Hexagon and Reinforcement: Unveiling Strategic Mastery with Advanced Reinforcment Learning

Langela Regincos Jan
FH Technikum Wien, Vienna, Austria
ai23m057@technikum-wien.at

Marcon Elias
FH Technikum Wien, Vienna, Austria
ai23m020@technikum-wien.at

***Abstract*—This paper explores advanced Reinforcement Learning (RL) techniques, specifically Proximal Policy Optimization (PPO) and Deep Q-Learning (DQL), applied to the strategic board game Hex. Initial comparisons between PPO and DQL highlight DQL's superiority after PPO fails to consistently outperform random agents. DQL agents are examined across varying network architectures—fully connected and convolutional—with results demonstrating significant performance disparities. Convolutional layers, particularly with 2 hidden layers of size 256, prove effective in enhancing decision-making by extracting crucial spatial features from the hex board. The study underscores the importance of architectural simplicity and early training initiation in achieving robust performance, with deeper networks showing slower convergence. Practical challenges such as state copy strategies and episode mirroring delay are also addressed, emphasizing their impact on agent learning and performance. Recommendations include further optimizing convolutional architectures tailored for the Hex game and addressing training methodologies to enhance agent robustness in strategic decision-making environments.**

***Index Terms*—Deep Q-Learning, Proximal Policy Optimization, Hex Game, Actor-Critic, Reinforcement Learning, Deep Learning**

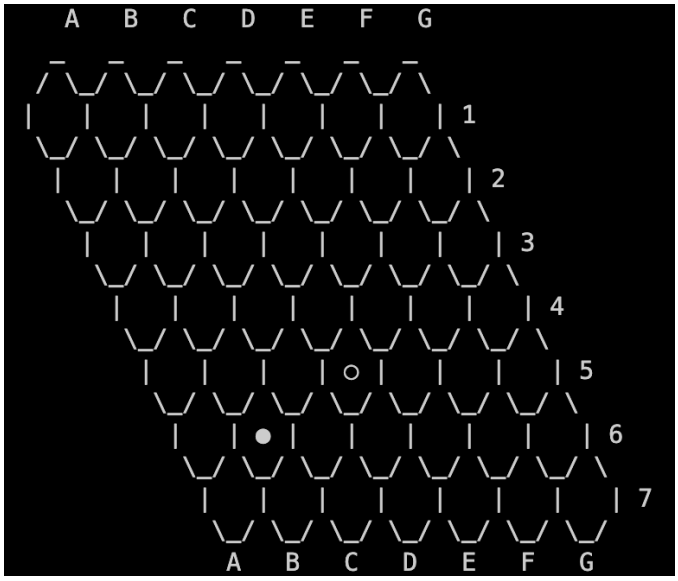

Fig. 1. Example of a 7x7 Hex grid showing players placing stones.

I. Introduction

This paper explores the application of advanced Reinforcement Learning (RL) techniques, specifically Proximal Policy Optimization (PPO) and Deep Q-Learning (DQL), within the context of the board game Hex. Hex is a strategic board game played on a hexagonal grid, commonly featuring board sizes that range between 11x11 and 19x19 cells in traditional settings. However, this study specifically concentrates on a 7x7 grid configuration. Participants alternate placing coloured stones to establish a continuous connection between their respective sides of the board—typically, the player aiming to connect the top and bottom edges plays with black stones, while the player aiming to connect the left and right edges plays with white stones. The primary objective entails forming an uninterrupted chain of stones spanning from one designated side to its opposite, achieved through horizontal or diagonal placement strategies. An illustrative example of the 7x7 Hex grid and players placing stones is depicted in Figure 1.

The primary motivation behind this study is to investigate how these RL algorithms can be effectively adapted to master Hex, akin to achievements in games like Go [1] but with unique strategic intricacies. The scope of this paper encompasses a comprehensive review of existing literature on RL methods applied to strategic board games, focusing particularly on their adaptation and performance in Hex scenarios. The paper adheres to a structured approach.

Section 2 introduces foundational principles and related work in RL, specifically focusing on Hex.

Section 3 outlines key methods used for creating the agents. Subsection 3A introduces the Hex Engine, critical for simulating gameplay scenarios. It details the setup and operational frameworks essential for Artificial Intelligence (AI) agent development and evaluation in Hex. Subsection 3B delves into the Agent Class, facilitating AI agent interactions. Discussions include model persistence, decision-making processes, and real-time board visualization crucial for iterative training and evaluation. Subsections 3C and 3D focus on adapting and refining PPO and DQL algorithms. Each explores neural network integration, training methods, and metrics for assessing AI agent effectiveness. Sections 3E and 3F detail comprehensive training methodologies. 3E iteratively refines agents through "Self Play from Random," "Tournament," and "Dojo" phases, ensuring robust strategy development. 3F progresses from PPO proof of concept on a 4x4 board to custom DQL on a

7x7 board, addressing challenges like colourblind training and integrating convolutional layers for enhanced performance in Hex AI development.

Section 4 introduces evaluation metrics crucial for assessing RL agent performance in Hex. These metrics capture strategic progress and competitive prowess across diverse game scenarios, illuminating the agents' efficacy in mastering Hex. Subsequent sections delve into detailed performance analyses of PPO and DQL agents. Section 4B focuses on PPO, highlighting its training outcomes and challenges encountered during evaluation. Despite initial promise, PPO agents exhibit inconsistencies in outperforming random agents, leading to their exclusion from the final evaluation due to insufficient robustness in competitive scenarios. In contrast, Section 4C centres on DQL, categorizing agents based on architectural configurations and evaluating their performance. Agents utilizing convolutional layers and 2 hidden layers consistently demonstrate superior decision-making capabilities compared to alternative configurations. This section explores the implications of architectural complexity and training duration, emphasizing early training initiation and extended periods to optimize performance for complex neural network architectures.

