Schwab Royale: A Reinforcement Learning Approach Using Deep Q-Networks

Harshith Manikhanta Sunkara, Sai Prudhvi Ram
Department of Computer Science
VIT-AP University
Amaravathi, India
Email: harshith.22bce7836@vitapstudent.ac.in, prudhvi.22bce7137@vitapstudent.ac.in

***Abstract*—Schwab Royale is an advanced multiplayer battle royale game that leverages reinforcement learning techniques to optimize decision-making and enhance gameplay experiences. This report explores the application of Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) to improve player strategies, non-player character (NPC) behaviors, and game mechanics. By utilizing DDQN, the game reduces overestimation bias, resulting in more stable and efficient learning compared to traditional Q-learning methods. The study first reviews the foundations of reinforcement learning, highlighting advancements in deep learning-based Q-networks and their impact on gaming environments. Schwab Royale integrates AI-driven decision-making, where players navigate complex battle arenas with dynamically changing elements. The implementation involves training deep neural networks to predict optimal actions, utilizing an epsilon-greedy policy for exploration, and employing experience replay for efficient learning. A comprehensive evaluation framework is developed to measure reward accumulation, learning stability, computational efficiency, and overall gameplay performance. Results indicate that DDQN significantly improves strategic decision-making, allowing for more intelligent and adaptive player interactions. The study also explores potential enhancements, including multi-agent reinforcement learning, virtual and augmented reality integration, and AI-driven anti-cheat mechanisms.**

I. Introduction

Schwab Royale is a competitive multiplayer battle royale game that challenges players to survive and strategize in a dynamic combat environment. With the rise of artificial intelligence in gaming, reinforcement learning techniques, particularly Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN), have shown promise in enhancing game mechanics and player decision-making. By integrating these AI-driven techniques, Schwab Royale can create a more adaptive and intelligent gaming experience. Reinforcement learning (RL) enables agents to learn optimal strategies through trial-and-error interactions within the game environment. Traditional Q-learning methods often suffer from overestimation bias, leading to unstable decision-making. DDQN addresses this issue by using separate networks for action selection and evaluation, improving the learning process and overall game stability. The primary objective of this project is to implement and evaluate the effectiveness of DDQN in optimizing in-game decisions, NPC behaviors, and player interactions. This involves designing a neural network architecture capable of handling complex combat scenarios, utilizing experience replay, and employing an epsilon-greedy strategy for balanced exploration and exploitation. This report explores the application of RL in Schwab Royale, detailing the methodology, implementation, and evaluation metrics. The findings demonstrate that DDQN enhances strategic decision-making, providing players with a more engaging and competitive experience. Future improvements will focus on expanding AI capabilities, real-time adaptation, and immersive gameplay elements such as virtual and augmented reality integration.

A. Motivation:

Traditional gaming AI often relies on scripted behaviors, limiting adaptability and strategic depth. Reinforcement learning (RL) offers a powerful alternative by enabling AI agents to learn optimal strategies through experience. By implementing DDQN, Schwab Royale aims to create a more dynamic and challenging gameplay experience where AI-driven opponents and decision-making adapt to evolving game conditions, enhancing engagement and competitiveness.

B. Objectives:

- Implement Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) to optimize in-game decision-making.
- Reduce overestimation bias and improve stability in AI training.
- Develop an AI-driven opponent system capable of learning from player interactions.
- Evaluate the performance of reinforcement learning in a real-time multiplayer gaming environment.

Literature Review

Schwab Royale's integration of reinforcement learning (RL) techniques builds upon significant advancements in deep learning and game AI research. This section explores the foundations of RL, major breakthroughs in deep RL, and the application of continuous control algorithms to improve in-game decision-making.

1. Foundations of Reinforcement Learning (RL)

Reinforcement Learning (RL) is a machine learning paradigm where agents learn optimal actions through trial and error by interacting with an environment. It is formalized

within the Markov Decision Process (MDP) framework, consisting of states, actions, rewards, and transition probabilities. Sutton and Barto (1998) laid the groundwork for RL with their work on policy optimization and Q-learning.

A key advancement in RL was the introduction of Q-learning, a model-free approach that enables agents to estimate action values without prior knowledge of the environment's dynamics. However, traditional tabular Q-learning struggles with high-dimensional state spaces, which led to the emergence of Deep Q-Networks (DQN) as a scalable solution.

2. Deep Reinforcement Learning (Deep RL) Breakthroughs

Deep RL combines deep learning with RL algorithms to handle complex, high-dimensional problems. Mnih et al. (2015) revolutionized the field by introducing Deep Q-Networks (DQN), which use neural networks to approximate Q-values and enable human-level performance in video games. The key innovations in DQN include:

- Experience Replay: Storing past experiences to improve learning stability.
- Target Networks: Using separate networks for evaluation and learning to reduce instability.

