Vision('enemyTurretHPplate.png')
127 vision_allyMinion = Vision('alliedMinionHP.png')
128 vision_ultimate = Vision('ultimateAbility.png')
129 vision_allyChampion = Vision('alliedChampion.png')
130 sleep(2)
131
132 ATTACKING_CHAMPION = 0
133 ATTACKING_TURRET = 0
134 FARMING = 0
135 GOING_BACK = 0
136 MOVING_FORWARD = 0
137 RETREATING = 0
138
139
140 backTimer = time.time()
141 gameTimer = time.time()
142 doransSold = False
143
144 keyboard.press("y")
145 sleep(0.01)
146 keyboard.release("y")
147 sleep(10)
148 buyStartingItem()
149 rightClick(1885, 842)
150 sleep(12)
151 keyboard.press("s")
152 sleep(0.01)
153 keyboard.release("s")
154 sleep(45)
155
156 while(not keyboard.is_pressed('l')):
```

```
    lvlUp()

    screenshot = wincap.get_screenshot()
    pixelLowHP = pyautogui.screenshot().getpixel((841, 1053))

    enemy_turret = vision_enemyTurret.find(screenshot, 0.9)
    enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
    enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
    ally_minions = vision_allyMinion.find(screenshot, 0.95)
    enemy_turretPlate = vision_enemyTurretPlate.find(screenshot, 0.9)

    if pixelLowHP == (1, 13, 7) or time.time() - backTimer >= 300:
        dead = False
        rightClick(1666, 1056)
        useW()
        if pixelLowHP == (1, 13, 7):
            useF()
        while len(enemy_champions) or len(enemy_minions):
            rightClick(1666, 1056)
            screenshot = wincap.get_screenshot()
            enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
            enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
            pixelDead = pyautogui.screenshot().getpixel((711, 1032))
            if pixelDead == (185,185,185):
                dead = True
            sleep(0.5)
        if dead == False:
            keyboard.press("b")
            sleep(0.01)
            keyboard.release("b")
            sleep(13)
            backTimer = time.time()
            GOING_BACK += 1
        else:
            while pixelDead == (185,185,185):
                pixelDead = pyautogui.screenshot().getpixel((711, 1032))
            backTimer = time.time()
        continue

    if len(enemy_turret) or len(enemy_turretPlate):
        if len(ally_minions) < 3:
            rightClick(1666, 1056)
            sleep(0.3)
            RETREATING += 1
            continue
        if not len(enemy_minions) and not len(enemy_champions):
            screenshot = wincap.get_screenshot()
            enemy_turret = vision_enemyTurret.find(screenshot, 0.95)
            enemy_turretPlate = vision_enemyTurretPlate.find(screenshot, 0.95)
            if len(enemy_turret):
                rightClick(enemy_turret[len(enemy_turret)-1][0]+75,enemy_turret[len(
                 enemy_turret)-1][1]+150)

```

```
209             if len(enemy_turretPlate):
210                 rightClick(enemy_turretPlate[len(enemy_turretPlate)-1][0]+75,
                     enemy_turretPlate[len(enemy_turretPlate)-1][1]+150)
211             ATTACKING_TURRET += 1
212             continue
213         if len(enemy_minions) and not len(enemy_champions):
214             rightClick(ally_minions[len(ally_minions)-1][0]+30,ally_minions[len(
                 ally_minions)-1][1]+60)
215             sleep(0.1)
216             screenshot = wincap.get_screenshot()
217             enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
218             if len(enemy_minions):
219                 rightClick(enemy_minions[len(enemy_minions)-1][0]+30,enemy_minions[
                     len(enemy_minions)-1][1]+60)
220             FARMING += 1
221             continue
222         if len(enemy_minions) and len(enemy_champions):
223             rightClick(1666, 1056)
224             sleep(0.3)
225             RETREATING += 1
226             continue
227     else:
228         if not len(enemy_minions) and not len(enemy_champions):
229             if pyautogui.locateOnScreen('shop.png', confidence=0.95):
230                 if doransSold == False and gameTimer > 1200:
231                     doransSold = True
232                     rightClick(1090,1010)
233                 buyItem()
234                 sleep(1)
235             rightClick(1885, 842)
236             MOVING_FORWARD += 1
237             continue
238         if not len(enemy_minions) and len(enemy_champions):
239             screenshot = wincap.get_screenshot()
240             enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
241             if len(enemy_champions):
242                 keyboard.press("2")
243                 keyboard.release("2")
244                 win32api.SetCursorPos((enemy_champions[len(enemy_champions)
                     -1][0]+20,enemy_champions[len(enemy_champions)-1][1]+150))
245                 useW()
246                 useE()
247                 leftClickAA(enemy_champions[len(enemy_champions)-1][0]+40,
                     enemy_champions[len(enemy_champions)-1][1]+120)
248                 sleep(1)
249             keyboard.press("s")
250             sleep(0.01)
251             keyboard.release("s")
252             screenshot = wincap.get_screenshot()
253
254             enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
255             if len(enemy_champions):
256                 leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
```

