# Reinforcement Learning in Strategy-Based and Atari Games: A Review of Google DeepMind's Innovations

Abdelrhman Shaheen
*Computer Science Engineering Undergraduate Student*
*Egypt Japan University of Science and Technology*
Alexandria, Egypt
abdelrhman.shaheen@ejust.edu.eg

Anas Badr
*Computer Science Engineering Undergraduate Student*
*Egypt Japan University of Science and Technology*
Alexandria, Egypt
anas.badr@ejust.edu.eg

Ali Abohendy
*Computer Science Engineering Undergraduate Student*
*Egypt Japan University of Science and Technology*
Alexandria, Egypt
ali.abohendy@ejust.edu.eg

Hatem Alsaadawy
*Computer Science Engineering Undergraduate Student*
*Egypt Japan University of Science and Technology*
Alexandria, Egypt
hatem.alsaadawy@ejust.edu.eg

Nadine Alsayad
*Computer Science Engineering Undergraduate Student*
*Egypt Japan University of Science and Technology*
Alexandria, Egypt
nadine.alsayad@ejust.edu.eg

***Abstract*—Abstract**

***Index Terms*—Deep Reinforcement Learning, Google DeepMind, AlphaGo, AlphaGo Zero, MuZero, Atari Games, Go, Chess, Shogi,**

## I. Introduction

Artificial Intelligence (AI) has revolutionized the gaming industry, both as a tool for creating intelligent in-game opponents and as a testing environment for advancing AI research. Games serve as an ideal environment for training and evaluating AI systems because they provide well-defined rules, measurable objectives, and diverse challenges. From simple puzzles to complex strategy games, AI research in gaming has pushed the boundaries of machine learning and reinforcement learning. Also The benfits from such employment helped game developers to realize the power of AI methods to analyze large volumes of player data and optimize game designs. [1]
Atari games, in particular, with their retro visuals and straightforward mechanics, offer a challenging yet accessible benchmark for testing AI algorithms. The simplicity of Atari games hides complexity that they require strategies that involve planning, adaptability, and fast decision-making, making them also a good testing environment for evaluating AI's ability to learn and generalize.
The development of AI in games has been a long journey, starting with rule-based systems and evolving into more sophisticated machine learning models. However, the machine learning models had a few challenges, from these challenges is that the games employing AI involves decision making in the game evironment. Machine learning models are unable to interact with the decisions that the user make because it depends on learning from datasets and have no interaction with the environment. To overcome such problem, game developers started to employ reinforcement learning (RL) in developing games. Years later, deep learning (DL) was developed and shows remarkable results in video games [2]. The combination between both reinforcement learning and deep learning resulted in Deep Reinforcement Learning (DRL). The first employment of DRL was by game developers in atari game [3]. One of the famous companies that employed DRL in developing AI models is Google DeepMind. This company is known for developing AI models, including games. Google DeepMind passed through a long journey in developing AI models for games. Prior to the first DRL game model they

develop, which is AlphaGo, Google DeepMind gave a lot of contributions in developing DRL, by which these contributions were first employed in Atari games.

For the employment of DRL in games to be efficient, solving tasks in games need to be sequential, so Google DeepMind combined RL-like techniques with neural networks to create models capable of learning algorithms and solving tasks that require memory and reasoning, which is the Neural Turing Machines (NTMs) [4]. They then introduced the Deep Q-network (DQN) algorithm, which is combine deep learning with Q-learning and RL algorithm. Q-learning is a model in reinforcement learning which use the Q-network, which is is a type of neural network to approximate the Q-function, which predicts the value of taking a particular action in a given state [5]. The DQN algorithm was the first algorithm that was able to learn directly from high-dimensional sensory input, the data that have a large number of features or dimensions [6].

To enhance the speed of learning in reinforcement learning agents, Google DeepMind introduced the concept of experience replay, which is a technique that randomly samples previous experiences from the agent's memory to break the correlation between experiences and stabilize the learning process [7]. They then developed asynchronus methods for DRL, which is the Actor-Critic (A3C) model. This model showed faster and more stable training and showed a remarkable performance in Atari games [8]. By the usage of these algorithms, Google DeepMind was able to develop the first AI model that was able to beat the world champion in the game of Go, which is AlphaGo in 2016.

