IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 July 2024, accepted 21 August 2024, date of publication 26 August 2024, date of current version 10 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3450297

RESEARCH ARTICLE

M0RV Model: Advancing the MuZero Algorithm Through Strategic Data Optimization Reuse and Value Function Refinement

XUEJIAN CHEN, YANG CAO, HONGFEI YU, AND CAIHUA SUN
School of Artificial Intelligence and Software, Liaoning Petrochemical University, Fushun 113001, China

Corresponding author: Yang Cao (caoyang821213@163.com)

This work was supported in part by the Basic Scientific Research Fund of Liaoning Provincial Education Department under Grant LJKMZ20220754 and Grant LJKMZ20220723.

ABSTRACT This paper introduces a model, M0RV, that improves the MuZero algorithm through data reuse and loss function optimization. It proposes reusing training trajectories generated by Monte Carlo Tree Search (MCTS) after filtering through an evaluation function trace into the training process, and on this basis, employs the Advantage-Value method to optimize the neural network loss function, ultimately optimizing the training process. A comparative analysis is conducted between the baseline MuZero algorithm, its A0GB algorithm-enhanced variant M0GB, and the further refined M0RV algorithm, across a spectrum of Atari and intricate board games. Notably, M0RV outperforms its predecessors in both the Lunar Lander and Breakout games, as well as in the board game Hex, under consistent steps parameters and unified reward benchmarks. The empirical findings demonstrate that the M0RV model, in comparison to the MuZero model, substantially enhances training efficacy, successfully fulfilling the objective of optimizing the training methodology.

INDEX TERMS MuZero, MCTS, game, training optimization.

I. INTRODUCTION

Board games, distinguished by their precise rulesets and objectives, are quintessential for appraising intelligent game-playing algorithm efficacy and are emblematic in artificial intelligence research. In 1995, TD-Gammon [1] pioneered the use of neural networks and self-play methodologies in backgammon, attaining parity with human intellect. Subsequently, in 1997, Deep Blue [2] demonstrated the prowess of tree search techniques by vanquishing a human champion in chess. The year 2016 marked a milestone as AlphaGo [3] amalgamated neural networks and tree search techniques to secure an unequivocal triumph in Go, a domain known for its intricate strategic depth, against human competitors. In 2017, AlphaGo Zero [4], an evolution of its predecessor AlphaGo, integrated a sophisticated self-play mechanism, thereby effortlessly surpassing the performance of the original AlphaGo. Concurrently, Alpha Zero [5] expanded these techniques to a broader spectrum of board games, consistently exceeding human performance benchmarks. In 2020, the MuZero [6] algorithm represented a significant advancement over the Alpha Zero framework by attaining superhuman performance across a spectrum of challenging and intricate strategic game domains, all achieved without necessitating an environmental dynamics model.

In summation, across diverse gaming platforms, artificial intelligence algorithms have exhibited astonishing efficacy and adaptability [7], [8], [9]. Nevertheless, optimizing algorithmic efficiency presents ongoing challenges and constraints. In recent years, many methods have been developed to improve algorithm efficiency [10], [11], [12], [13], including the continuous optimization of the training process from Soft-Z [14] to A0C [15] to A0GB [16] as proposed by scholars. Within the context of A0GB, a novel acceleration of training via simulated game data from AlphaZero was posited, yet specific enhancements for MuZero remain scant. it is imperative to investigate the utilization of extant data and enhancement of training velocity as methodologies to refine the MuZero algorithm. This study, grounded in the context

The associate editor coordinating the review of this manuscript and approving it for publication was Kalyan Koley.

of board and Atari games [17], [18], [19], seeks to maximize data utilization and quality to refine the MuZero framework. We introduce two novel contributions:

(1) An examination of A0GB's adaptability to MuZero, adapting its enhancements to suit the unique learning architecture and data format requirements of MuZero, thereby extending the applicability of A0GB.
(2) An investigation into resource allocation within MuZero's deep reinforcement learning paradigm and Monte Carlo Tree Search process, conserving and repurposing high-fidelity simulation trajectories from MCTS, enriching the data pool, and applying the Advantage-Value [20], [21] approach to amplify the predictive and decision-making acumen of the algorithm.

The subsequent sections of this document are structured as follows. Section II delineates the foundational background and literature pertinent to the evolution from Alpha Zero to MuZero models. Section III articulates the methodologies proposed in this study. Section IV elucidates the experimental design, the empirical findings, and a comprehensive analysis thereof. Conclusively, Section V synthesizes the overarching insights gleaned from this research and proposes trajectories for future inquiries.

II. BACKGROUND

A. A0GB

A0GB is an algorithm that further optimizes the training process based on AlphaZero. The optimization improvements of A0GB primarily focus on two aspects:

Firstly, A0GB combines the data generated by Monte Carlo Tree Search (MCTS) simulation with game training data to improve the efficiency of acquiring training samples, thereby accelerating the training process. Monte Carlo Tree Search expands new nodes during the path selection process and evaluates these nodes using a neural network to propagate evaluation results (such as win rate, value) from leaf nodes to the root node. Each node updates its statistical data, including visit count and accumulated value, throughout the process. The A0GB algorithm stores backup data for all nodes. Eventually, these stored data are encapsulated into neural network input format and added to the training process, expanding the training samples.

Secondly, the A0GB algorithm improves the training process of AlphaZero by using a more effective value target function trained by neural networks. The role of the value target in AlphaZero is to guide the action selection of the neural network. For the value target in AlphaZero, the effectiveness can be evaluated through Monte Carlo Tree Search to choose the optimal value target. In the value target function, the A0GB method discards the traditional approach of obtaining the average value of child nodes as the value of the root node. The value target value is modified to select the maximum value among its child nodes. The value of a child node is modified to the expected value of all leaf nodes of that child node. This helps to address the issue of overestimation or underestimation of action values caused by environmental noise and selection randomness. The value function formula contains two parts: immediate reward and long-term reward. The long-term reward is estimated using the value function. The value function formula of A0GB calculates the Q-value of an action as the average of the expected values of all leaf nodes reached during the simulation. The target value at time step t is updated based on the immediate reward and the discounted maximum Q-value of future actions, considering the discount factor and the number of long-term reward rounds. The value function formula of A0GB can be written as Equation 1 and Equation 2.

$$Q(s_t, a) = \frac{1}{n} \sum_{l \in \text{ leaf nodes of } c} v_l \tag{1}$$

$$z_t = r_t + \gamma^k * \max\left(Q(s_t, a)\right) \tag{2}$$

where the value $Q(s_t, a)$ is the sum of the expected values v_l of all its leaf nodes l, r_t represents the obtained reward, γ is the discount factor controlling the influence of long-term rewards, and k denotes the number of long-term reward rounds, z_t denotes the target value of the neural network.

To evaluate the improvement effects of different value targets on AlphaZero, A0GB compares the rewards obtained during training under different value targets. Through experimentation, it has been demonstrated that the improved value target of A0GB significantly affects the training time of the AlphaZero algorithm. In particular, A0GB's backup mechanism is particularly suitable for environments that require rapid learning and adaptation, making A0GB outperform traditional AlphaZero in performance.

B. ADVANTAGE-VALUE

The Advantage-Value algorithm is an important technique in the field of deep reinforcement lteearning, which integrates the core ideas of the Actor-Critic [24] method and improves the stability and efficiency of the training process by introducing an advantage function. In reinforcement learning tasks, an agent learns action policies by interacting with the environment, aiming to maximize long-term rewards. The Actor-Critic method, as a fundamental framework, achieves this by separating policy (Actor) and value evaluation (Critic) to maximize long-term rewards. However, during training, many issues arise. Firstly, updating the policy directly based on overall rewards may result in high variance in rewards for different actions, leading to high signal noise during the learning process. Secondly, evaluating action values in complex environments may not be precise enough, affecting policy optimization and resulting in inefficient and unstable learning of the agent. Therefore, the advantage function $A(s, a)$ aims to address these issues by reducing value variance and providing more accurate action evaluations. The core of the advantage function lies in distinguishing the advantage of each action relative to the average action value, thereby finely adjusting the policy to optimize long-term rewards.