```
                enemy_champions[len(enemy_champions)-1][1]+80)
257             useQ()
258             useD()
259
260             leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
                 enemy_champions[len(enemy_champions)-1][1]+80)
261             sleep(1)
262         screenshot = wincap.get_screenshot()
263         enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
264         ultimate_available = vision_ultimate.find(screenshot, 0.98)
265         if len(enemy_champions):
266             leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
                 enemy_champions[len(enemy_champions)-1][1]+80)
267             if len(ultimate_available):
268                 useR()
269                 sleep(3)
270         ATTACKING_CHAMPION += 1
271         continue
272     ally_champions = vision_allyChampion.find(screenshot, 0.95)
273     if time.time() - gameTimer <= 1200:
274         if len(ally_minions):
275             if len(enemy_minions) and not len(enemy_champions):
276                 rightClick(ally_minions[len(ally_minions)-1][0],ally_minions[len
                     (ally_minions)-1][1]+80)
277                 sleep(0.1)
278                 screenshot = wincap.get_screenshot()
279                 enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
280                 if len(enemy_minions):
281                     rightClick(enemy_minions[len(enemy_minions)-1][0]+30,
                         enemy_minions[len(enemy_minions)-1][1]+60)
282                 FARMING += 1
283                 continue
284             if len(enemy_minions) and len(enemy_champions):
285                 if len(enemy_minions) > len(ally_minions)+3:
286                     rightClick(1666, 1056)
287                     sleep(0.2)
288                     RETREATING += 1
289                     continue
290                 elif len(ally_minions) < len(enemy_minions)+1:
291                     rightClick(ally_minions[len(ally_minions)-1][0],ally_minions
                         [len(ally_minions)-1][1]+80)
292                     sleep(0.1)
293                     screenshot = wincap.get_screenshot()
294                     enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
295                     if len(enemy_minions):
296                         rightClick(enemy_minions[len(enemy_minions)-1][0]+30,
                             enemy_minions[len(enemy_minions)-1][1]+60)
297                         useQ()
298                     FARMING += 1
299                     continue
300                 else:
301                     rightClick(ally_minions[0][0]+30,ally_minions[0][1]+60)
302                     sleep(0.2)
```

```
303                         if len(enemy_champions):
304                             win32api.SetCursorPos((enemy_champions[len(
                                 enemy_champions)-1][0]+20,enemy_champions[len(
                                 enemy_champions)-1][1]+150))
305                             useE()
306                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+40,enemy_champions[len(enemy_champions)
                                 -1][1]+120)
307                             useW()
308                             sleep(1)
309                         keyboard.press("s")
310                         sleep(0.01)
311                         keyboard.release("s")
312                         screenshot = wincap.get_screenshot()
313 
314                         enemy_champions = vision_enemyChampion.find(screenshot,
                             0.95)
315                         if len(enemy_champions):
316                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
317                             useQ()
318                             useD()
319                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
320                             sleep(1)
321                         screenshot = wincap.get_screenshot()
322                         enemy_champions = vision_enemyChampion.find(screenshot,
                             0.95)
323                         ultimate_available = vision_ultimate.find(screenshot, 0.98)
324                         if len(enemy_champions):
325                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
326                             if len(ultimate_available):
327                                 useR()
328                                 sleep(3)
329                         ATTACKING_CHAMPION += 1
330                         continue
331             else:
332                 if len(ally_champions)>=2:
333                     if len(enemy_champions):
334                         win32api.SetCursorPos((enemy_champions[len(enemy_champions)
                             -1][0]+20,enemy_champions[len(enemy_champions)-1][1]+150))
335                         useE()
336                         leftClickAA(enemy_champions[len(enemy_champions)-1][0]+40,
                             enemy_champions[len(enemy_champions)-1][1]+120)
337                         useW()
338                         sleep(1)
339                         keyboard.press("s")
340                         sleep(0.01)
341                         keyboard.release("s")
```

