Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-20)

Deep Learning Bot for League of Legends

Aishwarya Lohokare,* **Aayush Shah,*** **Michael Zyda**
University of Southern California
Los Angeles, California
{lohokare, aayushsh, zyda}@usc.edu

Abstract

In this paper, we take the first step towards building comprehensive bots capable of playing a simplified version of League of Legends, a popular 5v5 online multiplayer game. We implement two different agents, one using Reinforcement Learning and the other via Supervised Long Short Term Memory Network. League of Legends provides a partially observable game environment with an action space much larger than games like Chess or Go. Source code and demonstrations can be found at https://github.com/csci-599-applied-ml-for-games/league-of-legends-bot.

Introduction

League of Legends (LOL) is a Multiplayer Online Battle Arena (MOBA) (Silva and Chaimowicz 2017) game developed by Riot Games in 2009. On average each LOL game is 35 minutes long. For the 5v5 mode, each player selects one unique champion out of 148 possible champions. Each champion has set of unique offensive and defensive abilities. Victory in LOL is achieved by destroying the opponents' Nexus which is essentially the core of their base.

Research has been limited in LOL due to a lack of official API support to access underlying game information as opposed to DOTA2 which has an existing API to access this information. In 2015, one LOL work (Silva and Chaimowicz 2017) used a third party tool known as Bot of Legends, to capture underlying game information. However, Bot of Legends also modified game files, which led to them being shut down in 2017 after losing a $10 million lawsuit against Riot Games.

Due to this API limitation in LOL, we decided to simplify the game-space by working on 1v1 game-play in MidLane only. We use Ashe as our champion and allow only one skill move, Volley, which is her best offensive skill move. We do not purchase any items for our champion. We have set a sub-objective which involves first blood or first kill as the terminating condition rather than going all the way until the Nexus is destroyed. Veigar is the champion used by the enemy for both approaches. For reward shaping we used the research done in (Zhang et al. 2019). This work describes a method to model micro decisions in King of Glory as a Reinforcement Learning problem with a simple reward. King of Glory is a MOBA game similar to League of Legends. We were able to use this as a reference to formulate rewards for our League of Legends bot.

One notable related work in MOBA involves OpenAI Five (OpenAI et al. 2019). OpenAI Five was the first AI system to defeat professional world champions in Dota 2. OpenAI Five used a Proximal Policy Optimization (PPO) (Schulman et al. 2017) + Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) based approach which was continually trained in a distributed framework consisting of 128,000 CPU cores and 256 P100 GPUs on GCP for a period of 10 months to achieve this superhuman performance. Another notable MOBA AI contribution involves Tencent AI's bot for Honor of Kings (Ye et al. 2020). The AI was trained using a combination of Actor-Critic Network (Espeholt et al. 2018), LSTM and PPO using 600,000 CPU cores and 1,064 Nvidia GPUs (mixture of Tesla P40 and V100). The trained AI could defeat professional players in a 1v1 setting.

Proposed Methodology

We have trained a CNN based model to perform object detection. Our object detection API was motivated by the work done in (Struckmeier 2019) which involved generating a synthetic dataset for object detection of 5 classes in League of Legends. Our model is implemented using the YOLOv3 (Redmon and Farhadi 2018) architecture. The model is trained until convergence on 1200 manually annotated frames from League of Legends consisting of 15 different classes. The different classes include 2 champions (Ashe and Veigar), minion, super-minion, canon-minion, turret, inhibitor, and nexus for red and blue teams. It also outputs the mouse location from the screen. Figure 1 shows YOLOv3 output for one frame. We use this data as observations for our algorithms. The actions (keyboard and mouse clicks) predicted by our algorithms are simulated in the game using Python scripts.

*These authors contributed equally to this work

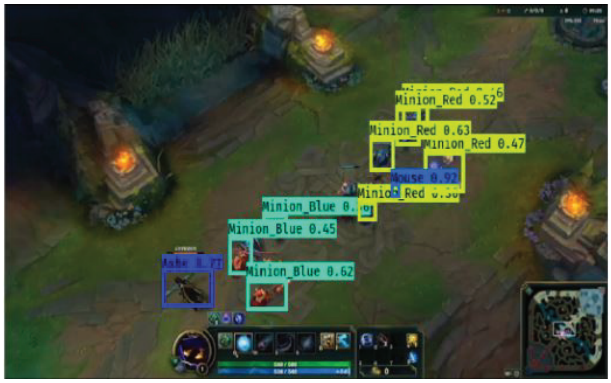

Figure 1: Bounding boxes from YOLOv3

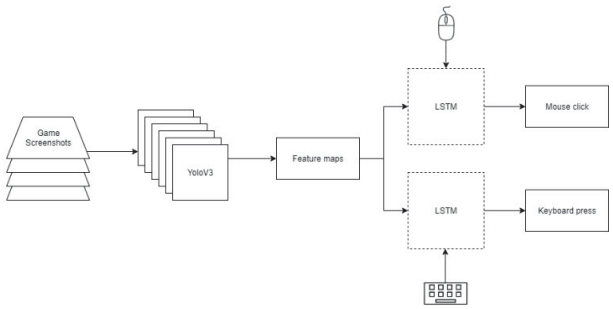

Figure 2: LSTM Architecture

LSTM

Our first approach involves the use of a Long-short term memory (LSTM) model. LSTMs are a specific set of Recurrent Neutral Networks that are proficient in learning long-term dependencies. In this method, we used neural networks to determine the game-playing patterns of the human players which would then teach the bot to play the game. Figure 2 shows a high level overview of this architecture. Building the bot included three major stages for the neural network.