A key problem addressed by the advantage function $A(s, a)$ is that in a given state, some actions may be better than

average, while others may be worse. Focusing solely on the overall value expectation may overlook differences in values, leading to reduced learning efficiency. The advantage function measures the improvement of taking action a in state s relative to the average value of all possible actions under the current policy by calculating the difference between action value $Q(s, a)$ and state value $V(s)$. In the algorithm, the calculation of the advantage function typically involves estimating Q values and V values. Q value represents the expected return after taking action a in state s. V value represents the expected return in state s. Since directly computing Q values and V values may be very complex, these functions are usually obtained through neural networks in practical applications. In the Advantage-Value algorithm, the formula for the advantage function is as Equation 3:

$$A(s, a) = Q(s, a) - V(s) \tag{3}$$

where $A(s, a)$ is the advantage function value of taking action a in state s. $Q(s, a)$ is the expected return after taking action a in state s. $V(s)$ is the expected return in state s.

This approach allows the algorithm to distinguish which actions are actually better than average, thus paying more attention to these actions when learning the policy. Through this mechanism, the algorithm can optimize the policy and improve long-term rewards, as well as accelerate the learning process by accurately evaluating the relative advantages of each action.

C. MuZero

1) NEURAL NETWORKS IN MuZero

MuZero integrates a Monte Carlo Tree Search (MCTS)-based planning algorithm with an intricately learned environmental model. Unlike traditional approaches that rely on direct environmental observations for input states, MuZero employs a sophisticated internal state representation to simulate transitions and rewards. It leverages a trio of networks: the representation network, dynamics network, and policy-value network, to construct and navigate dynamic environments effectively, thereby enhancing action planning.

The representation network ingests known observational data about the game state and transforms it into a latent state representation. This representation encapsulates the complex and nuanced dynamics of the game environment in a comprehensive yet abstract manner, making it suitable for downstream processing by the dynamics and policy-value networks. The transformation is mathematically represented as depicted in Equation 4.

$$S_t = h_\theta\left(O_1, \ldots, O_t\right) \tag{4}$$

where S_t is the latent state representation at time t, h_θ is the representation neteqwork parameterized by θ, and O_1 to O_t are the observed game states up to time t.

The dynamics network is designed to meticulously simulate the physical laws governing the environment. It processes the current hidden state s_t and subsequent action a_{t+1} as inputs, yielding the ensuing hidden state s_{t+1} and the immediate reward r_{t+1} as outputs. This framework allows for a precise prediction of future states and rewards based on the underlying mechanics of the environment. As depicted in Equation 5.

$$r_{t+1}, S_{t+1} = g_\theta\left(S_t, a_{t+1}\right) \tag{5}$$

The policy value network generates action policies and evaluates the current state's value, taking the current hidden state S_t as input and outputting the policy p_t and value v_t. As depicted in Equation 6.

$$p_t, v_t = f_\theta(S_t) \tag{6}$$

2) MONTE CARLO TREE SEARCH FOR MuZero

Monte Carlo Tree Search (MCTS) is an exemplary best-first search algorithm characterized by its strategic exploration of extensive state spaces [25]. Initiating from a specific game state s, the algorithm iteratively constructs a decision tree through self-simulated play, progressively refining node information with each search iteration. This process culminates in the identification of the most advantageous action strategy amidst complex state configurations. Each node within the tree embodies a distinct game state s, while branches delineate potential actions a. Furthermore, each node encapsulates critical metrics: visitation count $N(s, a)$, mean action value $Q(s, a)$, strategy probability $P(s, a)$, immediate reward $R(s, a)$, and state transitions $S(s, a)$, collectively facilitating a comprehensive and dynamic strategy formulation.

The Monte Carlo tree search (MCTS) algorithm first uses a representation network to obtain an initial hidden state, denoted as s_0, which serves as the root node of each simulation.

1) Simulation: The algorithm follows the PUCT [26] rule to traverse the tree from s_0 to a leaf node, denoted as s_l, at each time step $t = 0 \ldots\ldots l$, ultimately selecting the optimal action, denoted as a_t. As depicted in Equation 7.

$$a_t = \text{argmax}_a \left(Q(s, a) + P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)} \times \left(c_1 + \log\left(\frac{\sum_b N(s, b) + c_2 + 1}{c_2}\right)\right)\right) \tag{7}$$

Here, c_1 and c_2 are used to control the influence of the prior probability $P(s, a_i)$ relative to $Q(s, a_i)$.

2) Expansion: If the action a_t leads to a new state s_{t+1} that already exists in the search tree, the algorithm proceeds to the next iteration of the search. If s_{t+1} is not in the search tree, it is added as a new node to the tree.
3) Backup: The return R is used to update the visit count $N(s)$ and the average return $Q(s)$ for all nodes along the search path in a backward pass. The statistical data for each edge is also updated by backpropagating along the simulation path.
4) Evaluation: A neural network is used to predict the total return R along the current path, and this value

is stored in the corresponding node of the search tree. As depicted in Equation 8 and Equation 9.

$$Q\left(\mathrm{s}_{\mathrm{t}-1}, a_t\right) = \frac{N\left(\mathrm{s}_{\mathrm{t}-1}, a_t\right) \cdot Q\left(\mathrm{s}_{\mathrm{t}-1}, a_t\right)}{N\left(\mathrm{s}_{\mathrm{t}-1}, a_t\right) + 1} \tag{8}$$

$$N(s_{t-1}, a_t) = N(s_{t-1}, a_t) + 1 \tag{9}$$

MuZero combines Monte Carlo tree search (MCTS) with a neural network to perform multiple simulations on the game tree and use the network's predictions to guide more effective searches and update node information. The optimal action strategy a_t can be found in the end. By continuously updating the neural network based on the game outcomes, the efficiency and accuracy of the search are further improved. This entire process combines the advantages of MCTS and neural networks to form an efficient decision-making and optimization system.

3) TRAINING OF MuZero

The training process of MuZero involves two primary stages: self-play training and fine-tuning. During self-play training, MuZero engages in self-play games using Monte Carlo Tree Search (MCTS) and experience replay to generate training data. In the fine-tuning stage, the neural network parameters are adjusted using policy gradient and value function gradient algorithms to optimize the policy and value outputs, respectively.

During the self-play training stage, MuZero uses Monte Carlo tree search (MCTS) and experience replay algorithms [27] to engage in self-play and update network parameters based on the resulting game data. Using a representation network, raw game states are transformed into internal state representations, which are then used by the dynamics and prediction networks to predict the next state and reward, respectively. The value network is used to predict the outcome of the game. MCTS algorithm is used to explore the game tree during self-play, and decisions are made using the PUCT algorithm based on the predictions of the network and the results of the tree search. These self-play data are saved in an experience replay pool, and deep learning methods are used to train the network model for accurate prediction of the next state and reward to make better decisions.

In the fine-tuning stage, MuZero uses policy gradient algorithms to adjust the network's policy output and value function gradient algorithms to adjust the network's value output. Through fine-tuning, MuZero can adapt better to different game rules and environments, and demonstrate higher levels of game intelligence.