Section 5 transitions into a detailed discussion of factors influencing DQL agent performance in Hex. It delves into critical considerations such as state copy implementations, episode mirroring delay, and the impact of architectural complexity, offering insights into how these factors shape agent learning and strategic adaptation in gameplay.

Section 6 synthesizes findings into a conclusive overview of the study. It discusses the implications of results, identifies avenues for future research, and underscores the broader significance of advancing AI capabilities in strategic decision-making domains through the lens of Hex and similar complex board games.

The source code is publicly available at this Github Repository.[1]

II. Related Work

RL demonstrates advancements, particularly with developing sophisticated algorithms capable of learning to play games such as chess, shogi, and Go [1], [2]. This section reviews pertinent literature related to this work, focusing on RL methods applied to the game of Hex and the application of PPO and DQL in strategic board games like Hex.

A. Reinforcement Learning in Hex

Hex poses challenges for RL due to its large state and action spaces and the requirement for strategic planning. Early approaches to solving Hex involved heuristic-based methods and search algorithms. Anshelevich [3] utilizes a combinatorial game theory approach to develop a strong Hex-playing program. Subsequent efforts leveraged Monte Carlo Tree Search (MCTS), which proved effective due to its balance of exploration and exploitation [4].

[1]https://github.com/janpilu/HEX-REIL/tree/main

Advancements in the field focus on incorporating deep learning techniques into Hex-playing agents. The combination of deep neural networks with Monte Carlo Tree Search (MCTS) in the game of Go serves as an inspiration for similar approaches in Hex. The Hex research community applies Convolutional Neural Networks (CNNs) to approximate value functions and policies, significantly improving the strength of Hex agents [5].

B. Proximal Policy Optimization

PPO represents a state-of-the-art policy gradient method introduced by Schulman et al. [6]. PPO enhances the stability and reliability of policy gradient methods by enforcing a trust region constraint on policy updates through a clipped objective function, limiting the extent of each policy update to prevent destabilizing changes.

PPO applies successfully to control tasks such as robotic manipulation, autonomous driving, and locomotion, outperforming many prior approaches regarding performance and computational efficiency. Its effectiveness in environments with high-dimensional state spaces positions it as a strong candidate for strategic board games like Hex [6].

C. Deep Q-Learning

DQL integrates Q-Learning with deep neural networks to facilitate learning of value functions for high-dimensional state spaces [7]. Mnih et al. [7] introduce DQL with innovations like experience replay and target networks, stabilizing training and significantly enhancing learning efficiency and policy performance in complex environments.

Applying DQL to Hex involves leveraging deep neural networks to approximate the Q-values of game states. This allows the agent to learn optimal moves through extensive self-play and experience replay. Challenges such as state space dimensionality and sparse rewards are addressed using advanced techniques like Double DQL and Dueling Network Architectures, which further stabilize and improve the performance of learning agents in Hex [7]–[10].

III. Methods

This section introduces the methodologies employed to develop and evaluate AI agents within the Hex game environment. The framework is structured into several key subsections: Hex Engine, Agent Class, PPO Class, DQL Class, Training Pipeline and Training Process.

The Hex Engine subsection implements core mechanics and utilities for simulating the Hex game environment. It facilitates game management and evaluation, and includes features for accessibility such as colour-blind AI training.

The Agent class provides a flexible framework for developing AI agents tailored to Hex game complexities. It manages Hex environment interactions, ensures model persistence, and supports comprehensive performance evaluation. Key features include saving/loading models, strategic decision-making, and real-time board visualization. This framework facilitates iterative agent training and optimization in Hex game scenarios.

The PPO Class refines the Stable Baselines3 PPO agent framework for optimal performance in Hex game scenarios. It incorporates custom adaptations and undergoes empirical testing to enhance its effectiveness.

The DQL Class implements a neural network-based algorithm tailored for strategic decision-making within the Hex game environment. It supports configurable architectures and iterative learning processes to improve agent strategies over time.

The Training Pipeline outlines a structured approach for training AI agents. It includes phases such as initial policy diversity, tournament evaluation, and iterative refinement through dynamic strategy adjustments and performance-based checkpoints. This pipeline aims to continuously improve agent performance and adaptability in complex gaming environments.

The training process begins with a PPO proof of concept on a 4x4 Hex board, adapting the environment for rapid iteration. Transitioning to a 7x7 board highlights the need for colour-specific strategies, addressed through a colour-focus mode to improve agent performance with both black and white tiles. Despite challenges, PPO fails to converge effectively, leading to a switch to DQL, which successfully trains agents surpassing random and trained counterparts. Adding a convolutional block further enhances performance by recognizing keyboard patterns effectively.

A. Hex Engine

The hex engine class embodies the essential mechanics of a Hex game environment tailored for research in RL. Hex is a strategic game where players aim to connect opposite sides of a hexagonal board. This class streamlines game management and evaluation through a suite of methods and attributes. Upon instantiation, an object of the hex engine initializes with a specified board size, typically defaulting to 7x7 but constrained between 2x2 and 26x26. Key attributes include a board to represent the game state, a player indicating the current player ('white' or 'black'), and a winner determining the game outcome ('white', 'black', or 'no winner' until game conclusion). Fundamental methods enable gameplay operations: move permits stone placement, reset clears the board, and evaluate assesses if a player has won. The class provides visual representation through print, showcasing the board in a terminal-friendly format.

For research flexibility, utility methods like get_action_space enumerate available moves, replay_history revisits past game states, and save serializes game instances for persistence. Advanced functionalities support AI development and experimentation: human_vs_machine enables human-computer gameplay, machine_vs_machine pits AI algorithms against each other, and recode_black_as_white transforms the board to facilitate colour-blind AI training.

The recode_black_as_white function is pivotal for colour-blind AI training in Hex. It transforms the board so that originally 'black' stones (represented as -1 in the board) become 'white' (1), and vice versa, by flipping colours along the southwest-to-northeast diagonal. This ensures AI models learn strategies without bias from colour-specific features, crucial for robust performance across diverse game scenarios.