However, DQN suffers from overestimation bias, leading to suboptimal policies. Double Deep Q-Networks (DDQN), proposed by Van Hasselt et al. (2016), addresses this issue by decoupling action selection from evaluation, resulting in improved training stability. Further advancements include Dueling DQN, Prioritized Experience Replay, and Rainbow DQN, which integrate multiple optimizations for better performance.

3. Continuous Control Algorithms

While DQN and DDQN work well for discrete action spaces, real-world applications, such as autonomous driving and robotic control, require continuous action spaces. Several algorithms address this challenge:

- Deep Deterministic Policy Gradient (DDPG): A model-free RL approach that combines policy gradients with Q-learning for continuous control.
- Proximal Policy Optimization (PPO): An improved version of policy gradient methods, balancing exploration and exploitation efficiently.
- Soft Actor-Critic (SAC): Uses entropy maximization to improve learning efficiency and robustness in continuous environments.

4. Parameter Sharing Applications

Parameter sharing is a technique in reinforcement learning where multiple agents share a common neural network to reduce computational complexity and improve learning efficiency. In Schwab Royale, parameter sharing can be applied to train multiple AI-controlled opponents simultaneously, enabling them to learn from collective experiences while maintaining distinct behaviors. This approach enhances multi-agent coordination, reduces training time, and improves generalization across different game scenarios. By implementing parameter sharing, Schwab Royale can create more realistic, adaptive, and competitive AI opponents in the battle royale environment.

Problem Statement

Traditional gaming AI relies on scripted behaviors, limiting adaptability and strategic depth in multiplayer battle royale games. Schwab Royale aims to implement Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) to enhance decision-making, reduce overestimation bias, and create dynamic AI-driven gameplay. The challenge lies in training AI agents to learn optimal strategies, adapt to player actions, and improve game balance in real-time. Additionally, ensuring computational efficiency and smooth integration of RL models within a multiplayer environment is crucial.

Environment Design: Grid-Based World

The environment is structured as a 10x10 grid where agents interact and make decisions based on their surroundings. This grid serves as the battlefield, where agents can:

- Move in four directions.
- Attack opponents.
- Heal teammates.
- Communicate with their team.

Simplifies state representation – Positions are discrete, making it easier for agents to learn.
Enables strategic positioning – Agents must decide the best place to move, attack, or retreat.
Allows team-based coordination – Agents can communicate and plan together.

Multi-Agent System: Teams and Roles

There are 6 agents, divided into two teams:

II. Roles and Effects Table

The table below describes the roles and their corresponding effects:

- **Team 1:** agent_0, agent_2, agent_4
- **Team 2:** agent_1, agent_3, agent_5

III. Roles and Effects Table

The table below describes the roles and their corresponding effects in the game: Each agent has a specific role within their team.

Role	Effect
Healer	Restores +5 health to teammates.
Scorer	Attacks enemies and reduces their health by -10.
Soldier	Attacks enemies but with lower damage (-5).

IV. Roles

- Encourages specialization in agent behavior.
- Creates a balance between offense and defense.
- Requires team coordination for optimal performance.

V. Available Actions

Each agent can take one of seven possible actions per step:

Action	Effect
`move_up`	Moves agent up on the grid.
`move_down`	Moves agent down on the grid.
`move_left`	Moves agent left on the grid.
`move_right`	Moves agent right on the grid.
`attack`	Deals damage to an opponent.
`heal`	Restores health to a teammate.
`communicate`	Shares information with teammates.

1) Move Up
2) Move Down
3) Move Left
4) Move Right
5) Attack
6) Heal
7) Communicate

VI. Observations: What Each Agent Sees

Each agent receives partial information about the environment, which includes:

- **Current position:** (x, y coordinates).
- **Current health:** 100 initially, but decreases with damage.
- **Team affiliation:** 0 or 1, representing the two teams.

Partial Observations

- Mimics real-world limitations (agents don't have full map awareness).
- Forces agents to make decisions with uncertainty.
- Encourages collaboration through communication.

VII. Reward System

Rewards are designed to balance individual and team-based incentives:

Type of Reward	Reward Value	Reason
Attack Success	+10	Encourages eliminating opponents.
Healing Teammate	+5	Rewards team support.
Movement	+1	Encourages exploration and positioning.
Communication	+2	Rewards teamwork.
Eliminating an Opponent Team	Team Bonus	Incentivizes teamwork.
Agent Death	Negative Reward	Discourages reckless behavior.