The loss function of the MuZero algorithm is tripartite, encompassing value loss, policy loss, and reward and simulation loss. These components are synergistically amalgamated to train the model, facilitating decision-making within gaming environments. The formula is as follows:

$$L = L_{\text{value}} + \alpha L_{\text{policy}} + \beta L_{\text{dynamics}} \tag{10}$$

$$L_{\text{value}} = (v_t - z_t)^2 \tag{11}$$

$$L_{\text{policy}} = \sum_a \pi_t\left(s_t, a\right) \log P_t\left(s_t, a\right) \tag{12}$$

$$L_{\text{dynamics}} = (u_t - r_t)^2 \tag{13}$$

Within this framework, L_{value}, L_{policy}, and $L_{dynamics}$ represent the respective loss functions for value assessment, policy formulation, and reward plus simulation dynamics. The hyperparameters α and β regulate the respective proportions of policy and reward/simulation losses. The variable v_t denotes the action value estimates derived from Monte Carlo Tree Search (MCTS) sampling, while z_t represents immediate rewards and estimated target state values. $P_t\left(s_t, a\right)$ delineates the probabilistic outcomes for action sequences as dictated by MCTS, contingent on the current and historical state matrices. Similarly, $\pi_t\left(s_t, a\right)$ specifies the probabilities associated with equivalent action sequences as allocated by the policy network, reflecting the likelihood of selecting a particular action sequence in accordance with the prevailing policy orientation. The dynamics loss, $L_{dynamics}$, is pivotal in sampling subsequent states S'. Throughout the self-play iterations, the network assimilates new transitions $\left(\mathrm{s}, \mathrm{a}, \mathrm{s}', \mathrm{r}\right)$ by sampling ensuing states and associated rewards, thereby perpetuating its training and optimization.

The overall loss function is formulated as follows:

$$L_t(\theta) = (v_t - z_t)^2 + \alpha \sum \pi_{tt}\left(s_t, a\right) \log P_t\left(s_t, a\right) + \beta \left(u_t - r_t\right)^2 + c\|\theta\|^2 \tag{14}$$

MuZero uses a model composed of neural networks and MCTS to generate complete game trajectories in the environment through self-play as training data. The network is then trained with these data updates, generating new data for further training and model updating until an optimal performance model is obtained. Through self-play, MuZero can generate data in the time allowed, solving the problem of obtaining large amounts of data in complex environments in deep reinforcement learning. By effectively updating models using these data, MuZero achieves good performance, illustrated in Figure 1. This "zero-data" training mode allows MuZero to be applied to various environments without the need for manual collection of large amounts of game data.

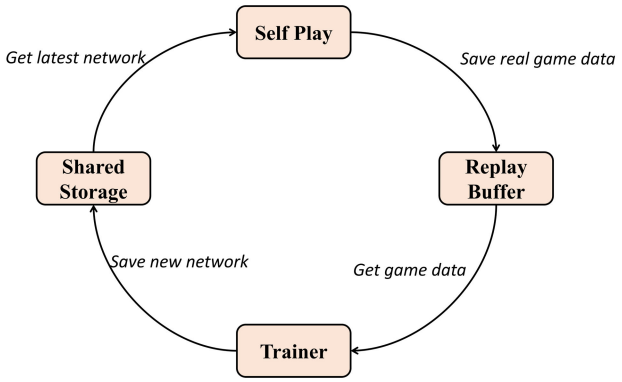


FIGURE 1. Muzero's training process outline diagram.

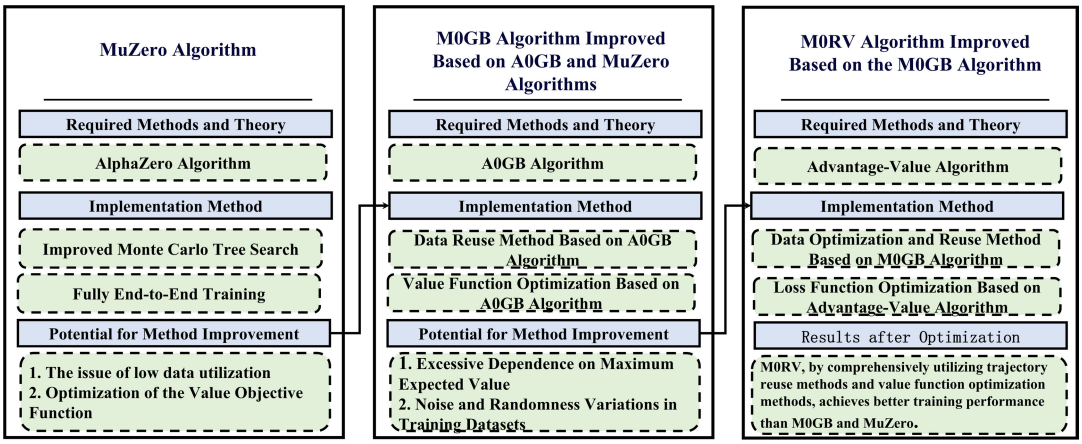


FIGURE 2. Overview diagram of research ideas.

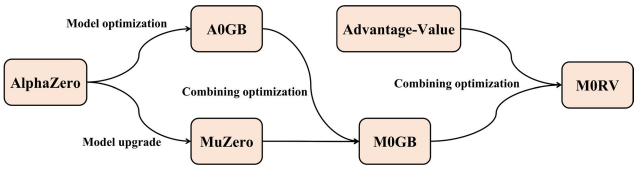


FIGURE 3. The relationship between the five AlphaZero, MuZero, A0GB, M0GB, and M0RV.

III. IMPLEMENTING MuZero ALGORITHM OPTIMIZATION

In this chapter, we delineate the intricate optimization processes integral to the MuZero algorithm. Initially, we present the M0GB method, an advanced strategic optimization construct that leverages the foundational aspects of MuZero while integrating the innovative concepts inherent in A0GB. Subsequently, we advance to the M0RV method, an augmented adaptation of M0GB that assimilates the Advantage-Value algorithm, further refining and optimizing the value function. The detailed research ideas are shown in Figure 2. The relationship between the five AlphaZero, MuZero, A0GB, M0GB, and M0RV as shown in Figure 3.

A. M0GB

In this work, we detail that MuZero's training data is derived from self-generated game trajectories during self-play, which serve to inform the neural network training. These are termed real game trajectories herein, as depicted in Figure 4. During action selection, MuZero employs Monte Carlo Tree Search (MCTS) to perform move simulations. This involves emulating opponent moves and engaging in self-play to generate what we refer to as simulated game trajectories, illustrated in Figure 5

In A0GB, the combination of Monte Carlo Tree Search (MCTS) simulated game data with game training data, and the modification of the value function, aims to enhance the convergence speed and efficiency of trajectory acquisition. This paper applies the improvement methods of A0GB to AlphaZero in the MuZero framework, forming the

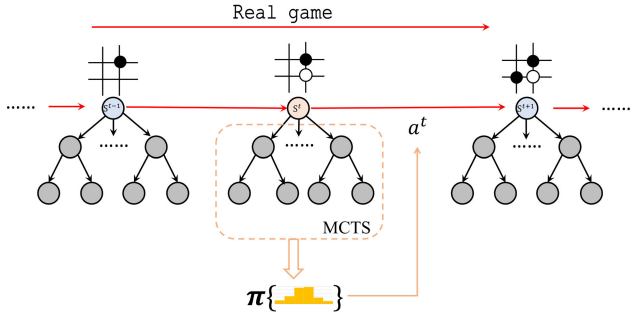


FIGURE 4. Action paths in real games.