Overall, the hex engine class offers a robust foundation for studying Hex game strategies within RL. It provides controlled environments and essential utilities for experimental research and algorithm development, supporting advancements in AI-driven decision-making and gameplay strategies.

B. Agent Class

The agent class serves as a foundational framework for developing AI agents (PPO and DQL) that interact intelligently with the Hex game environment. Upon instantiation, the class initializes with placeholders for the game environment (env) and the agent's model (model), ensuring flexibility in integrating different game scenarios and AI architectures. Methods such as set_env allow seamless configuration of the game environment, enabling agents to operate within specific contexts tailored to experimental needs.

For persistency and continuity across sessions, the class includes methods like save and load to serialize and deserialize models. This feature supports iterative model refinement and comparative analysis of agent performance over extended training periods. Central to its utility is the evaluate_games method, which systematically assesses an agent's performance by simulating multiple game instances (games). This evaluation includes tracking game outcomes, computing win rates, and optionally testing against varied opponent policies (opponent_policy). Such systematic evaluation provides empirical insights into the efficacy of agent strategies under different conditions, crucial for refining and optimizing AI decision-making processes.

In addition to evaluation, the class offers practical utilities such as get_masked_actions, which generates action masks based on the current board state. This function aids agents in focusing on valid moves, thereby enhancing decision-making efficiency and strategic depth during gameplay. Visualization is facilitated through print_board, which utilizes the hexPosition class to display the current game board in a user-friendly format. This feature supports real-time monitoring and debugging, crucial for understanding agent behaviour and diagnosing performance issues. Concrete subclasses derived from Agent are required to implement core methods (init_model, train, get_action, policy) that define how agents initialize their models, learn from interactions, and decide actions based on game states. These methods form the backbone of agent intelligence, enabling adaptive learning and strategic adaptation in response to evolving game dynamics.

In summary, the class serves as the parent class for specialized subclasses like PPO and DQL. It provides foundational functionalities for managing interactions with the Hex game environment, handling model logistics, evaluating performance, and making strategic decisions. This hierarchical structure enables PPO and DQL subclasses to inherit and extend core functionalities, facilitating tailored implementations of advanced RL algorithms specific to Hex gameplay.

C. PPO Class

In the initial phases of experimentation, testing involves the utilization of the Stable Baselines3 PPO agent framework. This established baseline provides a reference for validating the functionality and performance benchmarks within the Hex gameplay environment. Once the initial setup and validation stages are completed, further refinement and customization are pursued through the development of a bespoke PPO agent implementation. This custom agent amalgamates insights and methodologies gleaned from both the Stable Baselines3 PPO and supplementary sources, such as educational content from online platforms like YouTube.

The bespoke PPO agent, integrated into the broader framework of the PPOAgent class, is designed to enhance adaptability and performance optimization specifically tailored for Hex gameplay scenarios. Key adaptations include modifications to network architecture parameters such as hidden layer configurations and feature extraction methods, incorporating convolutional layers for enhanced spatial awareness if warranted. These adjustments aim to optimize decision-making processes, leveraging strategic insights and empirical findings derived from iterative testing and evaluation phases.

Throughout development and implementation, validation and comparative analysis against the Stable Baselines3 PPO agent are conducted to assess performance improvements and validate the efficacy of custom enhancements. This iterative refinement process not only validates the robustness of the custom agent but also contributes empirical evidence and insights relevant to advancing AI capabilities in complex gaming environments, particularly within the realm of RL research.

D. Deep Q-Learning Class

The DQL agent class implements a DQL algorithm tailored for optimizing strategies within the Hex game environment. It serves as a foundational component within the framework of AI agents designed specifically for Hex gameplay, leveraging principles from RL. Upon instantiation, the DQL agent initializes with configurable parameters such as hidden layer count, size, and the potential use of convolutional layers. These parameters define the neural network architecture crucial for interacting intelligently with the Hex environment and learning effective gameplay strategies.

The init_model method sets up the DQL model by configuring the neural network architecture based on the dimensions of the environment's observation space and the size of its action space. Hyperparameters such as learning rate, batch size, and specific architectural features like convolutional layers are precisely specified to optimize the agent's learning process and enhance decision-making capabilities. During the training phase, managed by the training method, the agent engages in iterative interactions with the Hex environment across multiple episodes. Employing an epsilon-greedy strategy, the agent dynamically adjusts the exploration rate (epsilon) to balance exploration of new actions and exploitation of learned policies, ensuring continuous learning and adaptation. Within each training episode, the agent interacts with the Hex environment by selecting actions based on the current state and the learned policy from the DQL model. Significant gameplay experiences, including state-action-reward transitions, are stored in dedicated memory buffers (episode_memory and opponent_episode_memory). These memories are critical for reinforcing successful strategies through periodic replay and learning updates.

The get_action method facilitates decision-making by selecting actions based on the current state of the Hex game board and the learned policy from the DQL model. It incorporates mechanisms to adjust the exploration rate (epsilon) dynamically, ensuring a balanced exploration-exploitation trade-off in action selection. The policy method encapsulates the agent's strategic decision-making process, adapting its approach based on the current player's turn and accommodating specific adjustments required for scenarios where board colour representation needs modification. This feature is particularly relevant for addressing colour-blindness in AI applications aimed at Hex gameplay.

Tested architectures include a 2-layer fully connected network with 256 units, a convolutional block followed by a 2-layer fully connected network with 256 units, and a convolutional block followed by a 3-layer fully connected network with 256 units (promising but under-trained). These architectures were developed based on insights from lecture materials, emphasizing both practical application and theoretical underpinnings relevant to the Hex game environment. Throughout its developmental stages, the DQL agent underwent testing and validation against established benchmarks and alternative neural network configurations. This evaluation aimed to refine the agent's learning efficiency and strategic acumen within the complex dynamics of Hex gameplay environments.

In essence, the DQL agent class exemplifies a disciplined application of DQL principles tailored specifically to master strategic challenges posed by Hex. Its deployment underscores its significant contribution to advancing AI research in RL, particularly in the domain of strategic decision-making in complex board games.