There are a lot of related work that reviewed the reinforcement learning in strategy-based and atari games. Arulkumaran et al [9] make a brief introduction of DRL, covering central algorithms and presenting a range of visual RL domains Zhao et al. [10] and Tang et al. [11] survey the development of DRL research, and focus on AlphaGo and AlphaGo Zero models. Shao et al. [12] review the development of DRL in game AI, from 2D to 3D, and from single-agent to multi-agent, and discuss the real-time strategy games.

In this paper, we will discuss the development that Google DeepMind made in developing AI models for games and the advancments. The main contribution will be the comparison between the three models AlphaGo, AlphaGo Zero, and MuZero, focusing on the challenges that each model faced and the improvements that were made. Also we will discuss the advancments that Google DeepMind made in developing AI models for games, reaching to the future directions.

## II. Background

Reinforcement Learning (RL) is a key area of machine learning that focuses on learning through interaction with the environment. In RL, an agent takes actions (A) in specific states (S) with the goal of maximizing the rewards (R) received from the environment. The foundations of RL can be traced back to 1911, when Thorndike introduced the Law of Effect, suggesting that actions leading to favorable outcomes are more likely to be repeated, while those causing discomfort are less likely to recur [13].

RL emulates the human learning process of trial and error. The agent receives positive rewards for beneficial actions and negative rewards for detrimental ones, enabling it to refine its policy function—a strategy that dictates the best action to take in each state. That's said, for a give agent in state $u$, if it takes action $u$, then the immediate reward $r$ can be modeled as $r(x, u) = \mathbb{E}[r_t \mid x = x_{t-1}, u = u_{t-1}]$.

So for a full episode of $T$ steps, the cumulative reward $R$ can be modeled as $R = \sum_{t=1}^{T} r_t$.

### A. Markov Decision Process (MDP)

In reinforcement learning, the environment is often modeled as a **Markov Decision Process (MDP)**, which is defined as a tuple $(S, A, P, R, \gamma)$, where:

- $S$ is the set of states,
- $A$ is the set of actions,
- $P$ is the transition probability function,
- $R$ is the reward function, and
- $\gamma$ is the discount factor.

The MDP framework is grounded in **sequential decision-making**, where the agent makes decisions at each time step based on its current state. This process adheres to the **Markov property**, which asserts that the future state and reward depend only on the present state and action, not on the history of past states and actions.

Formally, the Markov property is represented by:

$$P(s' \mid s, a) = \mathbb{P}[s_{t+1} = s' \mid s_t = s, a_t = a]$$

which denotes the probability of transitioning from state $s$ to state $s'$ when action $a$ is taken.

The reward function $R$ is similarly defined as:

$$R(s, a) = \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a]$$

which represents the expected reward received after taking action $a$ in state $S$.

### B. Policy and Value Functions

In reinforcement learning, an agent's goal is to find the optimal policy that the agent should follow to maximize cumulative rewards over time. To facilitate this process, we need to quantify the desirability of a given state, which is done through the **value function** $V(s)$. Value function estimates the expected cumulative reward an agent will receive starting from state $s$ and continuing thereafter. In essence, the value function reflects how beneficial it is to be in a particular state, guiding the agent's decision-making process. The value function is defined as:

$$V(s) = \mathbb{E}[G_t \mid s_t = s]$$

where $G_t$ is the cumulative reward from time step $t$ onwards. From here we can define the **action-value function** under

policy $\pi$ $Q_\pi(s, a)$, which again, estimates the expected cumulative reward from the state $s$ and taking action $a$ and then following policy $\pi$:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$$

$$= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s, a_t = a]$$

where $\gamma$ is the discount factor, which is a decimal value between 0 and 1 that detemines how much we care about immediate rewards versus future reward rewards [14].

### C. Reinforcement Learning Algorithms

There are multiple reinforcement learning algorithms that have been developed that falls under a lot of categories. But, for the sake of this review, we will focus on the following algorithms that have been used by the Google DeepMind team in their reinforcement learning models.

*1) **Monte Carlo Algorithm**:* The Monte Carlo Algorithm is a model-free reinforcement learning algorithm used to estimate the value of states or state-action pairs under a given policy by averaging the returns of multiple episodes. This method alternates between two main steps: policy evaluation and policy improvement.

- **Policy Evaluation:** The algorithm evaluates the value of a state or state-action pair by averaging the returns from multiple episodes that include that state or state-action pair.
- **Policy Improvement:** The algorithm improves the policy by selecting the action that maximizes the value of the state-action pair.