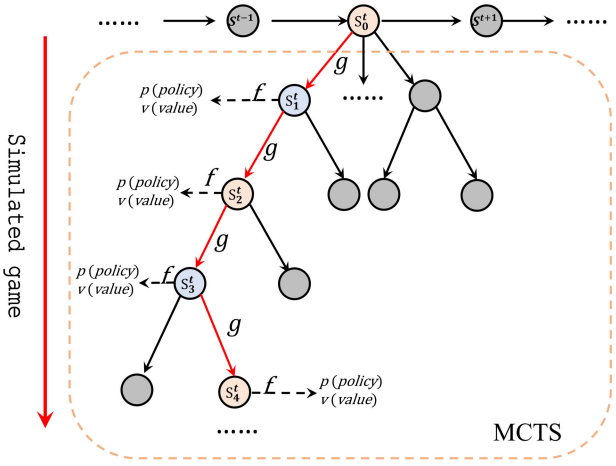


FIGURE 5. Simulated game action paths in MCTS.

M0GB model. However, in AlphaZero, for each simulated game, a specific target value is assigned to the final leaf node state based on the game outcome (win, loss, or draw) in the environment model, which guides the learning process of the algorithm and helps optimize the performance of the neural network. The learning framework of MuZero differs from AlphaZero; while AlphaZero's policy and value functions depend on the state information of the environment model,

MuZero does not require such information. In AlphaZero, the target value comes directly from the results of Monte Carlo Tree Search (MCTS), i.e., the expected returns obtained based on the state information of the environment model. In contrast, MuZero continuously optimizes network parameters through self-training in its dynamic model, which involves representation network, MCTS, policy value network, and dynamics network. Due to these differences, the A0GB algorithm applicable to AlphaZero cannot be directly applied to MuZero and requires adjustments based on MuZero's data sampling method and the structure of the value function.

Initially, we refined the data sampling methodology. Temporal sequences in actual gameplay are denoted using the superscript t, while those in simulated play are indicated with subscript N. During data sampling, the observation and extraction of simulated trajectories are critical. Within the MCTS framework, observing these simulated paths allows for the compilation of a sequence of actions, observations, and rewards. This sequence, aligned with the temporal length of simulation 'N', forms an ordered set $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_{N-1}, a_{N-1}, r_N, s_N$, where s_{N-1}, a_{N-1}, r_N, and s_N respectively represent the observed subsequent state and reward r_N, s_N after executing action a_{N-1} in state s_{N-1}. During training, this sequence is dissected into three components: the action sequence $a_0, a_1, \ldots, a_{N-1}$, the observation sequence $s_0, s_1, s_2, \ldots, s_N$, and the reward sequence $r_1, r_2, \ldots, r_N$.

Within the Monte Carlo Tree Search (MCTS) framework, the action trajectory paths for simulated games differ from those in actual games. MCTS iteratively simulates the current game, selecting and evaluating actions to eventually generate a new action O^{t+1} for the real game. During training, copied simulated paths are employed to generate data. Subsequently, these paths initiate from the neural network's latent states and progress through the policy value network, yielding predicted outcomes and historical rewards $u_0^t, \ldots\ldots, u_N^t$ as well as historical values $z_0^t, \ldots\ldots, z_N^t$. From MCTS, we derive a simulated path trajectory of length N, commencing at s_0 and culminating at s_N. Post-acquisition of the simulated path trajectory, length adjustment is necessary to align the simulated path's length with the designated training path length K. If the path length N exceeds the training path length K, we truncate the simulated path from its origin to a length of K for training data. This ensures adequate data for training purposes. Conversely, if the path length N is less than the training path length K, we discard the data.

In terms of the value function, we have adopted and adapted the A0GB framework to refine MuZero's value function, transitioning from traditional value targets to an enhanced objective derived from the maximal value ascertained at the leaf node states, as depicted in Equation 15.

$$z_t = r_t + \gamma^k \max\left(\sum_a \pi_t\left(s_t, a\right)\left(Q_t\left(s_t, a\right)\right)\right) \tag{15}$$

As elucidated by the formula, following the principles of A0GB, we have replaced the original formulation of $Q_t\left(s_t, a\right)$ with the maximum value within the action value estimate $Q_t\left(s_t, a\right)$.

By employing the A0GB approach within MuZero, it is possible to significantly enhance the accuracy and learning rate of the value function estimation. This method utilizes the output from the dynamic programming network, employing both the value function and policy as prior knowledge to guide the expansion of the search tree. It then selects the maximum value of the value function as the current value function estimate.

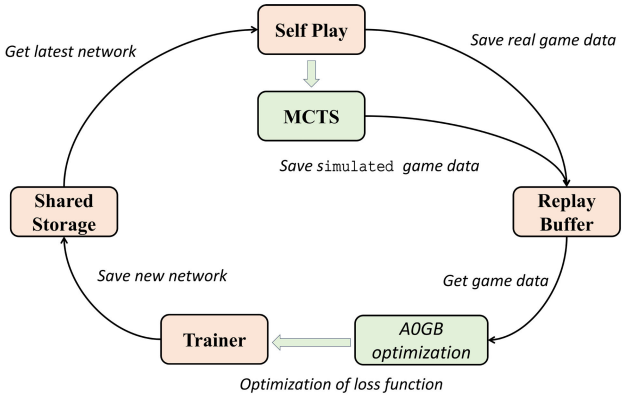


FIGURE 6. M0GB's training process outline diagram.

M0GB's training process outline diagram as shown in Figure 6. Utilizing the aforementioned techniques, we have adapted the A0GB approach for application within the MuZero framework, herein referred to as M0GB. M0GB is designed to investigate the applicability and efficacy of the A0GB algorithm specifically within the MuZero context.

B. M0RV

The M0RV algorithm represents an advancement over M0GB, featuring further refinements in trajectory reuse methodologies and loss function optimization.

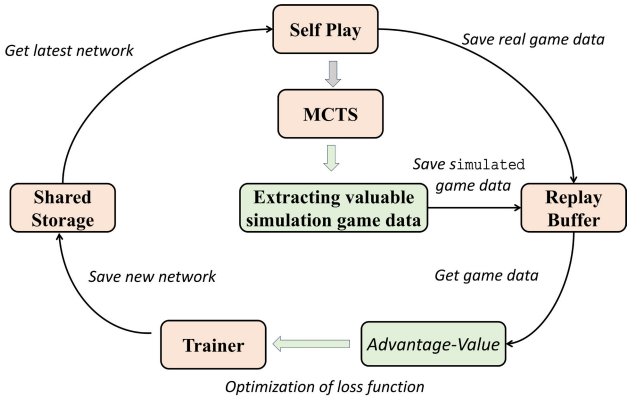


FIGURE 7. M0RV's training process outline diagram.

1) OPTIMIZATION OF TRAJECTORY REUSE METHODS

Throughout the M0GB simulation process, all game data may be subject to noise or inaccuracies, leading to unpredictable quality of data. Moreover, during trajectory backup, the trajectory data is merely truncated from the simulation's initial node to a length of K for training, not fully utilizing the potential of simulated game data. Thus, we have augmented the M0GB framework by introducing a simulation trajectory evaluation function. This function computes the total returns, action counts, value loss, and policy loss of a trajectory. It also includes tailored methods for truncating simulated game trajectories of various lengths. This enhanced version of the M0GB algorithm is referred to as M0RV in this work.

M0RV's training process outline diagram as shown in Figure 7. The M0RV method's trajectory reuse process is delineated into three primary steps:

1) Appropriately padding or truncating simulated trajectories of varying lengths to standardize their length to K.
2) Assessing the quality of paths within the simulation tree to extract valuable simulated trajectories.
3) Integrating the processed simulated trajectory data with actual game trajectories to serve as training data for the neural network.