E. Training Pipeline

This section delineates the comprehensive training methodology employed to develop AI agents for the Hex game environment. It comprises three distinct subsections: "Self Play from Random," "Tournament," and "Dojo." Each subsection details a specific phase of the training process, designed to iteratively refine agent strategies through diverse and challenging scenarios. The overarching goal is to ensure the development of robust and adaptable AI models capable of handling various strategic complexities inherent in Hex gameplay.

1) Self play from random: To achieve diverse and efficient training for each agent, a general batch of AI models is created initially. This approach establishes a base of models that are slightly better than random, providing a foundation for developing varied policies. This process is illustrated in Figure 2.

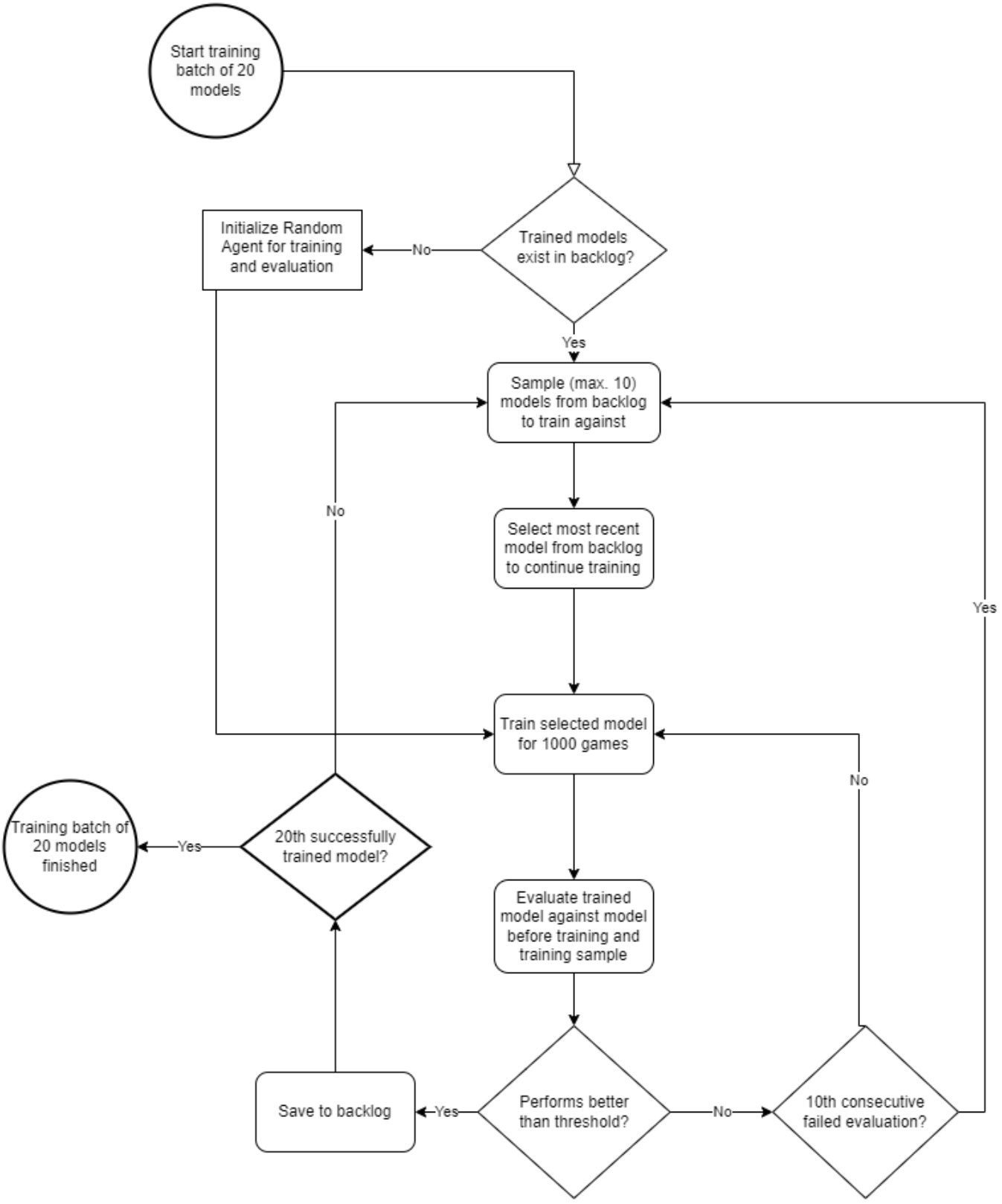

Fig. 2. Self play from random pipeline.

