Generalized Self-Play Reinforcement Learning for Othello under Dynamic Board Constraints

Byeongchang Kim*, Yeaeun Lee†, Euiseok Hwang‡
*Department of AI Convergence, †‡Department of Electrical Engineering and Computer Science
Gwangju Institute of Science and Technology
Gwangju Metropolitan City, Republic of Korea
{*kbc202179, †leeyeaeun}@gm.gist.ac.kr, ‡euiseokh@gist.ac.kr

***Abstract*—This research presents a self-play reinforcement learning framework for the game of Othello, enabling generalization across diverse board constraints. These constraints include variable board sizes, blocked cells, and total inference time limitations. FastOthelloNet incorporates a lightweight convolutional input architecture and employs Monte Carlo Tree Search (MCTS) for efficient planning. Unlike AlphaZero, which assumes a fixed board structure, or MuZero, which explicitly models latent dynamics, FastOthelloNet is trained directly on a randomized Othello environment with dynamic constraints, eliminating the need for a complex world model.**

***Index Terms*—Reinforcement learning, generalization, self-play, Othello, Monte Carlo Tree Search, convolutional neural networks**

I. Introduction

Recent research in deep reinforcement learning has shown significant progress in strategic decision-making tasks, particularly in board games. Algorithms such as AlphaZero [1] and MuZero [2] have demonstrated superhuman performance by combining Monte Carlo Tree Search (MCTS) with deep neural networks. Othello, also known as Reversi, is a well-known board game that has attracted various research efforts aiming to surpass human expert-level performance. However, these models are trained in static environments and require complex architectures to achieve high performance. Consequently, they exhibit limitations in generalizing to dynamic board constraints such as varying board sizes, blocked cells, and restricted inference time. *FastOthelloNet* is designed and trained for robust generalization to dynamic board constraints through self-play in randomized board conditions. The environment configuration is randomized at the beginning of each episode. During training, parameters such as board size ((6×6), (8×8)), the placement of blocked cells, and a strict inference time limit (10 seconds) are varied. This diversity encourages the emergence of strategies that are not overfitted to a specific layout or rule set. To ensure computational efficiency under varying spatial dimensions, *FastOthelloNet* adopts a lightweight convolutional architecture with depthwise separable convolutions. Instead of explicitly modeling transition dynamics, the model achieves generalization by being exposed to a broad distribution of gameplay conditions during training.

II. Background and Related Work

A. MCTS-based Self-Play Learning

AlphaZero [1] demonstrated that reinforcement learning agents can attain expert-level performance in games such as Chess, Shogi, and Go through self-play, without the need for expert demonstrations or handcrafted rules. This approach integrates a deep residual network with MCTS, where the network is trained using data collected from tree-guided simulations. Policy targets are derived from search visit distributions, while value targets reflect the final game outcome from the agent's perspective. A composite loss function is used to update the shared network by combining both objectives.

Expanding upon this foundation, MuZero [2] introduced a generalized framework that does not require access to the true environment dynamics. Instead of modeling the environment explicitly, MuZero jointly learns a latent dynamics model alongside the policy and value networks. This enables the agent to perform planning through internal representations of state transitions and rewards, showing strong performance in visually complex domains such as Atari.

While both AlphaZero and MuZero have achieved impressive results, these methods typically operate under the assumption of a fixed and fully observable environment during training. Key aspects such as board dimensions, state representations, and action spaces are held constant. As a result, their ability to adapt to novel structures or modified gameplay mechanics remains limited.

B. Toward Generalization and Environmental Diversity

Recent studies have highlighted the importance of generalization in reinforcement learning. EfficientZero [3] extends MuZero with improved data efficiency but remains limited to single-environment configurations. DreamerV3 [4] achieves strong generalization in continuous control domains through the use of a world model with symlog targets and long-horizon planning. However, these approaches primarily address visually or physically complex environments rather than structured, rule-based games.

In contrast, research in procedural and open-ended reinforcement learning suggests that environmental diversity during training improves agent robustness. The ProcGen benchmark [5], for instance, evaluates generalization using procedurally generated levels, where agents benefit from architectural

invariance or population-based training. Gato [6] explores multi-task learning across modalities, though its performance in combinatorial games remains limited.

The present work builds on these insights by focusing on symbolic environments with discrete action spaces and rigid rules. Variation in board size and playable regions is introduced within the Othello domain to induce generalization, without relying on learned world models or extensive data augmentation.