Initially, upon acquiring the simulated path trajectory, it is imperative to conduct length adjustment to ensure that the length of the simulated trajectory aligns with the prescribed training trajectory length K.

1) When the path length N exceeds the training trajectory length K, we truncate the simulated trajectory from the initial node to a length of K for use as training data. This ensures sufficient data for training. As shown in Figure 8 (a).
2) When the path length N is less than the training trajectory length K, to guarantee a minimum greedy path length of K, we propose three methods to address the extension of the path, each with its advantages and disadvantages: Method 1:
    a) Adopt the MCTS Roll Out strategy to extend the trajectory from the leaf node s_N until the path length reaches K. As shown in Figure 8 (b). While this method ensures the length of the trajectory, it does not guarantee the quality of the trajectory produced during the MCTS Roll Out process, presenting drawbacks for practical training application.
    b) Initiate additional simulations from the leaf node s_N to ensure a minimum greedy path length of K. As shown in Figure 8 (c) This approach allows for more simulations to guarantee both the length of the training trajectory K and the quality of the trajectory, albeit with a risk of premature game termination within the final path K.
    c) continuing to execute more simulations at the leaf node would result in excessively high costs, and there is a possibility of premature game termination within the final path of length K. As shown in Figure 8 (d). To address this situation, we can backtrack the already executed actions and add them as a head path to the simulation path to ensure the minimum length of K for the greedy path, and use K_1 to represent the true path extended to. As depicted in Equation 16.

$$K = K_1 + N \tag{16}$$

This method can obtain more simulations to ensure the minimum length of the greedy path, but there is a possibility that the final path length might be less than $K_1 + N < K$. Consequently, to synergize the advantages of both approaches, we amalgamate Methods 2 and 3 to address the issues presented in scenario 2

Subsequently, the total reward metric reflects the cumulative rewards obtained within a trajectory, serving as a crucial indicator for assessing trajectory quality. The number of steps penalty takes into account the trajectory length, promoting models to achieve favorable outcomes more efficiently. The value loss represents the loss function of the value network, aiding in the model's improved learning of state value functions. Similarly, policy loss corresponds to the policy network's loss function, facilitating the model's acquisition of superior strategies. As depicted in Equation 17.

$$score = total_{reward} - \alpha num_{steps} - \beta value_{loss} - \gamma policy_{loss} \tag{17}$$

A comprehensive score is computed by integrating these metrics, reflecting the overall quality of the trajectory as illustrated in Equation 7 [28], [29], [30]. This aggregate score guides the data sampling process, prioritizing trajectories with higher scores to enhance the quality of training data. This approach effectively balances the trajectory's rewards, length, and learning efficiency, offering a more nuanced method for assessing and selecting training data. Consequently, it contributes to the improvement of training data quality throughout the algorithm's training process. The total reward metric reflects the cumulative rewards obtained within a trajectory, serving as a crucial indicator for assessing trajectory quality. The number of steps penalty takes into account the trajectory length, promoting models to achieve favorable outcomes more efficiently. The value loss represents the loss function of the value network, aiding in the model's improved learning of state value functions. Similarly, policy loss corresponds to the policy network's loss function, facilitating the model's acquisition of superior strategies.

Ultimately, we integrate the processed simulated trajectory data with actual game trajectories to form a composite dataset for neural network training. This integration culminates in the training of the M0RV by amalgamating the trajectories from real and simulated games, enhancing the overall training process.

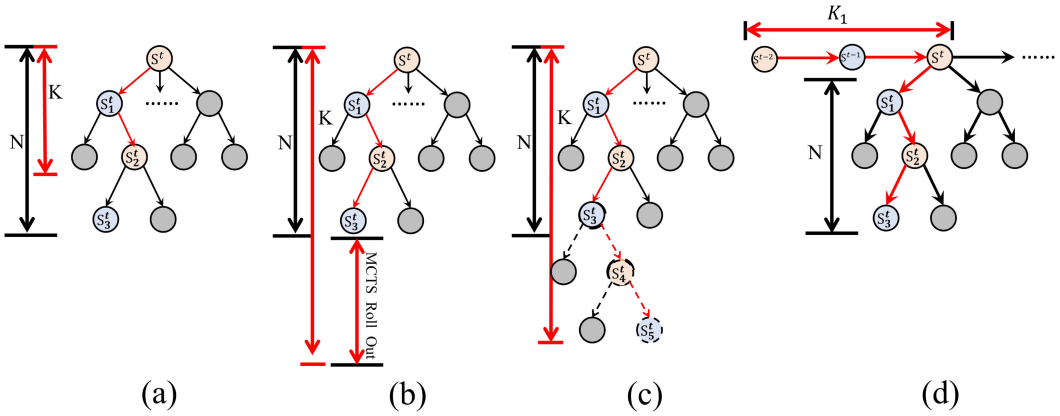


FIGURE 8. Four scenarios that may be encountered.

2) OPTIMIZATION OF THE VALUE FUNCTION

In MuZero, the objective of the value function is to update it using a weighted sum of the average return value and the action value based on the current policy. In section III-A, the improvement of MuZero's value target function is proposed based on the idea of A0GB. However, modifying the traditional value target to the maximum expected value obtained from the leaf node state may lead to the algorithm overly focusing on actions currently evaluated as optimal, while neglecting other potentially valuable actions or strategies, thus missing out on strategies with greater long-term gains. To address these drawbacks, the Advantage-Value method is adopted to enhance MuZero's value target function on this basis, thereby improving its loss function. The Advantage-Value method aims to better capture the advantage of executing a certain action relative to the average level, rather than greedily selecting the action with the maximum value as in A0GB.

In the M0RV algorithm, the Advantage-Value method is employed to optimize the policy function by maximizing the expected advantage, denoted as $A(s, a)$. In this context, $Q_t(s_t, a) - V(s_t)$ represents the advantage of taking action a relative to the average performance. The value function $V(s_t)$ is utilized to assess the expected return for all possible actions in state s_t, as depicted in equation 18.

$$V(s_t) = E[G_t | s_t = s] \tag{18}$$

In this context, $V(s_t)$ represents the expected return in state s_t. G_t denotes the cumulative return starting from time step t.

By combining the estimated value $Q_t(s_t, a)$ obtained through MCTS search with the calculated expected return $V(s_t)$, the advantage function $A(s, a)$ can be derived, as depicted in equation 19.

$$A(s, a) = Q_t(s_t, a) - V(s_t) \tag{19}$$

Afterward, multiplying $\pi_t(s_t, a)$ by the advantage $A(s, a)$, which represents the average value for all possible actions in that state, effectively weights the probability for each action. This adjustment aims to prioritize more advantageous actions by making them more likely to be chosen. The optimization of the policy loss function and value function is performed using the advantage function $A(s, a)$, as depicted in equation 20.

$$L_{policy} = \sum_a \pi_t(s_t, a) A(s, a) log P_t(s_t, a) \tag{20}$$

The objective of the policy loss function is to guide policy learning by maximizing the expected advantage. By multiplying action probabilities by their advantages and taking the expectation, the loss function encourages the model to prefer actions with higher advantages. This encourages the policy function to more effectively explore and exploit potential advantages in the environment.

Through this approach, the value target function z_t is updated accordingly, as depicted in equation 21.

$$z_t = r_t + \gamma^k \left(\sum_a \pi_t(s_t, a) A(s, a) log P_t(s_t, a) \right) \tag{21}$$

where z_t is the target value, r_t is the immediate reward obtained by a function that converts state s_t into a feature vector, γ is the discount factor, and $Q_\theta(s_t, a)$ represents the action value estimation when taking action a in state s_t. The actual action value of taking action a in state s_t, denoted as $\pi(a|s_t)$, represents the probability predicted by the policy network for action a in state s_t.

