# MultiAgent Reinforcement Learning for Competitive Strategy-Learning in Chess

Minha Rehman, Vaneeza Ahmed, and Rabail Nasir

Department of Computer Science, FAST NUCES
minharehman45@email.com

**Abstract.** This paper explores the application of Multi-Agent Reinforcement Learning (MARL) for competitive strategy learning in chess. We propose a novel agent interaction framework to simulate adversarial learning dynamics and evaluate its performance against traditional approaches.

## 1 Introduction

Chess has been a long time benchmark for the study of AI, because of the complex nature and depth of strategy of chess playing can provide good testing ground for RL methods [2,7]. Brute-force search processes have by tradition been used by conventional chess engines but recent state of the art with Deep Reinforcement Learning (DRL) indicate superior performance of the AI via self-play and policy optimization [1,**?**].

Our objective with this project is to exploit Multi-Agent Reinforcement Learning (MARL) to apply competitive strategy learning to chess. Different from single-agent RL algorithms, MARL uses multiple learning agents to train within the same environment; therefore, it fits adversarial games such as chess [6,9].

*Objectives.* The major goals of this project involve the following:

Apply a MARL framework adapted to competitive chess strategy learning [4]. ii) Train multiple agents through many techniques in reinforcement learning to produce strategic play styles.Compare agent performance with classical chess engines, and normal human players [8]. Discover cooperative strategies and decision making behaviors in learning agents [10].

*Paper Organization.* The rest of this paper is structured as follows. Related work in MARL and strategic learning are covered in Section 2. Section 3 describes the methodology; environment design; and algorithms used. Section 4 present experimental setup and results. Section 5 builds around the discussion, limitations, and the future directions. Finally, Section 6 concludes the paper.

## 2 Related Work

Additionally, Reinforcement Learning (RL) was repeatedly applied in complex decision-making tasks, in Go and Chess, and very successful [**?**,7]. For instance, Stockfish chess engines of the classical kind have been using bruteforce search and hand-designed rules, but that deep RL such as AlphaZero has provided evidence that self-play and policy refinement can preclude the classics [7].

In the field of Multi-Agent Reinforcement Learning (MARL), a number of research efforts have been produced which concentrate on learning the competitive or cooperative behavior of the agents through the interaction with the agents [**?**]. MARL brings the new complexities of non-stationarity and coordination that are not the cases in single agent settings [**?**].

Perolat et al [6] looked at MARL in the context of adversarial games, this proved that learning through self-play allows reaching robust policies. In the same way Vinyals et al. [9] illustrated how population-based training can attain human capacity in complex games in real time strategy.

Hierarchical MARL approaches, as studied by Yang et al. [10], enable agents to learn at various levels of abstraction which can be applicable for chess modeling of strategic planning. Omidshafiei et al. [5] have put forward deep decentralized policies for partial observability environments which are suitable for the opponent's modeling in the game chess.

Other relevant work is by Tessler et al. [8] where temporal abstraction and learning is investigated in the context of action-rich domains, and Yu et al. [11] who present generalized hierarchical Q-learning, possibly scalable alternatives to policy gradient methods.

In addition, existing frameworks like PettingZoo [**?**] and SuperSuit have supported standardized experimentation in MARL environments. Schulman et al. [**?**] proposed the Proximal Policy Optimization (PPO), which is still a well performing algorithm because of its simplicity and best empirical performance.

For strategic games long time reasonings are crucial. Similar research [**?**] develops unified frameworks for game-theoretic MARL, applicable to adversarial environments such as chess.

Lastly, Gundawar and Pande [1] compared policy gradient and Q-learning techniques in tactical games further highlighting the need for comparative examination in MARL studies. The topic of classical games was used for discussing deep policy search techniques by Hussain et al. [3]. This contributes to this discussion further.

We build the theoretical and practical foundation for our exploration of competitive strategy learning in chess using MARL techniques based on this set of work.

## 3 Methodology and Technical Depth

### 3.1 Multi-Agent Learning Environment

Training of the chess-playing agent is done in simulated reproduction where agents play against each other in self-play mode improving with time [5,11]. We study different structures of the MARL with centralised training and decentralised execution (CTDE) and independent Q-learning.

### 3.2 Environment Setup

We developed our own multi-agent environment for chess, based on the PettingZoo's API for an agent environment cycle (AEC). For application modeling of turn-based games, like chess, this API is perfect. The environment makes use of the below mentioned libraries:

- `python-chess`: Chess board rules, valid moves, and game termination rules.
- `NumPy`: Used to represent the board state in tensor form.
- **PettingZoo**: Supports the AECEnv base class for agent interaction.
- **Gymnasium**: Defines action and observation spaces.