- Positive rewards for eliminating opponents.
- Negative rewards for attacking teammates.
- Bonus points for successful healing or strategic positioning.
- Team-wide rewards for achieving objectives, promoting teamwork.

VIII. Game Loop: How the Game Progresses

The game follows a structured loop to ensure smooth execution:

1) **Initialize:**
   The environment sets up agent positions randomly.
   Each agent starts with 100 health.
2) **Observe:**
   Agents receive their current state (position, health, team info).
3) **Decide:**
   Each agent selects an action using Multi-Agent Deep Q-Learning (MADQN).
4) **Act:**
   The environment updates agent positions, health, and status.
5) **Reward:**
   Agents receive rewards or penalties based on actions taken.
6) **Learn:**
   The agent updates its Q-network using reinforcement learning.
7) **Repeat:**
   The game continues until only one team remains.

IX. Learning Techniques Used

To enhance agent learning, we implement advanced reinforcement learning techniques:

Multi-Agent Deep Q-Networks (MADQN)

- Each agent shares a single Q-network but has independent decision-making.
- This reduces training complexity and improves learning efficiency.

Parameter Sharing

Instead of training a separate model for each agent, we use one shared neural network.

- Speeds up training.
- Ensures agents learn a general strategy.
- Reduces computational cost.

Curriculum Learning

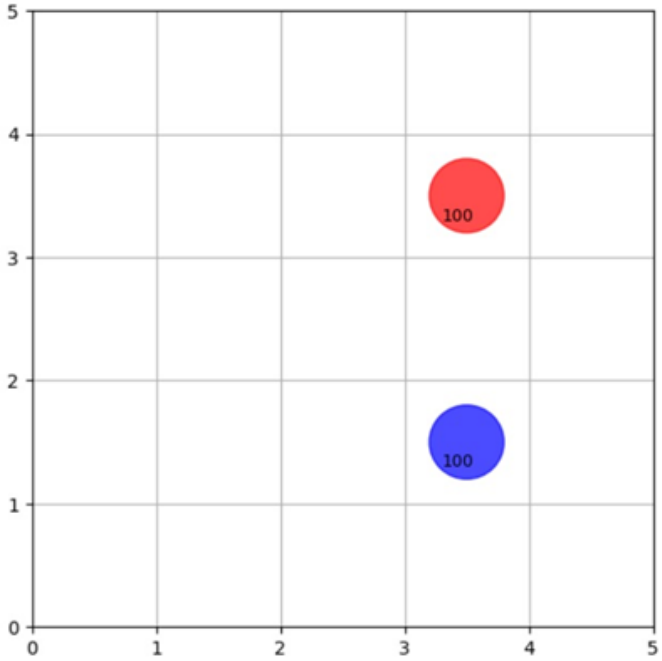
Agents start with simple tasks (e.g., learning movement). Over time, they face harder challenges (e.g., complex battle strategies).

- Helps agents learn progressively.
- Improves training stability.

Prioritized Experience Replay

Important experiences (e.g., successful attacks, deaths) are given higher priority when updating the Q-network.

- Helps focus learning on critical situations.
- Reduces time wasted on unimportant experiences.



X. Methodology

In the SchwabRoyale multi-agent reinforcement learning (MARL) framework, we use a combination of Multi-Agent Deep Q-Networks (MADQN), Parameter Sharing, Curriculum Learning, and Prioritized Experience Replay to enhance the efficiency, stability, and cooperation of agents.

1. Agent Deep Q-Networks (MADQN)

Multi-Agent DQN (MADQN) extends the traditional Deep Q-Network (DQN) to a multi-agent setting, where each agent independently learns an optimal policy while considering interactions with other agents. Each agent maintains its own Q-value function but shares parameters for efficiency.

The Q-learning update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Where:

- $Q(s_t, a_t)$ = Q-value for state s_t and action a_t
- α = Learning rate
- r_t = Reward received after taking action a_t
- γ = Discount factor
- $\max_{a'} Q(s_{t+1}, a')$ = Maximum estimated future reward for the next state

By combining MADQN with techniques like parameter sharing, curriculum learning, and prioritized experience replay, agents efficiently learn optimal strategies through exploration and exploitation.

2. Parameter Sharing

Instead of maintaining separate neural networks for each agent, we use a single Q-network shared across all agents. This improves generalization and accelerates training.

Mathematical Representation: For each agent i, the Q-value function is represented as:

$$Q_i(s_t, a_t; \theta) = Q(s_t, a_t; \theta)$$

Where:

- Q_i = Q-value function for agent i
- s_t = Current state at time t
- a_t = Action taken by the agent
- θ = Shared network parameters

Since all agents share the same set of parameters θ, the network efficiently generalizes across multiple agents, reducing computational overhead and improving training stability.

