Reinforcement Learning-Based Auto Chess Implementation

Kaya Daylor and Aleksei Glebov

*Introduction*

The goal of this project is to develop an Auto Chess agent using Reinforcement Learning (RL). Auto Chess is a strategy-based game that involves deploying units with different costs, abilities, and synergies, requiring the agent to make decisions on unit placement, itemization, and positioning. These decisions are based on game states which are constantly changing.

The problem we are working on is designing an RL model that can autonomously play Auto Chess while learning optimal strategies for the game. The challenge lies in the complex nature of the game, the large state and action space, and the need for the agent to adapt to its strategies based on both random and opponent-driven elements in the game. The RL agent must navigate these challenges without relying on predefined strategies but rather by learning directly from its interactions with the environment.

This project will explore the development of an RL agent capable of performing well in Auto Chess while addressing challenges like delayed rewards and decision making under uncertainty.

*Literature Review*

This project will be taking inspiration from several previous works and related articles. As in many RL models, and as in many similar problems (especially chess), we will be using Q-Function and expected outcomes to calculate the rewards for our agent. Past methods will be taken into consideration, however, mostly we will be trying our own ideas.

* “Simulating Team Fight Tactics” – Riot Games official article, <https://media.gdcvault.com/gdc2023/Slides/Simulating++Teamfight+Tactics_Cao_Ran.pdf>   
  This article is a treasure that highlights some challenges and how to deal with them. This one is especially useful because it is related directly to our project.
* “Large-scale deep learning to augment production RL workloads at Riot Games” – Anyscale YouTube channel, <https://www.youtube.com/watch?v=8EsQkFxWYhU>  
  Riot Games’ video that describes the production of their reinforcement learning bots in TFT.
* “TFTMuZeroAgent” – GitHub repository created by user silverlight6, <https://github.com/silverlight6/TFTMuZeroAgent>  
  Interesting representation of TFT environment, that we can use to take inspirations from.
* “Reinforcement Learning in Chess” – article on Medium made by Aditya in November 23, 2023,  
  <https://medium.com/@samgill1256/reinforcement-learning-in-chess-73d97fad96b3>  
  Useful article about representation of reinforcement learning in regular chess game. This is also a useful article, because of the similarity (although not a big one) between two games.

*Current Results*

So far we have made substantial progress in developing an Auto Chess environment. We have implemented a Unit class which has several attributes and methods for handling the different unit types, costs, and levels. A diverse roster of units has been created and is being tested. The Player class includes several essential functions for managing resources, the board state, and shop transactions. Lastly, we implemented the Shop class with probability-based unit selection for each player.

*Challenges Encountered*

We experienced several challenges that need to be addressed before the game becomes fully functional. While the basic mechanics exist, the fighting system is still incomplete with no concrete battle resolution logic between players. We also lack methods for unit positioning. There are also some minor bugs in the code that need to be fixed.

Additionally, we may not be able to implement the interactive/visual component as proposed initially. This would require significant additional development to create a user friendly interface beyond our current implementation. Although it would enhance player experience and audience engagement, this seems to be a technical challenge given our current progress on the essential game mechanics.

*Next Steps*

1. Finish the environment;
2. Implement the enemy bots (with hard-coded logic);
3. Implement our own RL Agent;
4. Increase the complexity of the environment;
5. Implement the emulator (potentially).

*References*