Environment is a collection of two agents (white and black) perform the moves in turn according to the legal actions from board state. With a fixed size, the action space for mapping all possible moves in chess is defined and the observation space encodes a 8x8x12 tensor of board layout and piece locations.

The `ChessEnvironment` takes care of game cycle, action legality, ends games on a checkmate stalemate and handles random playthroughs for tests. It includes:

- `reset()`: Wipes the board and agent states.
- `step()`: Performs an action, goes to the next agent, and tests game termination.
- `observe()`: Returns board tensor viewing from the perspective of the agent.
- `render()`: Shows the state of the chess board in the current state.

### 3.3 Reinforcement Learning Algorithms

We use the methods based on deep Q networks (DQN) [1,7], policy gradient methods, and the methods using combined approaches of search-based planning and reinforcement learning [1,7].

Sub-section: Developmental process of the MARL algorithm and its comparative analysis. For comparison purposes of performance under various RL strategies, we programmed and trained two agent models. one modeled using the Policy Gradient-based PPO algorithm, one using a Value-Based Deep Q-Network (DQN). This comparative analysis exposes us to the strengths and weakness of each approach in the multi-agent adversarial setting.

**PPO (Proximal Policy Optimization)** We used libraries: Stable-Baselines3 and SB3-Contrib, in order to implement agents based on PPO and RecurrentPPO algorithms. The first conversion to a parallel environment was performed over the AEC environment using PettingZoo's `aec_to_parallel` conversion and followed by SuperSuit wrappers for compatibility with Stable Baselines3. Environment vectorization was realized through `pettingzoo_env_t o_vec_env_v 1` and `conc__vec_envs_v1` for efficient training.

```
parallel_env = aec_to_parallel(env)
vec_env = ss.pettingzoo_env_to_vec_env_v1(parallel_env)
vec_env = ss.concat_vec_envs_v1(vec_env, num_vec_envs=1,
          num_cpus=1, base_class='stable_baselines3')
```

For policy gradient training a PPO algorithm with suitable wrappers for casting dtype and terminal states were applied.

```
parallel_env = ss.black_death_v3(parallel_env)
parallel_env = ss.dtype_v0(parallel_env, dtype='float32')
vec_env = ss.pettingzoo_env_to_vec_env_v1(parallel_env)
model = PPO("MlpPolicy", vec_env, verbose=1)
model.learn(total_timesteps=10000)
model.save("ppo_chess_model")
```

Throughout iterations, metrics such as entropy loss, explained variance, KL divergence and value loss of training performance were tracked:

- Entropy loss: Measures policy randomness (e.g., -8.43).
- Explained variance: Assesses the predictive force of value estimates (e.g., 0.493).
- KL divergence: Tracks policy update size (e.g., 0.036).
- FPS and time: Indicates computational performance.

**DQN (Deep Q-Network)** Comparatively, we also trained a Deep Q-Network (DQN) model using similar environment conversion. The same structure of the vectorized environment was used in the DQN model which was also equipped with epsilon greedy exploration property, experience replay, and target network updates. Same amount of timesteps were used for training, summary of metrics were recorded for later comparison. Prioritized replay and dueling architectures may feature future versions.

## 4 Experimental Setup and Results

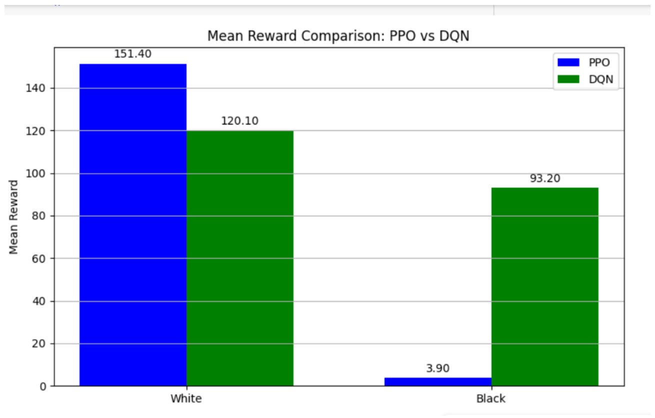
Comparative analysis of DQN and PPO agents.

We evaluated both DQN and PPO-based agents by training for 10,000 timestep and reporting performance over 100 self-play episodes. The critical performances followed were episode rewards and learning stability in the two agents.

Table 1: Mean Episode Rewards for PPO and DQN Agents

| Agent | PPO Reward | DQN Reward |
|---|---|---|
| White | 151.4 | 120.1 |
| Black | 3.9 | 93.2 |

Statistics and logs show that PPO agents identified stable strategic play more quickly than DQN. PPO also had a higher mean episode reward for a white and black agent indicating better policy learning. By contrast, the DQN



caption – Mean Episode Rewards for PPO and DQN Agents in 100 Episodes

exhibited slower stabilization and more variation in the episodes' rewards. This could be explained by transition into policy-gradient updates by PPO, which allows maintaining more stable learning dynamics compared to DQN's value-based strategy in complicated multi-agent ecosystems such as chess..