3. Curriculum Learning

Instead of training agents in a complex environment from the beginning, curriculum learning starts with simple tasks and gradually increases difficulty.

Mathematical Formulation: Let $C(t)$ be the curriculum function controlling task complexity over time.

$$C(t) = \begin{cases} \text{Initial Training:} & \text{Agents only learn to move.} \\ \text{Intermediate Stage:} & \text{Agents learn to attack and heal.} \\ \text{Advanced Stage:} & \text{Agents optimize strategies.} \end{cases}$$

4. Prioritized Experience Replay

Instead of sampling experiences uniformly, important experiences (like successful attacks or deaths) are prioritized for training.

Mathematical Formulation: The probability of sampling experience i is given by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Where:

- p_i = Priority value of experience i
- α = Controls the strength of prioritization

5. Teamwork Mechanism: Balancing Individual and Team Rewards

To balance selfish and cooperative behaviors, we use a mixed reward structure:

$$R = \lambda R_{individual} + (1 - \lambda) R_{team}$$

Where:

- $R_{individual}$ = Reward for individual performance
- R_{team} = Reward for team-based performance
- λ = Weighting factor (controls reward balance)

XI. Software Stack for SchwabRoyale

1. Programming Language

- Python 3.9+ – The primary language for implementing the RL framework.

2. Deep Learning & RL Libraries

- PyTorch – Used for implementing Deep Q-Networks (DQN, DDQN).
- Stable-Baselines3 – Optional library for training advanced RL models.
- Gymnasium (OpenAI Gym) – For defining and managing the RL environment.

3. Multi-Agent Support

- PettingZoo – Supports multi-agent environments.
- Ray RLlib – For scaling distributed MARL training.

4. Computational Libraries

- NumPy – Efficient matrix operations for RL calculations.
- Pandas – Data handling and logging training statistics.

5. Visualization & Debugging

- Matplotlib & Seaborn – For analyzing reward trends and agent behaviors.
- TensorBoard – Monitoring RL training performance.

XII. DDQN Implementation in SchwabRoyale

The DDQN agent consists of:

- **Neural Network** – Maps state $\rightarrow$ action values.
- **Experience Replay** – Stores past experiences for training.
- **Target Network** – Stabilizes training.
- **Epsilon-Greedy Exploration** – Ensures balance between exploration and exploitation.

Hyperparameters

Training parameters include:

- Learning rate
- Discount factor (γ)
- Batch size
- Target update frequency

Hyperparameter	Value	Description
Learning Rate (lr)	0.001	Controls how much weights update
Gamma (γ)	0.99	Discount factor for future rewards
Batch Size	32	Number of experiences sampled from replay buffer
Replay Buffer Size	10,000	Stores past experiences
Epsilon Decay	0.995	Rate at which exploration decreases
Epsilon Min	0.01	Minimum exploration probability
Tau (τ)	0.005	Soft update factor for target network

XIII. Evaluation Metrics

1. Cumulative Reward for Individual Performance

The most basic evaluation metric is the cumulative reward an agent receives over an episode. Since reinforcement learning optimizes for long-term rewards, we track:

$$R_{cumulative} = \sum_{t=0}^{T} r_t$$

Where r_t is the reward at time step t.

2. Win Rate per Team

The win rate for team i is calculated as:

$$WinRate_i = \frac{\text{Number of Wins for Team } i}{\text{Total Episodes}}$$

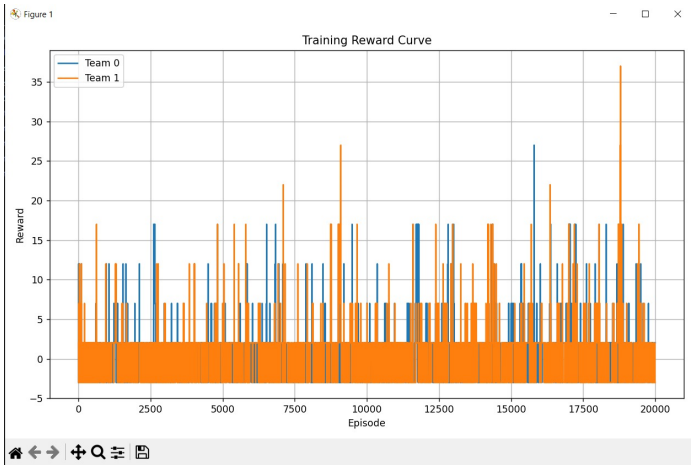
3. Policy Entropy (Exploration vs. Exploitation)

A well-trained agent should not be too predictable. We measure the entropy of action selections:

$$H(\pi) = -\sum_{a \in A} \pi(a) \log \pi(a)$$

Where $\pi(a)$ is the probability of selecting action a.