C. Othello and Previous Approaches

Othello, also known as Reversi, has been widely used as a benchmark for evaluating game-playing algorithms. Early systems relied on minimax search with handcrafted heuristics or pattern-based evaluation functions, embedding strategic knowledge directly into the evaluation process and focusing on exhaustive lookahead under a fixed rule set.

With the rise of deep reinforcement learning, AlphaZero-style methods have been applied to Othello using self-play and neural network-guided Monte Carlo Tree Search. These models have demonstrated strong performance on the standard 8×8 board, often surpassing traditional search-based approaches in fixed configurations [7], [8].

Nevertheless, most implementations remain constrained to static environments with fixed board geometry and deterministic rules. Variants involving spatial constraints or altered board structures are generally not considered. As a result, trained agents frequently fail to generalize when confronted with changes such as different board sizes or blocked regions.

The proposed approach addresses this limitation by introducing environmental variation during training. Each episode features a randomized board size and may include unplayable cells that disrupt standard tactics. This exposure to structural diversity enables the agent to learn adaptable policies without requiring retraining or environment-specific adjustments.

III. Proposed Method

The proposed method integrates MCTS with a neural policy and value network. Learning is performed through self-play over a distribution of randomized Othello environments. The key objective is to promote generalization by exposing the agent to diverse spatial constraints during training. The neural network architecture, *FastOthelloNet*, is designed to maintain efficiency and robustness across varying board sizes and configurations.

A. Randomized Othello Environment

A family of Othello environments is defined by two variables: board dimension $N \times N$ and blocked cell ratio r. Blocked cells are permanently unplayable and excluded from the valid action space. Prior to each episode, the board size N is sampled from a predefined range (e.g., 6 to 8), and a proportion of cells determined by r are randomly selected as blocked. The initial stone placement is preserved to maintain game validity. Training under these varying configurations enables the agent to adapt its policy to dynamic topologies and variable action spaces.

B. FastOthelloNet Architecture

FastOthelloNet is a lightweight Convolutional Neural Network (CNN) tailored for spatially structured games such as Othello. The network receives a two-channel board encoding as input and outputs: (1) a policy distribution over all board positions, and (2) a scalar value estimating the expected game outcome from the current state.

To support varying board sizes (e.g., 6×6, 8×8), the model is implemented as a fully convolutional network, containing no layers that assume fixed input dimensions. At the core of the architecture are depthwise separable convolutions, which decouple spatial and channel-wise operations. This design significantly reduces the number of parameters and computational cost compared to standard convolutions.

The model begins with a 3×3 depthwise convolution, followed by a pointwise convolution and ReLU activation. Multiple such blocks are stacked to form the network backbone. From the final feature map, two output heads are derived:

- **Policy Head**: A 1×1 convolution projects the feature map to a set of logits over the board grid. A softmax function is applied to produce a probability distribution over possible actions.
- **Value Head**: A global average pooling is applied across spatial dimensions, followed by a fully connected layer with $\tanh$ activation to produce a scalar value in the range $[-1, 1]$.

This architecture is specifically designed for fast inference during MCTS rollouts, where each model query must complete within a predefined computational budget. In the experimental setup, the total inference time for an entire game is constrained to a **10-second time limit**, including all neural network evaluations performed during MCTS. This constraint reflects the demands of time-sensitive applications and stands in contrast to large AlphaZero-style architectures that require substantial computational resources. FastOthelloNet enables search-based reinforcement learning in resource-constrained or embedded environments.

C. Monte Carlo Tree Search for Decision Making

Actions are selected using MCTS, which integrates the structure of the search tree with predictions from the neural network. Each node in the tree corresponds to a specific game state, and each edge represents a possible action leading to a successor state.

The search begins at the root node, where the algorithm balances exploitation and exploration to select actions. Exploitation favors actions with high value estimates, while exploration encourages visits to less-explored actions. This balance is achieved using the Predicted Upper Confidence bound applied to Trees (PUCT) formula [1].

