A Reinforcement Learning Environment for Checkers Game Optimization: Design and Implementation

FOZAME ENDEZOUMOU Armand Bryan
fozybryan@outlook.com

October 19, 2024

Abstract

In this paper, we present the design and implementation of a checkers game environment specifically adapted for reinforcement learning (RL). We discuss the state representation, action space, reward structure, and the integration of the OpenAI Gym framework for training AI agents. This work aims to contribute to the development of efficient training environments for strategic board games, enhancing the capabilities of AI models in complex decision-making tasks. Moreover, our goal is to advance research and develop neural network models that can become champions in this game.

1 Introduction

Reinforcement learning (RL) has demonstrated significant potential in training AI agents for complex tasks, particularly in strategic board games such as Chess, Go, and Checkers. This paper introduces a novel checkers game environment specifically designed for RL applications, utilizing the OpenAI Gym framework. Our objective is to create a comprehensive checkers environment, easily integrated into the Gym framework, which can be employed to train high-performing AI agents capable of excelling at this game.

1.1 Motivation

Checkers is a strategic game with a large decision space, making it a suitable challenge for RL algorithms. Unlike traditional hardcoded AI solutions, RL allows for dynamic learning based on interaction with the environment. This paper aims to bridge the gap between classical AI game agents and modern RL techniques by providing a flexible and customizable training environment.

2 Related Work

Several board games have been adapted for RL, with notable success in games like Chess and Go. However, limited research has been conducted on checkers, despite its historical significance in AI research. We review previous works on board game environments for RL, highlighting their strengths and limitations, and positioning our work within this context.

3 Chessboard

3.1 Engine Used

To successfully create this environment, the first step was to set up the checkers game board. For this purpose, we utilized the chessboard engine previously developed by `Everest Witman`, which is available on GitHub.

Given that there are various types of checkers boards and multiple rule sets, we based our implementation on international draughts. For more information about the game and its rules, please refer to the Wikipedia page: International Checkers.

However, as the engine did not fully meet our requirements, we made several modifications to align it with our objectives.

3.2 Modifications Made

We revised some of the mechanisms of the original chessboard engine. For instance, the mandatory capture rule was not implemented, so we integrated it into the environment. Additionally, we developed an interface to visualize various game statistics such as the number of rounds, the current player, and elapsed time.

Furthermore, we added more detailed information in a lower interface section, which displays data such as the probability of winning, rewards obtained, and other strategic indicators. These enhancements were necessary to facilitate a more comprehensive and interactive gaming experience, while also providing valuable feedback to the RL agents during training.

A preview of the checkers environment is shown below.

4 Gymnasium Environment Design for Checkers

We designed a checkers game environment using the OpenAI Gym engine to enable reinforcement learning agents to be trained. By integrating a custom game engine into the Gym environment, we can simulate checkers matches while collecting the necessary metrics to evaluate learning policies. Below, we detail the different components of the environment.