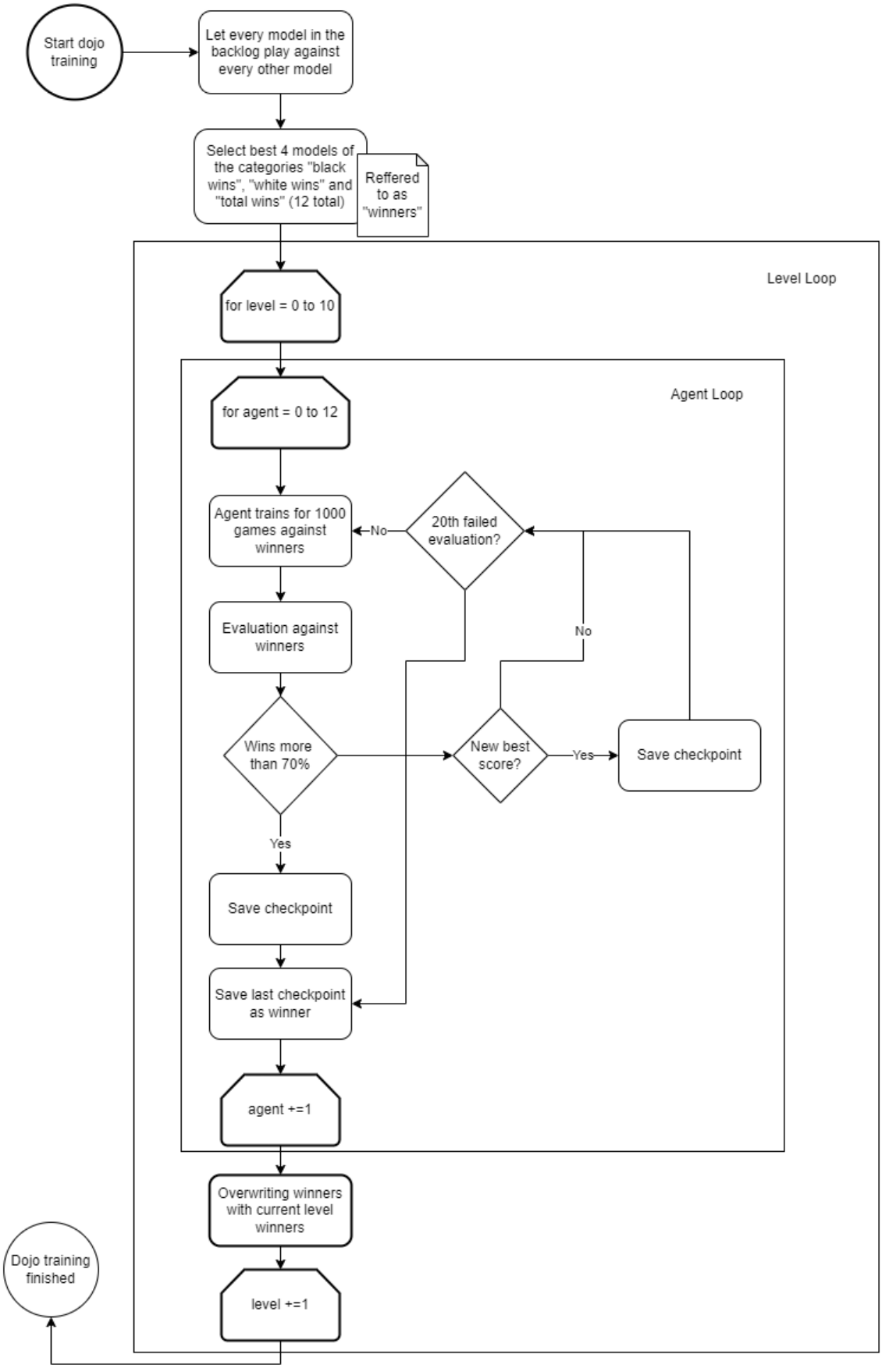

Fig. 3. Dojo Pipeline.

The training pipeline commences with the initiation of a batch of 20 AI models. The first step involves checking for the existence of trained models in the backlog. In the absence of trained models, a random agent is initialized for training and evaluation. If trained models are available, the pipeline samples up to 10 models from the backlog to serve as training opponents. This step ensures the presence of diverse policies, which the current agent must overcome to gain experience from playing against strategies. Opponents are sampled from a Gaussian distribution, slightly shifted relative to the current models, to optimize policy diversity and efficacy.

Subsequently, the pipeline selects the most recent model from the backlog to continue its training. This selected model undergoes additional training through 1000 games, refining its strategies. Post-training, the model is evaluated against both the model used prior to this session and the training sample from the backlog. This evaluation assesses whether the trained model surpasses a predefined performance threshold.

If the model meets the performance threshold, the pipeline checks if it is the 20th successfully trained model. If the model does not meet the threshold, the pipeline determines whether this is the 10th consecutive failed evaluation. If not, the model is subjected to further training. If it is the 10th consecutive failure, the model training is terminated. Upon achieving the 20th successfully trained model, the training batch is deemed complete, and the model is saved to the backlog for future utilization.

This systematic process integrates the concept of continuously pitting multiple policies against the current policy. Throughout training and evaluation, the pipeline consistently challenges the current policy with various other policies (up to 10 from the backlog). This methodology fosters the development of robust and adaptable strategies. By training against a diverse set of opponents and sampling different models from the backlog, the pipeline ensures that the AI agent evolves multiple strategies to effectively handle diverse situations. This approach promotes adaptability and robust decision-making in the Hex game environment, ensuring continuous improvement of AI agents through exposure to a variety of strategies and scenarios.

2) Tournament: A tournament framework is structured to identify optimal models for strategic decision-making within Hex game environments. This framework systematically evaluates AI agents across diverse model architectures and strategic approaches. Each agent competes as both white and black pieces, and their performance is rigorously assessed based on game outcomes. The tournament generates detailed metrics, consolidating performance data for comprehensive comparative analysis. Top-performing agents are selected based on cumulative victories across various categories, including total wins, wins as white, and wins as black. Selected agents have their models preserved in dedicated directories, ensuring retention of the most effective strategies for ongoing deployment and refinement in Hex gameplay scenarios.

3) Dojo: The dojo training process is a meticulously structured and iterative approach crafted to refine AI agent models tailored for mastering strategic decision-making on Hex game boards. This process spans multiple levels, typically totalling 10, and iteratively enhances agent policies by leveraging insights gained from encounters with diverse opponent strategies.

The process begins by letting every model in the backlog play against every other model. Based on these matches, the top four models in the categories of ”black wins,” ”white wins,” and ”total wins” are selected, resulting in a total of 12 models referred to as ”winners.”

The training regimen then enters a loop where it iterates over 10 levels, each consisting of training 12 agents. For each level, an agent trains for 1000 games against the winners. During these sessions, agents engage in gameplay simulations where they dynamically adjust their strategies based on real-time feedback and outcomes. A pivotal component of this iterative process is the policy selection mechanism, which orchestrates a balance between exploring novel strategies and exploiting well-learned tactics across both white and black gameplay phases. This ensures that agents continuously adapt and refine their decision-making processes in response to evolving game dynamics.