During the selection phase, each child node is evaluated using an upper confidence bound:

$$U(s,a) = c_{\text{PUCT}} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{1}$$

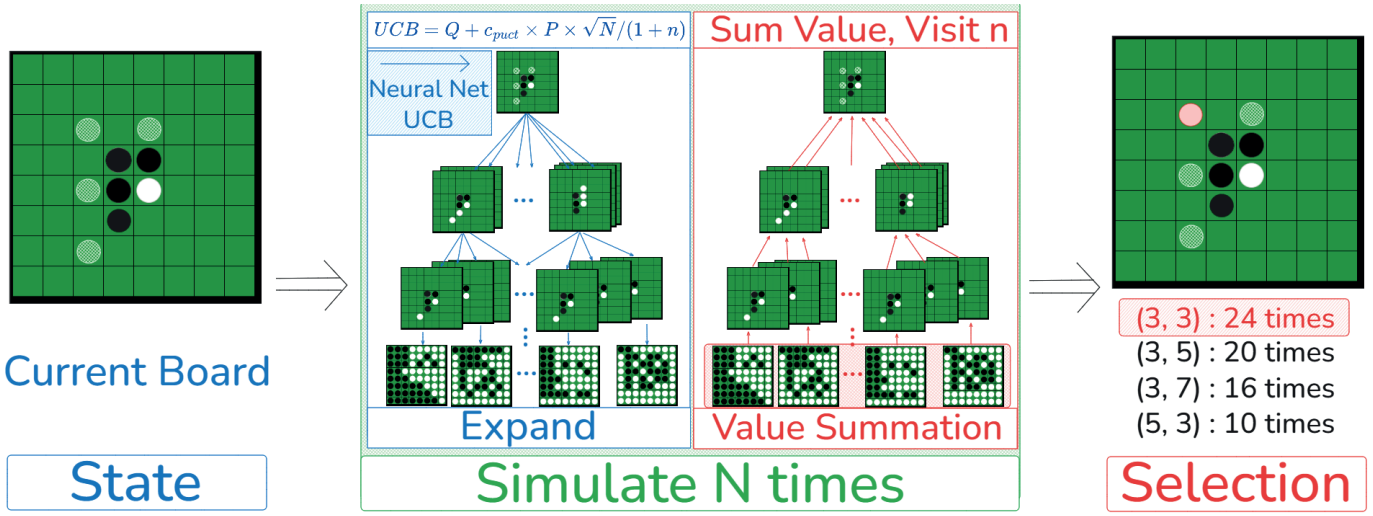


Fig. 1: FastOthelloNet architecture for value and policy prediction. The model uses depthwise separable convolutions to reduce computation. A fully convolutional backbone handles variable board sizes, followed by a policy head and a value head.

Here, $P(s, a)$ is the prior probability from the policy network, $N(s, a)$ is the visit count for action a at state s, and c_{PUCT} is a constant that controls the level of exploration.

At each step, the child node with the highest score, computed as $Q(s, a)+U(s, a)$, is selected. Here, $Q(s, a)$ represents the mean value of prior simulations, and $U(s, a)$ is the exploration term defined in (1). This selection is applied recursively until an unexpanded node is reached.

At the leaf node, the selected move is applied to advance the environment, yielding a new game state. The neural network then evaluates this state, producing a policy distribution over legal actions and a scalar value estimating the expected outcome.

These predictions are stored in the expanded node: the policy serves as a prior for future expansions, and the value is treated as the simulation result. During backpropagation, all nodes along the traversal path are updated. For each edge, the visit count $N(s, a)$ is incremented, and the value estimate $Q(s, a)$ is updated via incremental averaging.

This process is repeated for a fixed number of simulations, refining action evaluations at the root. During training, actions are sampled from the visit count distribution to encourage exploration; during evaluation, the action with the highest visit count is selected as the final move.

D. Training Pipeline

The training process follows a repeated two-stage cycle: self-play data generation and neural network parameter updates.

In the self-play stage, the agent plays games against itself using MCTS guided by the neural network. At each step, the following elements are recorded: the board state, the MCTS-derived policy (from visit counts), and the final game outcome. These form training tuples consisting of a board state tensor, a target policy distribution, and a scalar game result.

After accumulating a sufficient number of games, the training stage begins. The neural network is optimized to predict both the policy and value targets. The training objective consists of two loss terms.

The first loss term is the policy loss, defined as the cross-entropy between the MCTS visit count distribution and the network's predicted policy. For a given state s, it is computed as:

$$L_{\text{policy}}(s) = -\sum_{a} \pi_a \log p(a|s) \tag{2}$$

Here, π_a is the normalized visit count for action a, reflecting how frequently it was selected during MCTS. The term $p(a|s)$ denotes the predicted probability for action a. This loss encourages the network to imitate the MCTS decision pattern.