The total value loss function can be written in the following form:

$$L_{\text{value}} = \left(v_t - r_t - \gamma^k \max \times \left(\sum_a \pi_t(s_t, a) A(s, a) log P_t(s_t, a) \right) \right)^2 \tag{22}$$

In M0GB, there exists a problem of over-optimization, where excessive reliance on maximum expected value may

result in poor performance when faced with unknown or low-probability events. By adopting the Advantage-Value method, the variance in value function estimation can be reduced, thereby addressing these shortcomings and achieving a more comprehensive and balanced decision-making process. The M0RV algorithm optimizes both the policy function and the value function. This approach fully utilizes the advantage function to guide policy learning, while also more accurately estimating the value of actions through the value function, thereby achieving effective optimization of the model.

IV. EXPERIMENTAL EVALUATION

This paper contrasts the performance of agents using the original MuZero algorithm, the M0GB algorithm, and the M0RV algorithm in various game environments. The average reward serves as a signal of the learning efficiency of the current agent's algorithm during the training process, with actions selected based on this average reward. Gradual convergence of the average reward indicates an enhancement in the agent's ability to learn how to complete tasks more effectively.

A. GAME ENVIRONMENT

To validate the effectiveness of the methodologies, we conducted experiments with three selected games across board games and Atari gaming environments: For Atari, we chose the Lunar Lander and Breakout games [31], which require precise control over magnitude and angle, characterizing a relatively small state space. In contrast, for classical board games, we selected Hex, known for its larger state space, demanding long-term planning and strategic thinking. Our primary focus lies in quantifying the rewards attained by the algorithms and evaluating their effectiveness based on the rate of convergence.

1) LUNAR LANDER

The lunar lander game is a classic reinforcement learning environment within OpenAI Gym. It simulates the process of a lunar lander landing on the surface of the moon. The objective of the agent is to learn a policy that enables the lunar lander to safely land. This poses a complex control and optimization problem considering eight factors, including velocity, angles, and others. The agent in the game can open or close its engine, resulting in four types of actions: do nothing, fire left orientation engine, fire main engine, and fire right orientation engine. The state space consists of an 8-dimensional vector, representing the horizontal and vertical coordinates (x and y) of the lander, the linear velocities v(x) and v(y), the angle θ, the angular velocity ω, and two Boolean values indicating whether each leg is in contact with the ground. The reward function for the lunar lander takes into account factors such as energy consumption, deviation from the desired trajectory, and touchdown position. Specifically, the rewards are as follows: Deduct 100 points if the lunar lander crashes. Deduct 10 points if the lunar lander flies out of the designated landing zone. Earn between 100 to 140 points depending on the distance of the lunar lander's touchdown position from the center of the landing platform. Earn 100 points if the lunar lander comes to a complete stop. Earn 10 points for each leg in contact with the ground. Deduct 0.03 points for each frame the side engine is activated. Deduct 0.3 points for each frame the main engine is activated. This comprehensive reward structure encapsulates the complexities of the lunar landing task, providing incentives for the agent to learn a successful landing strategy while considering various operational constraints. As shown in Figure 9 (a).

2) BREAKOUT

Breakout is a game where a paddle is moved to rebound a ball against a wall of bricks at the top of the screen, destroying bricks to garner rewards. The player controls the direction and angle of the ball to break the wall. The action states include launching the ball initially and moving the paddle left or right. As shown in Figure 9 (b). In the game, players score varying points by hitting bricks of different colors: red and orange bricks yield 7 points, yellow and green 4 points, light green and blue 1 point, with a total of 864 points available. This environment challenges the player's precision and strategy in ball control and angle calculation to maximize the score.

3) HEX GAME

The International Computer Games Association (ICGA) organizes the Computer Olympics, which includes the game of Hex. A typical Hex board consists of 11×11 hexagonal cells, with the top and bottom boundaries colored red and the left and right boundaries colored blue. The red coordinates (A-K) represent the horizontal range, while the blue coordinates (1)-(11) represent the vertical range. The game pieces are circular and come in two colors, either red and blue or black and white. Each player takes control of one color of pieces. As shown in Figure 9 (c).

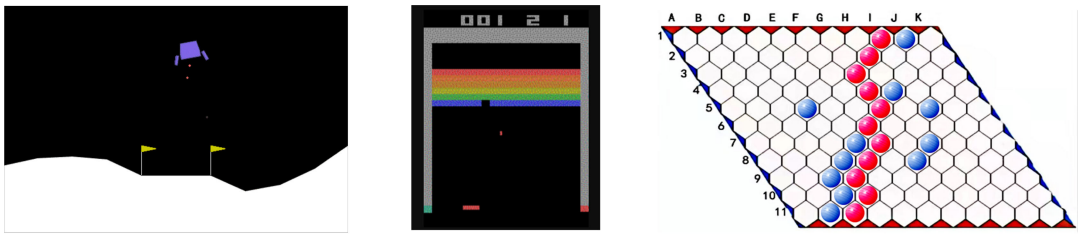
The rules of Hex are as follows:

1) After the start of the game, players take turns placing a single piece on the board, with each piece occupying a hexagonal cell.
2) Two adjacent pieces of the same color are considered connected to each other.
3) The first player to connect their two opposite boundaries with pieces of the same color is declared the winner.
4) Draws are not possible in Hex.

B. EXPERIMENTAL PROCEDURE AND RESULTS

In this section, we first introduce the foundational setup for experimental training and then proceed to present the final results.

The lunar lander, brick breaker, and hex chess games are comprehensively controlled during the training process by setting training parameters. The maximum number of moves for each game is set to 700, 2500, and 121, respectively. Failure to complete the game within the specified number of moves results in a game over and the end of the round.



(a) Lunar Lander (b) Breakout (c) HEX

FIGURE 9. Three kinds of game interface.

The number of self-play games retained in the data pool for each game type is 5000, 5000, and 10000, respectively. During each neural network iteration, 2000, 2000, and 10000 self-play iterations are executed, and 30, 100, and 600 Monte Carlo Tree Search (MCTS) simulations are performed for each decision during training. The reward discount rate is uniformly set to 0.99. The initial learning rate is set to 0.05 and gradually reaches a constant learning rate as training progresses. For training in hex chess, the Elo formula's importance coefficient K is set to 1, with initial Elo ratings set to 0. Other parameter settings are consistent across all models and are not reiterated here. Considering the difficulty of the games, limitations of computational resources, and expectations regarding the effectiveness of model training, specific game move limits, iteration counts, and MCTS simulation numbers are chosen to ensure comprehensive and systematic control during the training of lunar lander, brick breaker, and hex chess games. Setting a maximum number of moves helps prevent infinite loops in the games, while selecting specific iteration and MCTS simulation counts balances model performance and computational resource consumption.

The training results for the lunar lander game for M0RV, M0GB, and MuZero are depicted in Figure 10(a), while the training results for the brick breaker game are shown in Figure 10(b). Performing a third-order polynomial fitting on the results of the lunar lander and brick breaker games enables a clearer observation of the trend of reward acquisition for each algorithm. From the results of the third-order polynomial fitting, it is evident that during the initial stages of training across all game types, M0RV achieves higher rewards compared to M0GB and MuZero. As training progresses into the middle stages, M0RV maintains the reward gap. Ultimately, M0RV outperforms both M0GB and MuZero in terms of reward acquisition.

For the Hex game, we employed the Elo rating system for an effective evaluation of the training outcomes. The Elo rating system is a widely used method in chess and other competitive games to calculate players' relative skill levels. Throughout the training process, at every 1000th training iteration, the three agents under training were pitted against an already trained AlphaZero Hex program to derive their Elo scores. The specific results are depicted in Figure 11.

