

Implementation of an A.I. Player for the videogame League of Legends based on Image Recognition, Discretization and Monte Carlo Policy Evaluation

Oliver Struckmeier
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 the Monte-Carlo Policy Evaluation is used to determine 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 possibility 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 defensive 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 defensive capabilities and can only be taken down by clever actions through human players. It is the players task to influence this equilibrium in a way that the enemy base will be conquered 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 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 and minions 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

This is a first attempt in creating an agent for this videogame and thus the goal for this demonstration consists of teaching the agent to kill enemy minions in order to push through one lane and kill the enemy nexus within 1 hour of game time.

A next step would be to detect enemy structures and learn how to efficiently take them down.

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.

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 //TODO source. 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 3 updates per second which are enough

to interact with the game for now.

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 from running the game itself. The python loop that performs the image detection runs about 3 times per second which is still a satisfying number of actions we can execute per second. 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 space 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 sufficient 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.

Rewards can be determined by checking for each state if it contains a minion and then setting the reward to a certain value. A challenge will be the implementation of towers, as these provide a higher reward if they get destroyed, but also pose a serious threat to the player.

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: $A(x, y) = \text{mouse_click}(S_{x,y})$

On each click it is also checked if the state is usable or not. Obstacles like the minions of a players team or terrain are not clickable.

4 Evaluating a Policy

Implementation of the Monte-Carlo Policy Evaluation to determine the best position for the character to be in.

//TODO Implement Monte Carlo

4.1 Decide between moving and Attacking

If the player character is in range for an attack on a minion offering a reward it has to be determined by the policy if it is better to reposition once more or to attack the minion.

//TODO split actions into move and attack

5 Conclusions

//TODO

References

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