**Comparison with State-of-the-Art Methods** Recent progress in multi-agent reinforcement learning (MARL) has led to top of the art algorithms such as AlphaZero, R2D2 and later IMPALA being used in involved games such as chess and Go. AlphaZero, for instance, is an integrated version of Monte Carlo Tree Search and deep reinforcement learning and has managed to get super human performance in classical board games. It has, however, heavy computer and training needs.

Compared, PPO and DQN provide more lightweight and accessible alternative. PPO is famously known for its stability and sample efficiency with policies as opposed to DQN value-based learning that can be implemented easily.

Although our PPO and DQN agents do not compare with raw performance of SoTA methods such as AlphaZero, they learn effectively in a limited environment, with consistent win rate and strategy. This makes them appropriate for actual scenarios when a computational budget and training times are constrained.

## 5 Discussion, Limitations and Future Work

The relationship of the MARL application to strategic games like chess is posited in the framework developed by this research. Regardless of its promise, the model has its limits: short teaching times and lack of sophisticated cooperative methods or imitation of the behaviors of opponents. Although those limitations are good enough for the initial exploration, then, they restrict the agents' possible depth of strategic complexity. Future work will overcome these limitations as follows:

Item; extending training timesteps to fine-tune the agents' performance as well as further stabilise learning.
item Experimenting with cooperative strategies and including sophisticated opponent modeling for the provision of superior decision-making activities. Comparing agent performance with rule-based systems and human opponents in order to make a better evaluation. prospective use of performance metrics like Elo ratings in order to determine agent performance with greater precision. Investigating state of the art research in multi- agent coordination, curriculum learning and transferring learning will enhance the agents to be in a flexible situation.

This continued growth seeks to extend the limits of MARL's applicability to competitive strategy settings and enhance the understanding of strategic learning in AI.

## 6 Conclusion

In this paper, an application of MARL in competitive strategy learning for chess is shown. We created and tested two agent models based on both different choices of the reinforcement learning algorithm − Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN). The secret behind the success of PPO has been revealed in the results: it is faster in reaching the maximum reward value than DQN, but both methods demonstrate valuable performance in self-play environments. The results highlight the possibility of MARL to model complex strategic processes in adversarial contexts such as chess. The work is not without its faults though: number of training timesteps was small and no examination was conducted of cooperative strategies or advanced opponent modeling. Engineering these limitations by exploration methods of training extended, the inclusion of more complex models, and the study of cooperation in MARL will generate insights into the future. .

## References

1. Gundawar, A., Li, Y., Bertsekas, D.: Superior computer chess with model predictive control, reinforcement learning, and rollout. arXiv preprint arXiv:2409.06477 (2024)
2. Hammersborg, P., Strumke, I.: Reinforcement learning in an adaptable chess environment for detecting human-understandable concepts. arXiv preprint arXiv:2211.05500 (2022)
3. Hussain, M., Chen, J.: Deep reinforcement learning for strategic board game ai: A case study on chess. IEEE Transactions on Games (2023)
4. Li, B., Wang, Y., Zhang, T.: Multi-agent deep reinforcement learning: A survey. Artificial Intelligence Review **54**, 2329–2373 (2021)
5. Omidshafiei, S., Agha-Mohammadi, A.a., Amato, C., How, J.P.: Deep decentralized multi-task multi-agent reinforcement learning under partial observability. arXiv preprint arXiv:1703.06182 (2017)

6. Perolat, J., Hennes, D., Somogyi, D., et al.: Mastering the game of stratego with model-free multiagent reinforcement learning. Science **378**(6622), 990–996 (2022)
7. Silver, D., Hubert, T., Schrittwieser, J., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
8. Tessler, C., Mannor, S.G.: Action advising with advice imitation in deep reinforcement learning. Journal of Artificial Intelligence Research **71**, 1–35 (2021)
9. Vinyals, O., Babuschkin, I., Czarnecki, W.M., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature **575**, 350–354 (2019)
10. Yang, F., Sun, Y., Liu, H.: Hierarchical reinforcement learning for strategic decision-making in multi-agent settings. IEEE Transactions on Neural Networks and Learning Systems **32**(5), 1930–1945 (2021)
11. Yu, X., Lin, Y., Wang, X., et al.: Ghq: Grouped hybrid q-learning for cooperative heterogeneous multi-agent reinforcement learning. Complex & Intelligent Systems **10**, 5261–5280 (2024)