```
342                         screenshot = wincap.get_screenshot()
343                         enemy_champions = vision_enemyChampion.find(screenshot,
                             0.95)
344                         if len(enemy_champions):
345                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
346                             useQ()
347                             useD()
348                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
349                             sleep(1)
350                         screenshot = wincap.get_screenshot()
351                         enemy_champions = vision_enemyChampion.find(screenshot,
                             0.95)
352                         ultimate_available = vision_ultimate.find(screenshot, 0.98)
353                         if len(enemy_champions):
354                             leftClickAA(enemy_champions[len(enemy_champions)
                                 -1][0]+50,enemy_champions[len(enemy_champions)
                                 -1][1]+80)
355                             if len(ultimate_available):
356                                 useR()
357                                 sleep(3)
358                         ATTACKING_CHAMPION += 1
359                         continue
360                     if len(enemy_minions):
361                         screenshot = wincap.get_screenshot()
362                         enemy_minions = vision_enemyMinion.find(screenshot, 0.95)
363                         if len(enemy_minions):
364                             rightClick(enemy_minions[len(enemy_minions)-1][0]+30,
                                 enemy_minions[len(enemy_minions)-1][1]+60)
365                         FARMING += 1
366                         continue
367                     rightClick(1885, 842)
368                     MOVING_FORWARD += 1
369                     continue
370                 else:
371                     rightClick(1666, 1056)
372                     sleep(0.3)
373                     RETREATING += 1
374                     continue
375             else:
376                 if len(enemy_champions):
377                     win32api.SetCursorPos((enemy_champions[len(enemy_champions)
                         -1][0]+20,enemy_champions[len(enemy_champions)-1][1]+150))
378                     useE()
379                     leftClickAA(enemy_champions[len(enemy_champions)-1][0]+40,
                         enemy_champions[len(enemy_champions)-1][1]+120)
380                     useW()
381                     sleep(1)
382                     keyboard.press("s")
383                     sleep(0.01)
```

```
384                     keyboard.release("s")
385                     screenshot = wincap.get_screenshot()
386                     enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
387                     if len(enemy_champions):
388                         leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
                             enemy_champions[len(enemy_champions)-1][1]+80)
389                         useQ()
390                         useD()
391                         leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
                             enemy_champions[len(enemy_champions)-1][1]+80)
392                         sleep(1)
393                     screenshot = wincap.get_screenshot()
394                     enemy_champions = vision_enemyChampion.find(screenshot, 0.95)
395                     ultimate_available = vision_ultimate.find(screenshot, 0.98)
396                     if len(enemy_champions):
397                         leftClickAA(enemy_champions[len(enemy_champions)-1][0]+50,
                             enemy_champions[len(enemy_champions)-1][1]+80)
398                         if len(ultimate_available):
399                             useR()
400                             sleep(3)
401                     ATTACKING_CHAMPION += 1
402                     continue
403                 if len(enemy_minions):
404                     rightClick(enemy_minions[len(enemy_minions)-1][0]+30,enemy_minions[
                         len(enemy_minions)-1][1]+60)
405                     useQ()
406                     FARMING += 1
407                     continue
408                 rightClick(1885, 842)
409                 MOVING_FORWARD += 1
410                 continue
411
412 print('Attacking enemy champions: ')
413 print(ATTACKING_CHAMPION)
414 print('Attacking enemy turrets: ')
415 print(ATTACKING_TURRET)
416 print('Farming: ')
417 print(FARMING)
418 print('Going back: ')
419 print(GOING_BACK)
420 print('Moving forward: ')
421 print(MOVING_FORWARD)
422 print('Retreating: ')
423 print(RETREATING)
```

2. Code of ’vision.py’