After each training session, the agent is evaluated against the winners. If the agent wins more than 70% of the matches, a checkpoint is saved. If a new best score is achieved, another checkpoint is saved. This sophisticated evaluation protocol ensures that only the most effective strategies are retained. If the agent does not achieve the required performance, it undergoes additional training unless it reaches the 20th failed evaluation, at which point the training is terminated.

Upon achieving the 20th successfully trained model, the training batch is complete, and the successfully trained model is saved to the backlog for future use. This structured advancement mechanism promotes only the most effective strategies, fostering a competitive environment where agents continually strive for improvement.

By iterating through this process, the dojo training process ensures that AI agents are not confined to a single strategy but evolve multiple strategies to handle diverse situations effectively. This approach integrates the concept of pitting multiple policies against the current policy, continuously challenging the current policy with various other policies from the backlog. Through training against a diverse set of opponents and sampling different models, the process promotes adaptability and robust decision-making in the Hex game environment. A detailed architectural process is shown in Figure 3.

The complete training pipeline is illustrated in Figure 4. This pipeline can be iterated multiple times to generate a diverse set of models. Through repeated training cycles, each model learns from its opponents, progressively enhancing its strategies. Consequently, with each iteration, the models face increasingly sophisticated opponents, leading to continuous improvement in performance.

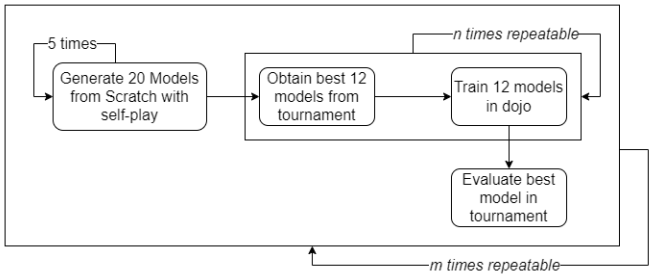

Fig. 4. Overall Pipeline.

F. Training Process

This section outlines the steps taken from building a proof of concept to model selection, culminating in the development of a robust agent.

1) Proof of Concept: Initially, the PPO implementation from the Stable Baselines3 package is utilized with a 4x4 Hex board. This approach allows for the construction of a suitable environment wrapper for the Hex engine. By employing a state-of-the-art implementation of a well-established advanced RL algorithm in a smaller action space environment, rapid iteration over changes in the environment architecture is feasible. For instance, ensuring correct agent training in a colour-blind manner by inverting the board when playing the black tiles is crucial. Upon successfully training agents capable of defeating both random and trained agents, the transition to a 7x7 board is made. This step is essential for refining the training pipeline, as it is observed that agents only learn suitable strategies when playing the white tiles on the 7x7 board. To address this issue, a colour focus mode is introduced. When an agent's win rate is less than 30% with a particular colour, the subsequent training run is dedicated exclusively to playing as that colour. This approach enables the production of agents that learn suitable strategies for both colours. After training several agents that perform well following multiple training runs with both colours, the custom implementation of the algorithm is adopted.

2) PPO: The logical progression involves replacing the Stable Baselines3 PPO implementation with a custom implementation. Despite various combinations of hyperparameters being tested, the PPO architecture fails to converge. The algorithm trains for thousands of games but still performs on par with or worse than a random agent. Consequently, a shift in strategy is made to train a DQL agent in the environment.

3) DQL: The implementation of DQL results in agents that converge on suitable strategies, outperforming both random and trained agents. To maximize learning potential per episode, the opponent's episodes are also tracked. Thus, for every state, action, reward, and next state set collected from the agent, the same data is collected from the opponent's perspective. This method allows weak agents to play against stronger ones to adapt their strategies and improve quickly. Additionally, for each winning game, an agent also learns what strategies to avoid when playing against itself.

4) Convolutional Block: Following favourable results with the DQL agent, a convolutional block is added to the network to assess its performance relative to agents with networks composed solely of fully connected layers. This proves to be an effective strategy, as agents with convolutional layers quickly outperform the previous agents. This improvement is likely due to the nature of the Hex game, where relevant patterns are effectively recognized on the 2-dimensional board by convolutional layers.

IV. Results

This section provides a detailed comparison of the agents discussed. Initially, the evaluation metrics employed for the comparison are introduced. Subsequently, the performance results of the agents are presented and evaluated.

A. Evaluation Metrics

In the evaluation phase, models undergo assessment in a tournament-style setup, where each agent competes against every other agent as both black and white pieces. This structured approach captures performance metrics such as wins when playing as white, wins as black, and total wins for each agent. By systematically logging these outcomes, the tournament framework provides comprehensive data on each agent's strategic progress across multiple game scenarios.

B. PPO

As the PPO agents fail to converge on a strategy that consistently outperforms a random agent, they are excluded from the final evaluation. Their average rewards and average episode lengths during training are depicted in 5 and 6, respectively. While the graphs indicate a gradual increase in reward over time, in practice, they do not achieve consistent superiority over random agents during evaluation. This inconsistency underscores their exclusion from the final assessment, as they do not demonstrate robust performance metrics required for competitive evaluation in the study.

Fig. 5. Average Episode Reward PPO.

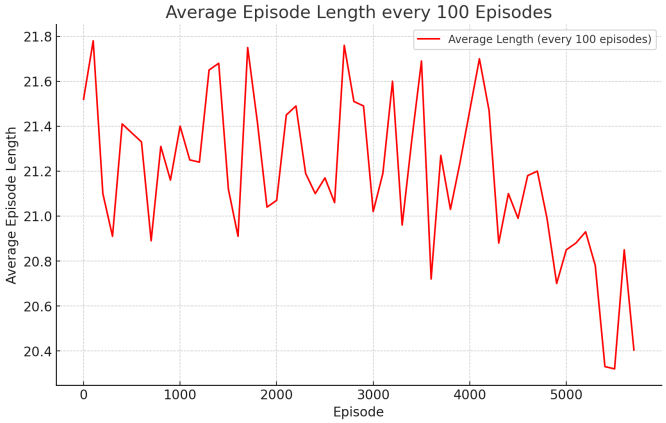

Fig. 6. Average Episode Length PPO.