The second term is the value loss, defined as the mean squared error between the predicted value and the actual game outcome:

$$L_{\text{value}}(s) = (v(s) - z)^2 \tag{3}$$

In this expression, $v(s)$ is the value predicted by the network, representing the estimated outcome from state s. The ground-truth result z is set to $+1$ for a win, -1 for a loss, and 0 for a draw.

The total loss combines both objectives:

$$L(s) = L_{\text{policy}}(s) + L_{\text{value}}(s) \tag{4}$$

The network is trained via gradient-based optimization, using either stochastic gradient descent or the Adam optimizer. Gradients are computed over mini-batches of training tuples collected from self-play.

To promote generalization, the training environment is progressively made more challenging by varying the board size

and the ratio of blocked cells. These variations force the agent to adapt to diverse spatial structures.

Throughout training, inference efficiency is preserved. The *FastOthelloNet* architecture maintains low computational overhead, enabling real-time decision-making under strict time constraints. This training pipeline supports effective learning in resource-constrained settings while achieving robust generalization performance.

IV. Experiments

The experiments were designed to evaluate the generalization capability of the proposed method across diverse board configurations. The assessment focuses on both absolute performance and adaptability to previously unseen structures.

A. Experimental Setup

The agent was trained using self-play reinforcement learning with MCTS. Each self-play episode was initialized on a randomized Othello board. The board size was sampled from either 6×6 or 8×8. For the 8×8 partial board, four blocked cells were fixed at predefined positions, while the 6×6 board contained no blocked cells.

B. Training and Evaluation Procedure

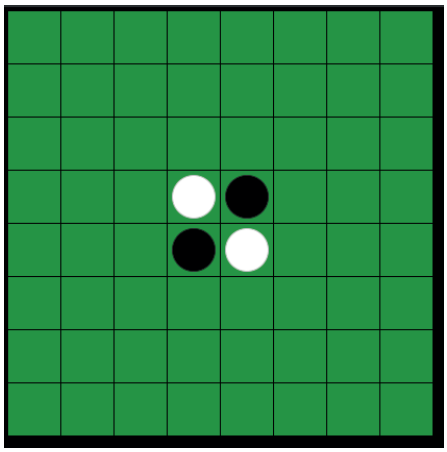
Algorithm 1 Training and Evaluation Pipeline

```
Initialize FastOthelloNet weights θ
while not converged do
    for each self-play game do
        Sample board size N ∈ {6, 8}
        Initialize environment with randomized board
        while game not over do
            Use MCTS(θ) to select action a from state s
            Store (s, π) from MCTS statistics
            Play action a to advance environment
        end while
        Assign final outcome z to all stored states
        Add (s, π, z) to training buffer
    end for
    Update θ on sampled (s, π, z) using gradient descent
end while
for board type ∈ {standard, small, partial} do
    for opponent ∈ {random, greedy, corner, positional} do
        Play evaluation matches using MCTS(θ)
        Record win rate and tournament rank
    end for
end for
```

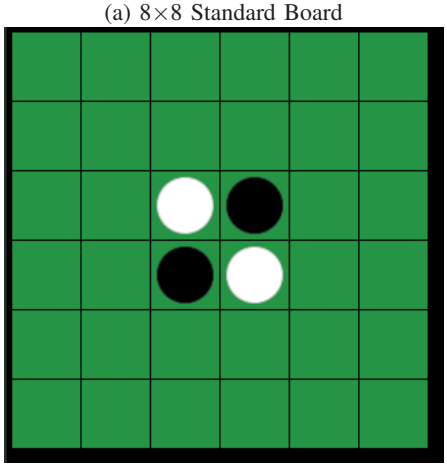
C. Evaluation

The trained agent was tested against four rule-based opponents, with evaluations conducted separately on three board configurations as illustrated in Figure 2.

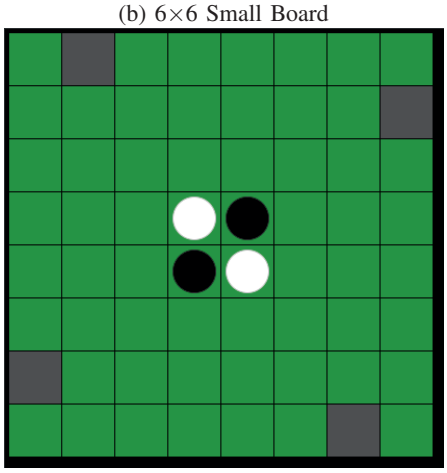
- **8×8 Standard** – The full Othello board without any blocked cells, representing the classic game setup.
- **6×6 Small** – A smaller board with fewer playable cells, requiring tighter spatial strategies.
- **8×8 Partial** – A standard-sized board with four blocked cells in fixed positions, introducing asymmetry and spatial constraints.