Since the algorithm is model-free and does not assume any prior knowledge of the environment's dynamics, it focuses on estimating state-action pair values ($Q(s, a)$) rather than just state values. The Monte Carlo Algorithm typically follows these steps:

1) Initialize the state-action pair values $Q(s, a)$ arbitrarily.
2) Start at a random state and take a random action, ensuring the possibility of visiting all state-action pairs over time.
3) Follow the current policy $\pi$ until a terminal state is reached.
4) Update the value of each state-action pair $Q(s, a)$ encountered in the episode using the observed returns.
5) For each state visited in the episode, update the policy $\pi(s)$ to select the action that maximizes $Q(s, a)$:

$$\pi(s) = \arg\max_a Q(s, a).$$

This algorithm is particularly well-suited for environments that are *episodic*, where each episode ends in a terminal state after a finite number of steps.

## III. AlphaGo

### A. Introduction

AlphaGo is a groundbreaking reinforcement learning model that utilizes neural networks and tree search to play the game of GO, which is thought to be one of the most challenging classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves.

AlphaGo uses value networks for position evaluation and policy networks for taking actions, that combined with Monte Carlo simulation achieved a 99.8% winning rate, and beating the European human Go champion in 5 out 5 games.

### B. Key Innovations

*Integration of Policy and Value Networks with MCTS:* AlphaGo combines policy and value networks in an MCTS framework to efficiently explore and evaluate the game tree. Each edge $(s, a)$ in the search tree stores:

- Action value $Q(s, a)$: The average reward for taking action $a$ from state $s$.
- Visit count $N(s, a)$: The number of times this action has been explored.
- Prior probability $P(s, a)$: The probability of selecting action $a$, provided by the policy network.

During the selection phase, actions are chosen to maximize:

$$a_t = \arg\max_a (Q(s, a) + u(s, a))$$

where the exploration bonus $u(s, a)$ is defined as:

$$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

When a simulation reaches a leaf node, its value is evaluated in two ways: 1. Value Network Evaluation: A forward pass through the value network predicts $v_\theta(s)$, the likelihood of winning. 2. Rollout Evaluation: A lightweight policy simulates the game to its conclusion, and the terminal result $z$ is recorded.

These evaluations are combined using a mixing parameter $\lambda$:

$$V(s_L) = \lambda v_\theta(s_L) + (1 - \lambda) z_L$$

The back propagation step updates the statistics of all nodes along the path from the root to the leaf.

### C. Training Process

*1) Supervised Learning for Policy Networks:* The policy network was initially trained using supervised learning on human expert games. The training data consisted of 30 million board positions sampled from professional games on the KGS Go Server. The goal was to maximize the likelihood of selecting the human move for each position:

$$\Delta\sigma \propto \nabla_\sigma \log p_\sigma(a|s)$$

where $p_\sigma(a|s)$ is the probability of selecting action $a$ given state $s$.

This supervised learning approach achieved a move prediction accuracy of 57.0% on the test set, significantly outperforming prior methods. This stage provided a solid foundation for replicating human expertise.

*2) Reinforcement Learning for Policy Networks:* The supervised learning network was further refined through reinforcement learning (RL). The weights of the RL policy network were initialized from the SL network. AlphaGo then engaged in self-play, where the RL policy network played against earlier versions of itself to iteratively improve.

The reward function used for RL was defined as:

$$r(s) = \begin{cases} +1 & \text{if win} \\ -1 & \text{if loss} \\ 0 & \text{otherwise (non-terminal states).} \end{cases}$$

At each time step $t$, the network updated its weights to maximize the expected reward using the policy gradient method:

$$\Delta\rho \propto z_t \nabla_\rho \log p_\rho(a_t|s_t)$$

where $z_t$ is the final game outcome from the perspective of the current player.

This self-play strategy allowed AlphaGo to discover novel strategies beyond human knowledge. The RL policy network outperformed the SL network with an 80% win rate and achieved an 85% win rate against Pachi, a strong open-source Go program, without using MCTS.

*3) Value Network Training:* The value network was designed to evaluate board positions by predicting the likelihood of winning from a given state. Unlike the policy network, it outputs a single scalar value $v_\theta(s)$ between $-1$ (loss) and $+1$ (win).