```
import cv2 as cv
import numpy as np


class Vision:
    needle_img = None
    needle_w = 0
    needle_h = 0
    method = None

    def __init__(self, needle_img_path, method=cv.TM_CCOEFF_NORMED):
        self.needle_img = cv.imread(needle_img_path, cv.IMREAD_UNCHANGED)

        self.needle_w = self.needle_img.shape[1]
        self.needle_h = self.needle_img.shape[0]

        self.method = method

    def find(self, haystack_img, threshold=0.5):
        result = cv.matchTemplate(haystack_img, self.needle_img, self.method)

        locations = np.where(result >= threshold)
        locations = list(zip(*locations[::-1]))

        rectangles = []
        for loc in locations:
            rect = [int(loc[0]), int(loc[1]), self.needle_w, self.needle_h]
            rectangles.append(rect)
            rectangles.append(rect)
        rectangles, weights = cv.groupRectangles(rectangles, groupThreshold=1, eps
         =0.5)

        points = []
        if len(rectangles):
            for (x, y, w, h) in rectangles:
                center_x = x + int(w/2)
                center_y = y + int(h/2)
                points.append((center_x, center_y))

        return points
```

3. Code of ’windowcapture.py’

```
import numpy as np
import win32gui, win32ui, win32con


class WindowCapture:

    w = 0
    h = 0
    hwnd = None
    cropped_x = 0
    cropped_y = 0
    offset_x = 0
    offset_y = 0

    def __init__(self, window_name=None):

        if window_name is None:
            self.hwnd = win32gui.GetDesktopWindow()
        else:
            self.hwnd = win32gui.FindWindow(None, window_name)
            if not self.hwnd:
                raise Exception('Window not found: {}'.format(window_name))

        window_rect = win32gui.GetWindowRect(self.hwnd)
        self.w = window_rect[2] - window_rect[0]
        self.h = window_rect[3] - window_rect[1]

        border_pixels = 0
        titlebar_pixels = 0
        self.w = self.w - (border_pixels * 2)
        self.h = self.h - titlebar_pixels - border_pixels
        self.cropped_x = border_pixels
        self.cropped_y = titlebar_pixels
        self.offset_x = window_rect[0] + self.cropped_x
        self.offset_y = window_rect[1] + self.cropped_y

    def get_screenshot(self):
        wDC = win32gui.GetWindowDC(self.hwnd)
        dcObj = win32ui.CreateDCFromHandle(wDC)
        cDC = dcObj.CreateCompatibleDC()
        dataBitMap = win32ui.CreateBitmap()
        dataBitMap.CreateCompatibleBitmap(dcObj, self.w, self.h)
        cDC.SelectObject(dataBitMap)
        cDC.BitBlt((0, 0), (self.w, self.h), dcObj, (self.cropped_x, self.cropped_y)
         , win32con.SRCCOPY)

        signedIntsArray = dataBitMap.GetBitmapBits(True)
        img = np.fromstring(signedIntsArray, dtype='uint8')
        img.shape = (self.h, self.w, 4)

```

```
        dcObj.DeleteDC()
        cDC.DeleteDC()
        win32gui.ReleaseDC(self.hwnd, wDC)
        win32gui.DeleteObject(dataBitMap.GetHandle())

        img = img[...,:3]
        img = np.ascontiguousarray(img)

        return img

    @staticmethod
    def list_window_names():
        def winEnumHandler(hwnd, ctx):
            if win32gui.IsWindowVisible(hwnd):
                print(hex(hwnd), win32gui.GetWindowText(hwnd))
        win32gui.EnumWindows(winEnumHandler, None)

    def get_screen_position(self, pos):
        return (pos[0] + self.offset_x, pos[1] + self.offset_y)
```