The first stage was Data Collection where we developed a script to capture the game screen, the keyboard presses, mouse movements, and clicks of a human player playing the game. The captured game screen images were then passed through the YOLOv3 object detection network to extract the salient feature vectors from the images. The keyboard presses were mapped to a binary value and the mouse clicks were captured in the form of a tuple of x and y screen coordinates. For every frame, this triple was wrapped to form an input to our LSTM architecture. A total of 50 games were played by two advanced League of Legends players to form the dataset. This amounted to approximately 70,000 frames of images annotated with keyboard and mouse presses.

The second stage was the Training phase, in which these champion movements (mouse movement/clicks) and skill selection (keyboard press) were trained parallelly on two separate neural networks because both were independent of each other. One LSTM network was fed with the image feature vectors and mouse data to learn champion movement, while the other was fed with feature vectors and keyboard data to predict whether a skill move must be used.

In the final stage, to make the bot play the game, we wrote a script to simulate the mouse and keyboard presses based on the output of our LSTM. This script used weights learned in the training phase to predict what action to take based on the current game screen.

PPO + LSTM

The second approach involves a combination of Proximal Policy Optimization (PPO) and Long-short term memory (LSTM) models for our bot. PPO is a policy gradient based algorithm that has achieved state of art performance on several Atari-2600 games. It involves clipping policy updates at each iteration to minimize bad decisions due to large differences between an old policy and a new policy.

We combine PPO with LSTM as per the OpenAI Five model for Dota 2 (OpenAI et al. 2019). The reward is calculated based on the distance from our champion's location to that of the enemy champion, turrets, and minion's. Our champion's health points which are read via the Optical Character Recognition using Tesseract OCR (Smith 2007), and enemy champion's health points also contribute to this calculation. The LSTM observes game state using feature vectors from our YOLOv3 model and outputs an action which involves using a skill move, attack enemy champion or minions, or fleeing from the enemy.

Our agent learns from playing against an inbuilt League of Legends bot in a custom 1v1 game mode. Our model plays against this bot repeatedly to update the policy. An iteration (episode) ends when either our champion dies, or when it kills the enemy champion. The model was trained for 72 hours on a Google Cloud Instance with one Tesla K80 GPU. We created a custom gym environment to facilitate this training.

Results

With limited training, both LSTM, and LSTM+PPO agents[1] picked up on basic rules of the game - attack minions to gain minion kills, attack champion to gain kills, stay clear of turrets and run away from a conflict in the event of low health. Table 1 illustrates a comparative performance of both approaches. Both agents were successfully able to achieve first blood against amateur players. The PPO+LSTM bot outperformed the LSTM bot in achieving first blood against the enemy champion.

The LSTM bot had a drawback of moving back and forth randomly in the presence of an enemy. This was due to poor interpretation of patterns in training data when a player moves away to dodge attacks from an enemy. Another limitation of this approach was the lack of training data. The performance of the bot should significantly improve by training on more hours of game play. The LSTM+PPO bot was more stable and did not have any sudden erratic movements. It was performing significantly better than a random agent playing in our limited environment. With many more hours of train-

[1]Watch bot in action at https://tinyurl.com/yb54ljx4

	Minions Attacked	Success Rate
LSTM	4	0.55
PPO + LSTM	7	0.70

Table 1: Average comparative performance of our agents against amateur players for over 20 games until first blood

ing on self-play with a few additions to action-space, we believe it can get very close to human level performance in our customized game.

Conclusion and Future Scope

The next steps in our development are completing the 1v1 game mode by including all abilities and items. To add abilities we need to expand our environment's action space. Adding these to our action space and retraining requires significant computing power which we do not have access to currently. To add items, we would require a secondary system to handle learning what items to buy at a given point of time for each champion in the game. We would also like to experiment with self-play methodology for training our Reinforcement Learning bot, and change the action space from discrete to continuous for the Reinforcement Learning bot. The supervised LSTM bot may be able to perform better with additional annotated data captured from professional League of Legends players.

In order to create an entire team of bots for 5v5 game play we would need to expand our environment and train different models for each champion. A communication channel will be required for multi-agent coordination. Since using feature vectors from YOLOv3 is not computationally fast on our machines, we want to experiment with some alternatives like MobileNet (Howard et al. 2017) for accessing underlying game information. Having access to high performance computers with powerful GPUs is required for learning good models for each champion.

References

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

OpenAI; Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Redmon, J., and Farhadi, A. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silva, V. d. N., and Chaimowicz, L. 2017. Moba: a new arena for game ai. *arXiv preprint arXiv:1705.10443*.

Smith, R. 2007. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, 629–633. IEEE.

Struckmeier, O. 2019. Leagueai: Improving object detector performance and flexibility through automatically generated training data and domain randomization. *arXiv preprint arXiv:1905.13546*.

Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; et al. 2020. Mastering complex control in moba games with deep reinforcement learning. In *AAAI*, 6672–6679.

Zhang, Z.; Li, H.; Zhang, L.; Zheng, T.; Zhang, T.; Hao, X.; Chen, X.; Chen, M.; Xiao, F.; and Zhou, W. 2019. Hierarchical reinforcement learning for multi-agent moba game. *arXiv preprint arXiv:1901.08004*.