Training the value network on full games led to overfitting due to the strong correlation between successive positions in the same game. To mitigate this, a new dataset of 30 million distinct board positions was generated through self-play, ensuring that positions came from diverse contexts.

The value network was trained by minimizing the mean squared error (MSE) between its predictions $v_\theta(s)$ and the actual game outcomes $z$:

$$L(\theta) = \mathbb{E}_{(s,z)\sim D}\left[(v_\theta(s) - z)^2\right]$$

### *D. Challenges and Solutions*

AlphaGo overcame several challenges:

- Overfitting: Training the value network on full games led to memorization. This was mitigated by generating a diverse self-play dataset.
- Scalability: Combining neural networks with MCTS required significant computational resources, addressed through parallel processing on GPUs and CPUs.
- Exploration vs. Exploitation: Balancing these in MCTS was achieved using the exploration bonus $u(s, a)$ and the policy network priors.

### *E. Performance Benchmarks*

AlphaGo achieved the following milestones:

- 85% win rate against Pachi without using MCTS.
- 99.8% win rate against other Go programs in a tournament held to evaluate the performance of AlphaGo.
- Won 77%, 86%, and 99% of handicap games against Crazy Stone, Zen and Pachi, respectively
- Victory against professional human players such as Fan Hui (5-0) and Lee Sedol (4-1), marking a significant breakthrough in AI.

## IV. AlphaGo Zero

AlphaGo Zero

## V. MuZero

MuZero

## VI. Advancements

Advancements

## VII. Challenges and Future Directions

Challenges and Future Directions

## VIII. Conclusion

Conclusion


## Acknowledgment


## References

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**]. Refer simply to the reference number, as in [**?**]—do not use "Ref. [**?**]" or "reference [**?**]" except at the beginning of a sentence: "Reference [**?**] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [**?**]. Papers that have been accepted for publication should be cited as "in press" [**?**]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [**?**].

## REFERENCES


[1] N. Y. Georgios and T. Julian, Artificial Intelligence and Games. New York: Springer, 2018.

[2] N. Justesen, P. Bontrager, J. Togelius, S. Risi, (2019). Deep learning for video game playing. arXiv. https://doi.org/10.48550/arXiv.1708.07902

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, (2013). Playing Atari with deep reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1312.5602

[4] A. Graves, G. Wayne, I. Danihelka, (2014). Neural Turing Machines. arXiv. https://doi.org/10.48550/arXiv.1410.5401

[5] C.J.C.H.Watkins, P. Dayan, Q-learning. Mach Learn 8, 279–292 (1992). https://doi.org/10.1007/BF00992698

[6] DeepMind, (2015, February 12), Deep reinforcement learning. https://deepmind.google/discover/blog/deep-reinforcement-learning/

[7] T. Schaul, J. Quan, I. Antonoglou, D. Silver, (2015). Prioritized Experience Replay. arXiv. https://doi.org/10.48550/arXiv.1511.05952

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, (2016). Asynchronous Methods for Deep Reinforcement Learning. arXiv. https://doi.org/10.48550/arXiv.1602.01783

[9] A. Kailash, P. D. Marc, B. Miles, and A. B. Anil, (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine, vol. 34, pp. 26–38, 2017. arXiv. https://doi.org/10.48550/arXiv.1708.05866

[10] D. Zhao, K. Shao, Y. Zhu, D. Li, Y. Chen, H. Wang, D. Liu, T. Zhou, and C. Wang, "Review of deep reinforcement learning and discussions on the development of computer Go," Control Theory and Applications, vol. 33, no. 6, pp. 701–717, 2016 arXiv. https://doi.org/10.48550/arXiv.1812.01123

[11] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," Control Theory and Applications, vol. 34, no. 12, pp. 1529–1546, 2017.

[12] K. Shao, Z. Tang, Y. Zhu, N. Li, D. Zhao, (2019). A survey of deep reinforcement learning in video games. arXiv. https://arxiv.org/abs/1912.10944

[13] L. Thorndike and D. Bruce, Animal Intelligence. Routledge, 2017.

[14] R. S. Sutton and A. Barto, Reinforcement learning : an introduction. Cambridge, Ma ; London: The Mit Press, 2018.

[15] A. Kumar Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement Learning Algorithms: A brief survey," Expert Systems with Applications, vol. 231, p. 120495, May 2023