Imlementation of an A.I. Player for the videogame League of Legends based on Image Recognition, Discretization and Gradient Optimization

Oliver Struckmeier
Eric Heung
School of Electrical Engineering
Aalto University

October 25, 2017

Abstract

The topic of this paper is to create an agent that is able to play 3rd person massive multiplayer online battle arena games (MOBA) like League of Legends, Dota 2 and Heroes of the Storm with the same input as a human player. Image recognition is used to detect objects in the game environment. The environment is then discretized by dividing the screen content in a grid. The grid contains information for the agent like rewards and obstacles. With these informations an implementation of a gradient optimization algorithm it can be determined which area of the screen has to be clicked or how the agent has to position in order to get a maximum reward that leads to winning the game.

1 Introduction

This report discusses the results of creating an agent that can execute simple tasks in a MOBA game (multiplayer online battle arena game). The game chosen is League of Legends due to its, relative to similar games, lower focus

on micro gameplay and fewer macro tactical decisions. It is inspired by the Dota 2 Bot created by OpenAi [1]. The first objective will be to determine the state and action space as well as the environment in which the agent is positioned. This is achieved with image recognition using OpenCV and TensorFlow is used to detect objects in the game. It would be possible to read game data like positions from the RAM of the computer but this action is considered an illegal action in the game as it provides a significant advantage to a human player. At first it will be presented how the image acquisition and recognition works. Then it will be discussed how we can create a representation of the game using discretization. The discretized environment allows the agent to use learning methods and decision making processes on the environment in order to learn behaviour policies to achieve certain goals by following a policy of with a maximum reward for each action.

2 An Introduction and League of Legends

First, some basics and certain terms that will be later used to describe actions, states and the game process have to be introduced. Also a target for this project will be declared to see if the method was succesfull.

Figure 1: The game level

2.1 League of Legends

is an online multiplayer game created by Riot Games in 2009. It is usually played between two teams of human player consisting of 5 members on each side. Each player can select one of more than 120 player characters, so called "Champions".

The game takes place in a square shaped static environment or level. Each team has it's own base either at the bottom left or top right corner of this map. In the base is the main structure called the Nexus. Losing this structure means losing the game. The two bases are connected by three "Lanes" originating of each base. When the game starts so called "Minions", computer controlled soldiers, swarm from one team's Nexus to the enemy

team's nexus and attack every enemy minion, character or structure they meet on the way. On each Lane are several towers which are strong deffensive structures and can't be attacked by a single player since because of their high damage output. It is the minions purpose to serve as a distraction for the player to take down these towers. Without a human's interaction to the game the waves of Minions would collide midways and attack each other but never make any significant progress due to the fact both sides are perfectly balanced and towers provide deffensive capabilites and can only be taken down by clever actions through uman players. It is the players task to influence this equilibrium in a way that the enemy base will be conqured and the enemy nexus destroyed. Figure 1 shows a zoomed out overview of the map with the red lines displaying lanes and the Nexus in a red rectangle.

2.2 Evaluation of the environment

This static environment provides a static enough scenario with obstacles and a finite space to learn how to navigate through it while still challenging the agent. The agent can interact with the environment by for example attack minions and thereby changing the balance of the game to his favor. Of course there are many more challenges to the game, especially considering the presence of other players who also want to win the game. For now we focus only on one player, the agent, trying to to win a game without enemies by influencing the environment. Furthermore macro decision making is ignored and the agent is trained to only detect his player character enemy minions and enemy towers to see if the system can be implemented to achieve satisfying results. Once this is the case other game elements can be added step by step.

2.3 Defining a Goal

The goal is to create a basic agent for the videogame League of Legends and teaching the agent to kill enemy minions in order to push through one lane and kill the enemy nexus within a reasonable amount of in-game time. This involves moving in a part of the game environment, determining where enemies and allies are and assessing their threat level or reward toward winning the game.

A next step would be to make the object detection more powerful by teaching it more objects like enemy champions, allied minions and structures. The precision of the object detection model is crucial for the precision of the

decision making and thus expanding its capabilites will also improve the overal performance.

Another challenge is to make the agent aware of the overal state of the game outside of its limited view into the game environment. For example the agent has to know where the enemy base is and how to get there and react to events that happen outside of its field of view somewhere else on the map.

A long term goal could be to introduce the official game A.I. to the game and see if the agent can beat it in a 1 versus 1 game. But for this the object detection has to be improved and the game awareness has to be expanded over the whole map.

3 Agent perception and Environment Representation

3.1 Environment Detection

As mentioned in the introduction the agent uses OpenCV to capture the game view in real-time and TensorFlow to perform image recognition on the data. For this purpose the TensorFlow Object detection API was used [2]. Since League Of Legends is not a realistic game the predefined models for object detection did not work. For this reason a custom model was trained to detect the player character as well as minions and towers.

Figure 2: The Player Character "Vayne" as seen by TensorFlow

As Figure 2 shows, the object detection of the player character works satisfying and the agent still provides roughly 1 update per second which is barely enough to interact with the game.

Performance The performance of this system is heavily depending on the image recognition as this is the most performance heavy task for the system apart form running the game itself. The python loop that performs the image detection runs about 1 time per second which is still a satisfying number of actions we can execute per second. This calculation time gets worse the more aditional calculations we execute and the more complex the decision making and learning algorithms are. The performance could be of course improved by using a computer with stronger hardware or distributing the object detection process to other hardware.

3.2 State Representation of the Environment using Discretization

The environment in which the player character is placed is limited and limited by the screen and the size of a single pixel on it. It is difficult to define a state pace in this environment because the small units of pixels make it a continuous space. A solution for this problem is to discretize the environment. The game does not allow clicks with single pixel precision and the player character is about 40 by 60 pixels, so it is sufficent to divide the screen into a grid of possible positions for the player character.