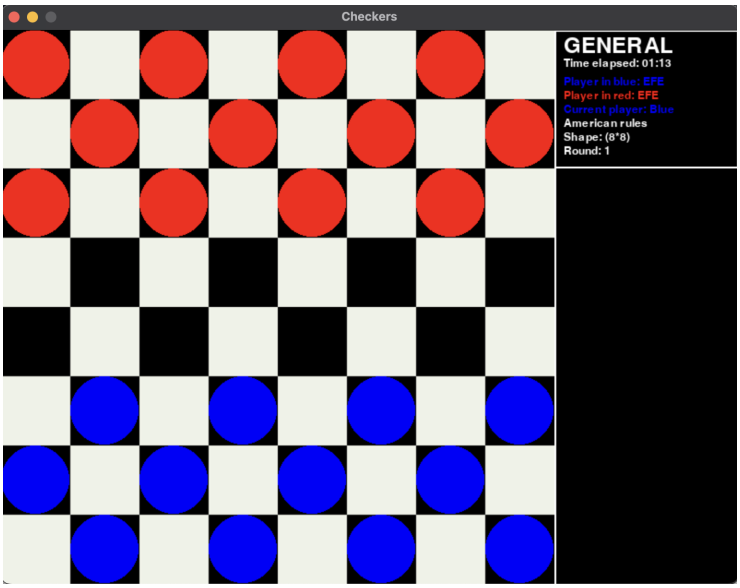

Figure 1: Checkers environment

4.1 State Representation

The checkers board is represented as an 8×8 matrix $\mathbf{B} \in \mathbb{Z}^{8\times 8}$, where each element corresponds to a cell on the board and its state. Each cell can contain:

$$B_{i,j} = \begin{cases} 0 & \text{if the cell is empty,} \\ 1 & \text{if a piece from player 1 (Red) occupies the cell,} \\ 2 & \text{if a piece from player 2 (Blue) occupies the cell,} \\ 3 & \text{if a king from player 1 occupies the cell,} \\ 4 & \text{if a king from player 2 occupies the cell.} \end{cases}$$

This matrix representation allows for a simple and direct visualization of the game's state and serves as input for reinforcement learning algorithms to make decisions.

4.2 Action Space

The action space in our environment is modeled as a discrete space of dimension $64 \times 64 = 4096$. Each action corresponds to a piece's movement from an initial position to a final position on the board. More formally, an action is a pair (s, e), where $s \in [0, 63]$ is the starting cell and $e \in [0, 63]$ is the destination cell, with indices based on matrix encoding.

Invalid actions, such as moves that violate the game's rules, are penalized.

4.3 Reward Structure

The reward system is designed to encourage strategic behavior from the agent. Rewards are assigned as follows:

- **Capture Reward**: The agent receives +10 for capturing an opponent's piece. This is modeled by the reward function $R_{\text{capture}} = 10 \cdot \mathbb{I}(\text{capture})$.
- **Illegal Move Penalty**: A penalty of -5 is applied for attempting an illegal move. This penalty can be expressed as $R_{\text{illegal}} = -5 \cdot \mathbb{I}(\text{illegal})$.
- **Mandatory Capture Penalty**: If the agent fails to capture when it is mandatory, a penalty of -5 is applied. $R_{\text{mandatory capture}} = -5 \cdot \mathbb{I}(\text{missed capture})$.
- **Victory Reward**: At the end of the game, a reward of +20 is given to the winner. The reward function is $R_{\text{win}} = 20 \cdot \mathbb{I}(\text{win})$.

4.4 Environment Integration Strategy

The environment has been designed to be compatible with popular RL libraries such as Stable Baselines3 and Ray RLlib, allowing for easy agent training and evaluation. It adheres to the Gym API, offering methods such as `step()`, `reset()`, and `render()`. The state is returned as a matrix, enabling direct integration with convolutional neural networks for feature extraction from the game board.

5 Experimental Results

We conducted a series of experiments to test our checkers game environment integrated within the Gym framework. A `logs` folder is provided to store metadata from the games. Initially, random actions were taken using Gym's random action feature, which was then controlled by our rule-enforcing system. For more information, the project's GitHub repository can be found here.

6 Conclusion

This paper presents a flexible and effective checkers environment for RL research, highlighting its potential for future work in strategic game AI. The environment can be used to develop advanced agents and can serve as a benchmark for evaluating new RL algorithms in the domain of board games.

7 Future Work

Future research could expand the complexity of the checkers environment by introducing variations in game rules, multi-agent settings, or incorporating human-AI competition. Additionally, improving the state representation and action space could lead to more efficient training of RL agents.

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[3] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.

[4] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3), 210-229.

[5] Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., ... & Sutphen, S. (2007). Checkers is solved. *Science*, 317(5844), 1518-1522.

[6] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... & Silver, D. (2020). Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839), 604-609.

[7] van der Heijden, L. (2019). *Checkers AI using Deep Reinforcement Learning*. Master's thesis, Utrecht University. Available at Utrecht University Repository.

[8] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.