(a) 8×8 Standard Board



(b) 6×6 Small Board



(c) 8×8 Partial Board

Fig. 2: Examples of board configurations used in evaluation: standard, small, and partial (with blocked cells).

The rule-based agents used in the evaluation were:

- **Random** – Uniform random action selection.
- **Greedy** – Selects the move that flips the most discs immediately.
- **Corner** – Prioritizes available corner moves.
- **Positional** – Uses a static positional weight table.

Round-robin tournaments were conducted with varying numbers of games. For each evaluation, the win rate against each baseline agent, tournament ranking, and overall average performance metrics were recorded.

V. Results

The agent's performance is evaluated using two metrics: win rate against rule-based agents and average tournament rank across different board configurations. These metrics are tracked as the number of self-play training games increases. All evaluations are conducted under a strict constraint: **the total inference time per game must not exceed 10 seconds**, ensuring practical deployability in real-time settings. Notably, the same trained model is evaluated across all board types—8×8 standard, 6×6 small, and 8×8 partial—without any retraining or architectural modification, demonstrating robustness and generalization across diverse environments.

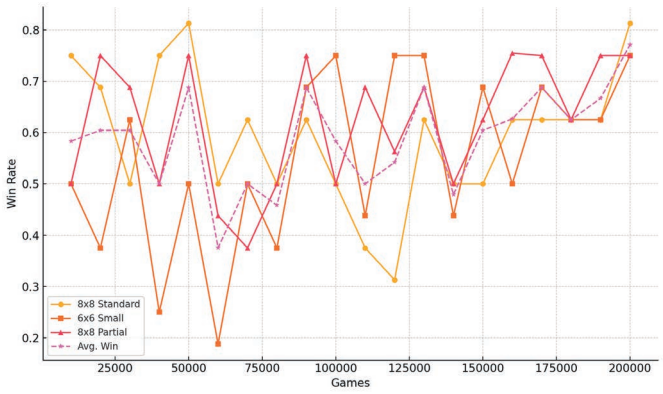


Fig. 3: Win rate over training games on each board type. Average win rate shown in bold.

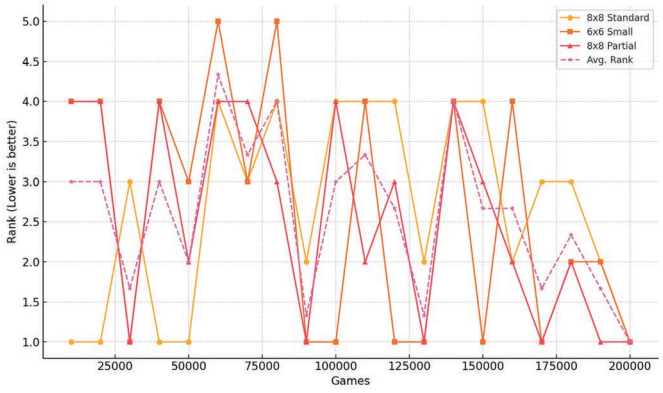


Fig. 4: Tournament rank over training games. Lower rank is better. Average rank shown in bold.

Figure 3 illustrates the evolution of win rate throughout training. On the 8×8 standard board, the agent exhibits steady improvement, reaching a final win rate of 81.3% after 200,000 self-play games. This strong performance is attributed to the board's frequent appearance during training. On the 6×6 board, learning is slower and less stable in the early stages, likely due to shorter game lengths that increase the impact of individual moves. Nevertheless, the agent adapts over time, achieving a win rate of 75.0% by the end of training. On the 8×8 partial board—with four fixed blocked cells—the agent demonstrates consistent progress, reaching 75.0% after 200,000 games, indicating its ability to handle asymmetric and irregular board structures.

Figure 4 shows the agent's tournament rankings during training. On the standard 8×8 board, the agent frequently maintains rank 1 in early stages. However, performance temporarily declines to ranks 2 or 3 between 100,000 and 180,000 games, followed by partial recovery in later stages.The 6×6 board exhibits the greatest fluctuation in performance, with rankings ranging from 1 to 5. Early training phases show instability, likely due to the reduced game length increasing the strategic impact of individual moves. Despite this, the agent improves in later stages, frequently achieving rank 1 after 90,000 games. On the 8×8 partial board, which includes blocked cells, the ranking trend is more stable. Following a noisy initial phase, the agent steadily improves, consistently attaining rank 1 from approximately 130,000 games onward. The magenta dashed line in Figure 4 represents the average tournament rank across all board types. While it fluctuates between 1.3 and 4.3 during early training, it converges to 1.0 by the end. This convergence demonstrates the agent's increasing generalization capability across structurally diverse environments, enabling it to outperform rule-based opponents across all configurations.