Figure 3: The Player Character in a discrete grid

In Figure 3 the red dot symbolizes the position after processing the box (marked in cyan) around the player figure. Based on the position of the player a grid with sizes based on the detected player character is layed over the whole screen, each representing one state and the player being the origin of the grid. A state S in the grid is given by: $S_{x,y}$

If we now introduce other game objects like minions into this environment we can determine in which state on the screen the minion is by matching its detected position into the grid. The same works for obstacles which make a state unusable for the agent.

Figure 4: A sample game situation from the perspective of the agent

Figure 5: The corresponding game view

Figure 7 shows a sample situation from the game with marked champions, tower and enemy minions. Their positions are marked as red dots and the states they occupy are framed by blue rectangles for the enemy minions and white for the enemy tower. The player state which is also the origin (0|0) of the state space is marked with a red rectangle.

3.3 Action Space for the Agent

The action space for the agent makes use of the grid generated by the discretization. As the grids are divided in small enough units, each state can be directly translated into a position that can be clicked by the agent. An action A is defined as a click on a certain state: $A(x, y) = mouse\_click(S_{x,y})$

To increase the effectivity of the clikcing process we make use of one of the games features called attack move and normal attack.

If a normal click hits terrain it makes the player character move to this position. If the normal click is executed on an enemy the character stays in place and attacks the target if it is in range or moves towards it and attacks the target when it gets in range. This mode is used for moving the player character around in the game environment. An attack move on the other hand is similar to the normal click but if applied to a position without an enemy it automatically attacks the closest target to the click position. Thereby we can eliminate the lack of precision when an enemy minion just barely is in a certain state but we try to attack it by clicking in the middle of the state and would miss.

4 Evaluating a Policy

//TODO Implement Monte Carlo and write more

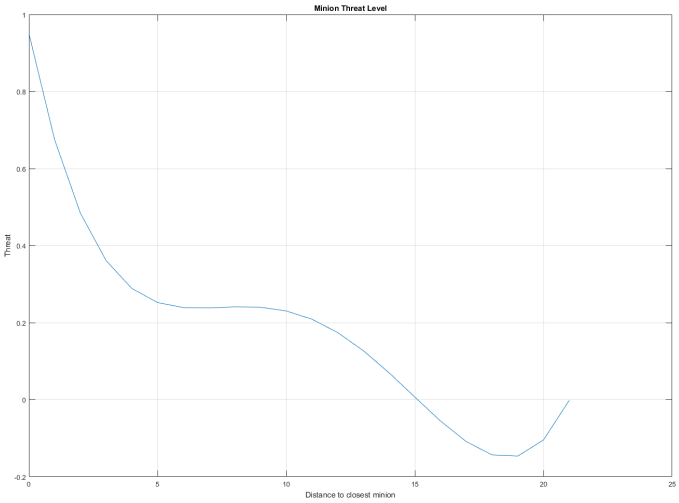

Figure 6: Threat level of a minion

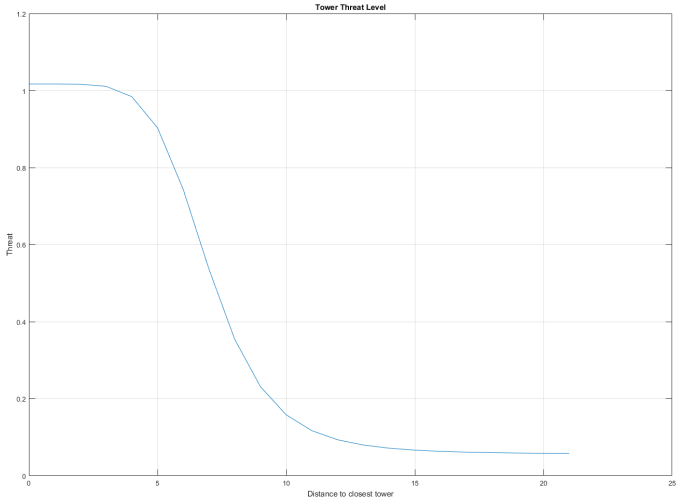

Figure 7: Threat level of a tower

5 Rewards, Feedback and learning

The rewards are used to improve the agents decision making. Since we have to make decisions all the time and we have no time to use Q-learning or Deep Learning we have to use the Temporal Difference Method to learn while playing the game. For this to work we calculate a reward with the following parameters:

- Player Hit Points
- The number of attacks executed
- The time the agent survived without dieing

This means the reward is higher the less hit points the agent loses while maximising the number of attacks and surviving as long as possible. In order to get a more reliable reward value we calculated one reward averaged over 1 second and one averaged over 5 seconds. This allows us to also take actions into account that are not directly punished, for example moving to a position over the course of a few seconds and then being in a bad spot.

6 Conclusions

The conclusion compares the outcomes to the more advanced Dota2 Bot made by OpenAi[1].

6.1 Problems

//good results it can do what was the goal. now it is to be improved

6.1.1 Object Detection

6.1.2 Performance

6.1.3 Learning

The problem with learnign was that it is not possible for us to let the agent practice a lot because we have no possibility to speed up the game process. The Dota2 Bot by OpenAI cooperated with the producers of the game and had an interface to the game which allowed it to play millions of games

against itself and learn from human players replays. Since we do not have these means the learning capabilities of our agent is quite limited and has to rely on learning by doing.

6.2 Future Improvements

References

[1] OpenAi Dota 2 bot,
https://blog.openai.com/dota-2/

[2] TensorFlow Object Detection API,
https://github.com/tensorflow/models/tree/master/research/object_detection