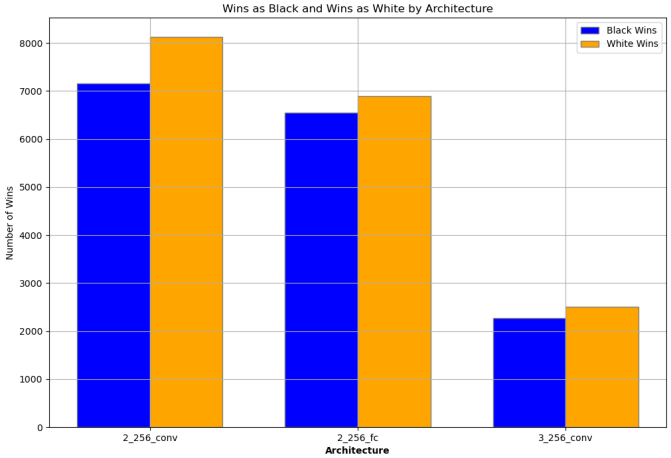

Fig. 7. Wins per color per architecture

C. Deep Q-Learning

The DQL agents are separated into three groups that represent their network architecture:

- Agents with a fully connected architecture containing 2 hidden layers of the size 256 (2_256_fc)
- Agents with a convolutional block in front of 2 hidden layers of the size 256 (2_256_conv)
- Agents with a convolutional block in front of 3 hidden layers of the size 256 (3_256_conv)

The results shown in Figure 7 highlight a clear and statistically significant disparity in performance across different architectural configurations. Agents utilizing the 2_256_conv architecture consistently achieve the highest win rates regardless of playing colour (black or white). This observation suggests that convolutional layers effectively extract pertinent spatial features from the Hex board representation, thereby enhancing decision-making capabilities compared to the fully connected architecture (2_256_fc).

Interestingly, the 3_256_conv architecture, despite having an additional hidden layer compared to 2_256_conv, demonstrates the lowest performance. This outcome implies that increased network complexity may not necessarily translate to improved performance in this specific context. Possible factors contributing to this discrepancy could include issues such as overfitting or challenges in learning optimal strategies with the additional layer.

As expected in Hex, all agents exhibited a higher win rate when playing white, confirming the well-established advantage that the first player (white) possesses in the game.

Furthermore, the win distribution across various evaluation metrics, as shown in Figure 8, remains consistent for all architectures. This consistency suggests that the overall win rate effectively captures performance across these metrics, validating the robustness of the evaluation process.

In essence, the convolutional layers in the 2_256_conv architecture appear to have captured the critical strategic elements of the Hex game across various performance measures, regardless of whether the agent played first or second. This reinforces the notion that convolutional layers are well-suited for extracting relevant spatial features in this domain.

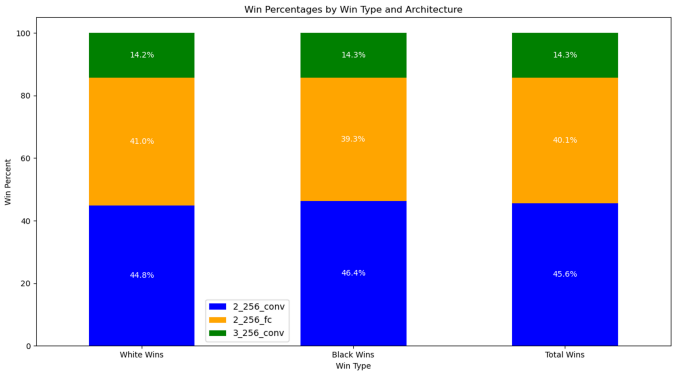

Fig. 8. Distribution of metrics per architecture

The results of the tournament evaluation show that agents with the 2_256_conv architecture outperformed other agents in all metrics. 2_256_fc came in second place and 3_256_conv performed the worst. This is shown in 7, furthermore, the graph shows that across architectures agents tend to win as white more frequently which is to be expected as in the game of hex the white player has first mover advantage. The distribution of wins across metrics is shown in 8 this graphic illustrates that the architectures perform similarly well across metrics with no architecture performing better in one discipline and worse in another.

V. Discussion

This section delves into critical factors influencing the performance of DQL agents in the Hex game environment. It begins by analyzing the impact of state copy implementations, emphasizing the necessity of deep copies to enhance the agent's ability to effectively learn state transitions. Moreover,

the discussion explores the challenges posed by episode mirroring delay and the importance of accurate opponent handling to ensure robust training outcomes.

Furthermore, the section investigates how architectural complexity and training duration affect agent performance. Findings indicate that agents equipped with convolutional layers and configured with 2 hidden layers consistently outperform their counterparts with 3 layers. This superiority is attributed to the early initiation of training for the 2 hidden layer agents, highlighting the critical role of extended training periods in optimizing performance for complex architectures.

A. Shallow copy vs. deep copy

While implementing the DQL training function, it is observed that the state and next state of an episode are identical. Upon inspecting the variable values at runtime, it is discovered that the state variable in the train function, whose value is obtained from the environment, updates once the state in the environment changes. This issue arises because the environment returns shallow copies of the state rather than deep copies. As the state is a 2-dimensional array, a shallow copy returns an array of array references rather than an array of arrays. This causes the state variable's value to change when the underlying referenced arrays are modified, resulting in the same value being stored for both the state and the next state in the agent's memory. This impedes learning, as the agent cannot determine which actions lead from one state to another. After modifying the environment to return only deep copies of the state, agents can find suitable strategies.

B. Episode Mirroring Delay

Episode mirroring refers to tracking the opponent's state, action, reward, and next state sets in addition to those of the agent. When implementing episode mirroring, it is crucial to correctly track the opponent's steps, as their state and next state values are observed in different steps of the training loop. Additionally, it is essential to consider the edge cases of the first and last steps, as these can occur outside of the loop tracking the agent.