These results confirm that the proposed training pipeline, which relies on sampling from a diverse distribution of randomized environments, effectively produces policies that generalize across board variations. In particular, the ability of a single unified model to perform robustly on all three board types(standard, small, and partially blocked)without any architectural or parameter modification highlights the generality and stability of the learned policy.

In contrast to rule-based agents, which depend on fixed heuristics and often fail under altered board geometries, the learned agent adapts dynamically. It identifies board specific opportunities while maintaining general strategies for control, mobility, and disc flipping. All evaluations were conducted under a strict inference constraint, requiring each game to complete within a 10 second total decision time budget. Despite this limitation, the agent consistently achieved real-time performance, indicating its suitability for interactive and embedded applications.

VI. Conclusion

This work presents a generalized self-play reinforcement learning framework for Othello that achieves robust gener-

alization across diverse board configurations. In contrast to conventional approaches that train agents in a single, fixed environment, the proposed method produces a single neural agent capable of adapting to multiple game variants—including different board sizes and blocked cell layouts—without requiring retraining or architectural modifications.

The proposed agent integrates MCTS with a lightweight convolutional network, *FastOthelloNet*, and is trained exclusively through self-play in a distribution of randomized environments. This setup enables the model to acquire transferable strategies that are not specific to any single configuration, thereby promoting strong generalization. The trained agent consistently outperforms rule-based opponents across three board types: standard, small, and partially blocked.

Importantly, the model operates under a strict real-time constraint: **the total inference time per game is limited to 10 seconds**, demonstrating the practical viability of the proposed method for interactive applications.

The findings demonstrate that AlphaZero-style architectures can be extended to support generalization *without explicit environment modeling*, in contrast to methods such as MuZero. When trained on a sufficiently diverse distribution and combined with efficient planning, even compact models can learn robust and high-performing policies. This work provides evidence for the feasibility of reinforcement learning agents that are not only strong, but also flexible, adaptive, and efficient across structurally diverse tasks.

VII. Future Work

This study opens several directions for future research.

(1) Extension to Other Games. The proposed training paradigm can be extended to other board games and environments with structural variability, such as dynamic grid sizes, obstacles, or rule changes. Generalization across such variations in Othello lays the foundation for agents capable of adapting to a broader class of strategic games, including Hex, Connect Four, or turn-based video games. These extensions would further evaluate the agent's ability for structural abstraction and transferable decision-making [5], [9].

(2) Integration with World Models. While the current approach relies on MCTS and model-free policy learning, integrating latent dynamics models presents a promising direction. Combining this generalization-driven training with world model approaches such as Dreamer [4], [10], [11] could enable imagined rollouts in latent space, improving sample efficiency and planning depth. Additionally, generalist models like Gato [6] offer a framework for unifying vision, language, and control via transformer-based architectures. Such hybrid agents may reduce reliance on explicit search while maintaining adaptability under diverse environments and limited computational resources.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2710084144).

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, 2020.

[3] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, "Mastering atari games with limited data," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 25 476–25 488.

[4] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," *arXiv preprint arXiv:2301.04104*, 2023.

[5] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 2048–2056. [Online]. Available: https://proceedings.mlr.press/v119/cobbe20a.html

[6] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas, "A generalist agent," 2022. [Online]. Available: https://arxiv.org/abs/2205.06175

[7] Z. Wu, J. Zhang, Y. Tian, J. Chen, and L. Xie, "Accelerating self-play learning in go and othello with neural networks and tree search," *arXiv preprint arXiv:1903.08129*, 2019.

[8] P. Liskowski, W. Jaśkowski, and K. Krawiec, "Learning to play othello with deep neural networks," *IEEE Transactions on Games*, vol. 10, no. 4, pp. 354–364, 2018.

[9] H. Furuta, Y. Matsuo, and S. S. Gu, "Generalized decision transformer for offline hindsight information matching," 2022. [Online]. Available: https://arxiv.org/abs/2111.10364

[10] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[11] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.