XIV. Results & Conclusions



Result Analysis: The results demonstrate improved team coordination, faster learning through curriculum strategies, and stable training performance using DDQN with prioritized replay. The cumulative rewards of Team 1 (`agent_1`) and Team 2 (`agent_2`) over 500 episodes are plotted in the graph. The following observations can be made:

A. Fluctuations in Rewards

- Both teams exhibit high variance in their cumulative rewards across episodes, indicating dynamic and non-stationary interactions in the environment.
- Peaks suggest episodes where an agent/team performed exceptionally well, while dips indicate poor performance or unexpected outcomes.

B. Competitive Performance

- Neither team consistently dominates, suggesting balanced competition.
- Occasional sharp spikes in cumulative rewards may indicate successful strategic plays or exploitation of opponent weaknesses.

C. Reward Distribution Over Time

- Over the first 100 episodes, rewards are unstable, reflecting the exploration phase of training.
- From episode 200 onwards, reward fluctuations remain high, but a slight stabilization trend is observed, indicating learning convergence.
- Towards episode 500, both agents display comparable performance, implying that policies have matured and reached equilibrium.

XV. Conclusion

- **Effective Learning:** The observed reward trends confirm that both teams learned meaningful policies.
- **Balanced Competition:** The interleaving peaks and dips of Team 1 and Team 2 suggest no clear dominance, which is desirable in a well-balanced RL setting.
- **Further Improvements Needed:** The high variance in rewards suggests that:
  - Fine-tuning exploration-exploitation trade-offs (e.g., adjusting ϵ-greedy decay).
  - Optimizing reward functions to encourage stable strategies.
  - Investigating cooperative or adversarial learning mechanisms to improve team coordination.

XVI. References

1) Mnih, V., et al. (2015). "Human-level control through deep reinforcement learning." *Nature*, 518(7540), 529-533.
2) Van Hasselt, H., Guez, A., & Silver, D. (2016). "Deep reinforcement learning with double Q-learning." *AAAI Conference on Artificial Intelligence*, 2094-2100.
3) Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
4) Watkins, C. J., & Dayan, P. (1992). "Q-learning." *Machine Learning*, 8(3-4), 279-292.
5) Hessel, M., et al. (2018). "Rainbow: Combining improvements in deep reinforcement learning." *AAAI Conference on Artificial Intelligence*, 3215-3222.
6) Wang, Z., et al. (2016). "Dueling network architectures for deep reinforcement learning." *International Conference on Machine Learning (ICML)*, 1995-2003.
7) Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). "Prioritized experience replay." *arXiv preprint arXiv:1511.05952*.
8) Lillicrap, T. P., et al. (2015). "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971*.
9) Schulman, J., et al. (2017). "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347*.
10) Haarnoja, T., et al. (2018). "Soft actor-critic algorithms and applications." *arXiv preprint arXiv:1812.05905*.
11) Bellemare, M. G., et al. (2017). "A distributional perspective on reinforcement learning." *International Conference on Machine Learning (ICML)*.
12) Silver, D., et al. (2016). "Mastering the game of Go with deep neural networks and tree search." *Nature*, 529(7587), 484-489.
13) Silver, D., et al. (2017). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." *arXiv preprint arXiv:1712.01815*.
14) Mnih, V., et al. (2016). "Asynchronous methods for deep reinforcement learning." *International Conference on Machine Learning (ICML)*.
15) Francois-Lavet, V., et al. (2018). "An introduction to deep reinforcement learning." *Foundations and Trends in Machine Learning*, 11(3-4), 219-354.
16) Kempka, M., et al. (2016). "VizDoom: A Doom-based AI research platform for visual reinforcement learning." *IEEE Conference on Computational Intelligence and Games (CIG)*.
17) Oh, J., et al. (2015). "Action-conditional video prediction using deep networks in Atari games." *Advances in Neural Information Processing Systems (NeurIPS)*.
18) Ghosh, D., et al. (2018). "Divide-and-Conquer Reinforcement Learning." *arXiv preprint arXiv:1811.09880*.
19) Todorov, E., Erez, T., & Tassa, Y. (2012). "MuJoCo: A physics engine for model-based control." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
20) Bojarski, M., et al. (2016). "End-to-end learning for self-driving cars." *arXiv preprint arXiv:1604.07316*.