C. Time Constraint

The results indicate that agents employing convolutional layers in their network architecture exhibit superior performance when configured with 2 hidden layers compared to those with 3 hidden layers. This performance disparity stems from the earlier initiation of training for the 2 hidden layer agents, providing them with a significant "head start" over their 3 hidden layer counterparts. The prolonged training duration required for the 3 hidden layer agents indicates a slower convergence towards optimal strategies. It is hypothesized that with additional training time, models utilizing larger architectures have the potential to surpass the performance of other configurations by undergoing further refinement and adaptation of their strategic decision-making processes.

VI. Conclusion

This study provides a comprehensive analysis of advanced RL algorithms applied to the game of Hex, focusing on DQL and PPO. The evaluation of these algorithms within the Hex environment highlights their strengths, challenges, and critical factors influencing agent performance. The initial exploration involved both PPO from the Stable Baselines3 package and a custom implementation of DQL. While PPO initially showed promise, it failed to consistently outperform random agents in evaluations. Consequently, the focus shifted to DQL, where various architectural configurations were tested to optimize performance.

One of the significant findings of this study is the notable performance enhancement achieved by agents utilizing convolutional layers, particularly in the 2_256_conv architecture. These agents consistently demonstrate superior win rates across both black and white playing colours compared to agents with fully connected architectures or additional hidden layers (3_256_conv). The effectiveness of convolutional layers in capturing relevant spatial features from the Hex board underscores their suitability for enhancing decision-making capabilities in complex game environments.

Furthermore, the study reveals that the early initiation of training for agents with 2 hidden layers, coupled with convolutional layers, is crucial for achieving robust performance. Agents with 3 hidden layers exhibit slower convergence and inferior outcomes, highlighting the potential pitfalls of increased network complexity without commensurate training duration. This underscores the importance of extended training periods to fully exploit the capabilities of complex neural network architectures in advanced RL applications.

Recommendations for future research include further optimization of convolutional architectures tailored specifically for Hex gameplay. Comparative studies with state-of-the-art image recognition architectures such as ResNet or VGG could provide insights into adapting neural network designs to enhance advanced RL agent performance in strategic board games like Hex. Additionally, addressing challenges related to episode mirroring delay and ensuring accurate opponent handling is crucial for refining training methodologies and improving agent robustness in real-world applications.

In conclusion, while DQL proves effective in mastering strategic complexities inherent in Hex, ongoing research and development are essential to overcome specific algorithmic challenges and harness the full potential of advanced RL in competitive gaming scenarios. This study contributes valuable insights toward advancing the capabilities of AI agents in strategic decision-making domains, with implications extending beyond gaming to broader applications in artificial intelligence and machine learning.

Acknowledgment

We extend our heartfelt thanks to Sharwin Rezagholi for his invaluable support during our weekly meetings, where he provided crucial ideas and the foundational implementation for the Hex engine. His expertise and dedication have significantly

enriched this project, laying a solid groundwork for our research in reinforcement learning and AI-driven strategies within the Hex game environment.

REFERENCES

[1] Shakya A. K., Pillai G., Chakrabarty S., "Reinforcement learning algorithms: A brief survey", *Expert Systems with Applications*, Vol.231, Art.no.120495, 2023. https://doi.org/10.1016/j.eswa.2023.120495

[2] Luo FM., Xu T., Lai H., Chen XH., Zhang W., Yu Y., "A survey on model-based reinforcement learning", *Science China Information Sciences*, Vol.67, No.2, Art.no.121101, pp.1869-1919, 2024. https://doi.org/10.1007/s11432-022-3696-5

[3] Anshelevich V. V., "A hierarchical approach to computer Hex", *Artificial Intelligence*, Vol.134, No.1, pp.101-120, 2002. https://doi.org/10.1016/S0004-3702(01)00154-0

[4] Arneson B., Hayward R. B., Henderson P., "Monte Carlo Tree Search in Hex", *IEEE Transactions on Computational Intelligence and AI in Games*, Vol.2, No.4, pp.251-258, 2010. https://doi.org/10.1109/TCIAIG.2010.2067212

[5] Gao C., Hayward R., Müller M., "Move Prediction Using Deep Convolutional Neural Networks in Hex," in: *IEEE Transactions on Games*, Vol.10, No.4, pp. 336-343, 2018. https://doi.org/10.1109/TG.2017.2785042

[6] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O., "Proximal Policy Optimization Algorithms", *arXiv.org*, 2017. https://doi.org/10.48550/arXiv.1707.06347

[7] Mnih V., Kavukcuoglu K., Silver D., Rusu A. A., Veness J., Bellemare M. G., Graves A., Riedmiller M., Fidjeland A. K., Ostrovski G., Petersen S., Beattie C., Sadik A., Antonoglou I., King H., Kumaran D., Wierstra D., Legg S., Hassabis D., "Human-level control through deep reinforcement learning", *Nature*, Vol.518, pp.529-533, 2015. https://doi.org/10.1038/nature14236

[8] Silver D., Schrittwieser J., Simonyan K., Antonoglou I., Huang A., Guez A., Hubert T., Baker L., Lai M., Bolton A., Chen Y., Lillicrap T., Hui F., Sifre L., van den Driessche G., Graepel T., Hassabis D., "Mastering the game of Go without human knowledge", *Nature*, Vol.550, pp.354–359, 2017. https://doi.org/10.1038/nature24270

[9] van Hasselt H., Guez A., Silver D., "Deep Reinforcement Learning with Double Q-learning", *arXiv.org*, 2015. https://doi.org/10.48550/arXiv.1509.06461

[10] Wang Z., Schaul T., Hessel M., van Hasselt H., Lanctot M., de Freitas N., "Dueling Network Architectures for Deep Reinforcement Learning", *arXiv.org*, 2016. https://doi.org/10.48550/arXiv.1511.06581