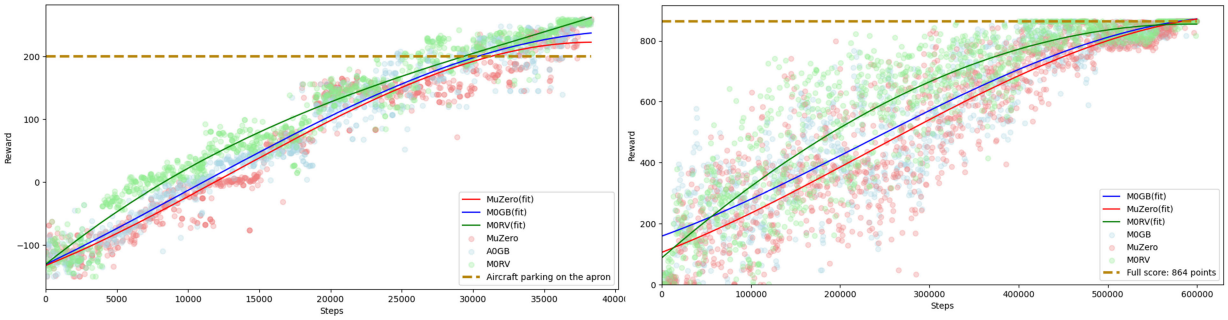
C. ANALYSIS OF RESULTS

The experimental results are shown in Table 1.

In the lunar lander game training, when the player's reward reaches 200, it can be considered that the spacecraft has completed the basic landing task. From the reward fitting curves, the order of reaching 200 during the training process is M0RV, M0GB, and MuZero. In terms of reward acquisition, the overall training average rewards for M0RV, M0GB, and MuZero are 99.14, 79.00, and 71.32, respectively. These data indicate that M0RV learns faster than the M0GB algorithm in the same training setup. The mean absolute errors (MAE) of the fitting curves for M0RV, M0GB, and MuZero are 18.12, 17.43, and 20.06, respectively. M0GB has the lowest MAE, with a difference of only 0.79 between M0RV and M0GB. In games with relatively small state spaces, the stability gap is smaller, and M0RV's stability performance meets expectations.

In the brick breaker game training, when the player's score reaches 864, it can be considered that the agent has completed the brick breaker task. From the reward fitting curves, the models' order of reaching a score of 864 during the training process is M0RV, M0GB, and MuZero. Regarding reward acquisition, the overall training average rewards for M0RV, M0GB, and MuZero are 553.00, 495.23, and 459.30, respectively. These data indicate that M0RV learns faster than the M0GB algorithm in the same training setup. The MAE of the fitting curves for M0RV, M0GB, and MuZero are 84.12, 94.48, and 97.61, respectively. M0RV has the lowest MAE, with a difference of 10.36 between M0RV and M0GB. M0RV exhibits superior stability performance.

Based on the outcomes observed in the Hex game, the final Elo ratings stand as follows: MuZero exhibits a median Elo rating of -451.28, with a concluding Elo rating of 407.67. Meanwhile, M0GB showcases a median Elo rating of -35.46, culminating in a final Elo rating of 416.37. On the other hand, M0RV demonstrates a median Elo rating of 51.37, concluding with the highest final Elo rating of 433.31 among the three algorithms. Throughout the initial matches, due to insufficient training, none of the algorithms managed to surpass the baseline program, leading to a consistent decline in scores. However, MuZero achieved its first victory against the baseline program after 589,486 training steps, maintaining a streak of consecutive victories after 757,401



(a) Lunar Lander (b) Breakout

FIGURE 10. Training results of Lunarlander and Breakout.

TABLE 1. Results for the several games tested.

Model	Hex		Lunar Lander		Breakout	
	Median(Elo)	Final score(Elo)	Average reward	MAE	Average reward	MAE
MuZero	-451.29	407.67	71.32	20.06	459.30	97.61
M0GB	-35.46	416.37	79.00	17.43	495.23	94.48
MuMR	51.37	433.31	99.14	18.12	553.00	84.12

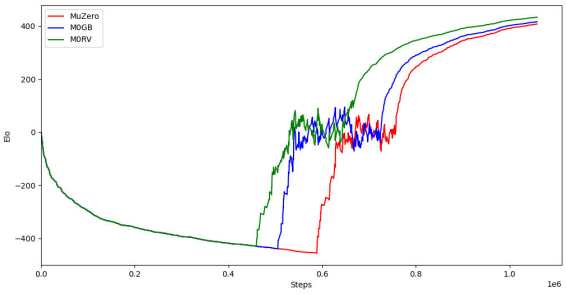


FIGURE 11. Training results of HEX.

training steps. Similarly, M0GB secured its initial victory against the baseline program after 506,108 training steps and maintained its winning streak after 724,205 training steps. Likewise, M0RV celebrated its first triumph against the baseline program after 459,479 training steps and continued to dominate after 642,781 training steps. Subsequently, M0RV's prowess surpassed that of the baseline program. Thus, ultimately, M0RV emerged with the highest Elo rating among the three algorithms.

The M0RV algorithm enhances the evaluation of simulated trajectories by introducing a composite evaluation function. This mechanism optimizes the data sampling process by ensuring the prioritization of high-quality trajectories. It addresses the instability in model learning caused by the noise and randomness disparities in the training dataset of M0GB. The composite evaluation function not only improves the quality of training data but also balances the rewards, lengths, and learning efficiency of trajectories. Additionally, the M0RV algorithm employs the Advantage-Value method to optimize the value function of the neural network loss function, mitigating the problem of over-optimization. Over-reliance on the maximum expected value can lead to poor performance when facing unknown, low-probability events, and complex action spaces. Through trajectory reuse optimization and value function improvement, the M0RV algorithm effectively enhances model training speed. Experimental results demonstrate superior performance of the M0RV algorithm over traditional methods in various simulated gaming environments, especially in terms of Elo rating improvement and reward convergence speed.

Overall, the effectiveness of the M0RV algorithm surpasses that of MuZero and M0GB algorithms. This is mainly attributed to the reuse of high-quality MCTS trajectories and optimized value functions. It provides richer and higher-quality training data for neural networks while accelerating the model convergence process. In contrast, MuZero relies solely on self-play-generated data, resulting in slower convergence. The experiments fully demonstrate that reusing high-quality training trajectories generated by Monte Carlo Tree Search (MCTS) during the training process, M0RV integrates the strengths of MuZero and M0GB and further optimizes the value function using the Advantage-Value method, achieving the goal of optimizing the training process and accelerating training. The introduction of the M0RV algorithm effectively addresses the issues present in the M0GB algorithm, improving both the quality and utilization efficiency of training data and optimizing the value function, thereby achieving significant performance improvements in multiple games. These improvements showcase the potential of deep reinforcement learning in addressing complex tasks.

The current study focuses on a limited number of games, including Lunar Lander, Breakout, and Hex, to validate

the effectiveness of the proposed M0RV algorithm. While these games cover a range of complexities and state space sizes, further evaluations on a wider array of games with varying characteristics are necessary to fully understand the generalizability of the M0RV model. To extend the study, future work should evaluate the M0RV algorithm on a broader range of games with varying complexity and state space sizes. This will test its robustness and adaptability across diverse gaming environments. Additionally, exploring real-world applications, such as in intelligent transportation, smart manufacturing, and healthcare, could harness the model's training efficiency and decision-making capabilities in complex, real-time decision-making scenarios.

V. CONCLUSION

The development of intelligent agents capable of autonomous planning and intelligent decision-making in complex environments is a central issue in the field of artificial intelligence. The MuZero algorithm provides a new approach to this problem by enabling intelligent decision-making without prior knowledge of the environment model. However, there is still room for improvement in terms of data utilization and training efficiency. To address this issue, this paper proposes optimizations based on the MuZero algorithm, with the following specific contributions:

(1) Building upon the MuZero and A0GB algorithms, this paper introduces the M0GB algorithm. Firstly, the M0GB algorithm enhances the efficiency of acquiring game trajectory data for training by combining simulated game data and real game data within Monte Carlo Tree Search (MCTS). Secondly, it improves the value objective function in MuZero, thereby enhancing the value loss function. This is achieved by transitioning the value objective function from the traditional average computation of value targets to the maximum value target obtained from leaf node states, thus addressing issues arising from environmental noise and the randomness of action selection causing overestimation or underestimation of action values. The M0GB algorithm aims to explore the applicability of the A0GB algorithm in MuZero and its optimization effects.

(2) Building upon the M0GB algorithm and the Advantage-Value algorithm, this paper proposes the M0RV algorithm. Firstly, addressing the issue of the quality of simulated game trajectory data in M0GB, the M0RV algorithm presents a more comprehensive method for truncating and supplementing simulated game trajectories when dealing with their lengths. Before incorporating the training data, it evaluates the quality of trajectories using a composite evaluation function that considers factors such as total return, steps, value loss, and policy loss. These two methods for ensuring data quality effectively enhance the quality and utilization efficiency of training data. Secondly, addressing the issue of over-optimization of the value objective function in M0GB, the M0RV algorithm utilizes the Advantage-Value algorithm to optimize the loss function in M0GB. By introducing an advantage function, the optimized loss function effectively captures the advantage of current actions relative to the average level.

To validate the effectiveness of the proposed methods, this paper compares the training performance of the original MuZero algorithm, the improved M0GB algorithm, and the further optimized M0RV algorithm on Atari games and complex board games. The M0RV algorithm demonstrates superior improvement rates in overall reward values for Lunar lander and Breakout games, as well as higher Elo score improvement rates for HEX games compared to the MuZero algorithm and M0GB algorithm. Experimental results indicate that the M0RV model significantly enhances training efficiency compared to the MuZero model, achieving the goal of optimizing the training process.

The M0RV model, through its innovative approaches to strategic data reuse and value function refinement, presents a substantial advancement over existing reinforcement learning algorithms. The practical implications of this model in real-world scenarios, such as intelligent transportation, smart manufacturing, healthcare, energy management, and financial markets, highlight its potential to revolutionize various industries. As artificial intelligence technologies continue to evolve, the M0RV model and its derivatives will undoubtedly play a pivotal role in addressing complex decision-making problems across diverse application domains.

REFERENCES

[1] G. Tesauro, "TD-Gammon: A self-teaching backgammon program," in *Applications of Neural Networks*. MA, USA: Springer, 1995, pp. 267–285.
[2] M. Campbell, A. J. Hoane Jr., and F. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, nos. 1–2, pp. 57–83, 2002.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[4] D. Silver, H. Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris, "The predictron: End-to-end learning and planning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3191–3199.
[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.
[6] 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, no. 7839, pp. 604–609, Dec. 2020.
[7] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, pp. 1–74, Mar. 2021.
[8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, and P. Georgiev, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.

[9] S. Y. Lee, C. Sungik, and S.-Y. Chung, "Sample-efficient deep reinforcement learning via episodic backward update," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–10.
[10] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, "Mastering Atari games with limited data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 25476–25488.
[11] T. K. Gilbert, S. Dean, N. Lambert, T. Zick, and A. Snoswell, "Reward reports for reinforcement learning," in *Proc. AAAI/ACM Conf. AI, Ethics, Society*, 2023, pp. 84–130.
[12] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and planning in complex action spaces," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 4476–4486.
[13] J. Hu, S. Hu, and S. W. Liao, "Policy regularization via noisy advantage values for cooperative multi-agent actor-critic methods," *Int. Found. Auton. Agents Multiagent Syst.*, 2021.
[14] F. Carlsson and J. Öhman, "Alphazero to alpha hero: A pre-study on additional tree sampling within self-play reinforcement learning," School Elect. Eng. Comput. Sci. (EECS), 2019.
[15] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0C: Alpha zero in continuous action space," 2018, *arXiv:1805.09613*.
[16] D. Willemsen, H. Baier, and M. Kaisers, "Value targets in off-policy AlphaZero: A new greedy backup," *Neural Comput. Appl.*, vol. 34, no. 3, pp. 1801–1814, Feb. 2022.
[17] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Trans. Games*, vol. 11, no. 3, pp. 195–214, Sep. 2019.
[18] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Trans. Games*, vol. 12, no. 1, pp. 1–20, Mar. 2020.
[19] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, and S. Levine, "Model-based reinforcement learning for Atari," 2019, *arXiv:1903.00374*.
[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[22] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.* Springer, 2006, pp. 282–293.
[23] Y.-C. Liu and Y. Tsuruoka, "Adapting improved upper confidence bounds for Monte-Carlo tree search," in *Advances in Computer Games*. Berlin, Germany: Springer, 2015, pp. 53–64.
[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[25] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Proc. Int. Conf. Comput. Games*. Springer, 2006, pp. 72–83.
[26] D. Auger, A. Couetoux, and O. Teytaud, "Continuous upper confidence trees with polynomial exploration–consistency," in *Proc. Eur. Conf.*, Prague, Czech Republic. Springer, Sep. 2013, pp. 194–209.
[27] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3061–3071.
[28] S. Majumdar and S. Chatterjee, "Feature selection using e-values," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 14753–14773.
[29] J. Stallrich, K. Allen-Moyer, and B. Jones, "D- and A-optimal screening designs," *Technometrics*, vol. 65, no. 4, pp. 492–501, 2023.
[30] W. Al Jurdi, J. B. Abdo, J. Demerjian, and A. Makhoul, "Group validation in recommender systems: Framework for multi-layer performance evaluation," 2022, *arXiv:2207.09320*.
[31] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, Jun. 2013.

XUEJIAN CHEN received the bachelor's degree in computer science and technology from Liaoning Petrochemical University, in 2021, where he is currently pursuing the master's degree in computer science.

From 2017 to 2024, his research interests have included deep reinforcement learning and machine gaming. He has published three papers and applied for three software copyrights. He has also been responsible for a national-level undergraduate innovation and entrepreneurship project.

Mr. Chen awards and honors include Three National Scholarships for undergraduates, Two Graduate Student Scholarships, and being recognized as an Outstanding Graduate of Liaoning Province, China, in 2021.

YANG CAO received the Ph.D. degree in agricultural environment and energy engineering from Inner Mongolia Agricultural University, in 2010.

From 2010 to 2021, she worked as a Lecturer at Liaoning Petrochemical University, where she has been appointed as an Associate Professor, since 2021. She has authored two books and more than 20 articles. Her research interests include machine gaming and applications in remote sensing image processing. She is a member of the Machine Gaming Committee of the Artificial Intelligence Society and also an Expert of the Smart Emergency Management Professional Committee of China Electronics Chamber of Commerce.

HONGFEI YU received the M.S. and Ph.D. degrees in applied mathematics and computer application from Northeastern University, Shenyang, China, in 2008 and 2015, respectively. Her research interests include computer vision, deep learning, image processing, and automatic driving.

CAIHUA SUN received the bachelor's degree in computer science and technology from Liaoning Petrochemical University, in 2021, where he is currently pursuing the master's degree in computer science. His research interests include remote sensing image processing, computer vision, and machine gaming.