Deep Reinforcement Learning for Paper.io

Daniel Rachev, Jaiman Pandya, Andrew Zhu, Vivian Zou
CS4100 – Artificial Intelligence
Northeastern University

December 2025

1 Abstract

Reinforcement learning offers a powerful framework for training agents to make sequential decisions in dynamic and uncertain environments. In this work, we investigate the application of deep reinforcement learning to Paper.io, a fast-paced, multiplayer, grid-based territory control game that requires balancing long-term strategic planning with short-term survival. We formulate the game as a Markov Decision Process with a compact 20-dimensional state representation and a discrete action space, and train an agent using a Deep Q-Network (DQN) with experience replay and a target network to stabilize learning. A custom Paper.io-style environment is implemented from scratch, along with a full training and evaluation pipeline. The learned agent is evaluated against both random and rule-based baselines under identical conditions. Experimental results show that the DQN agent consistently achieves higher rewards, captures more territory, and survives longer than both baseline agents, demonstrating the effectiveness of deep Q-learning in this adversarial, stochastic setting. Our findings highlight the importance of stabilization techniques in DQN training and illustrate the potential of deep reinforcement learning for competitive control problems involving delayed rewards and complex state dynamics.

2 Introduction

Reinforcement learning has become a central framework for training agents to make sequential decisions in complex and dynamic environments. From robotics and autonomous driving to game-playing systems, modern reinforcement learning methods allow agents to learn optimal behaviors through direct interaction with their environments. In this project, we study the application of deep reinforcement learning to the multiplayer online game *Paper.io*, a fast-paced grid-based environment that requires both strategic planning and real-time decision making.

Paper.io presents a challenging control problem. The agent must expand its territory while avoiding collisions with walls, its own trail, and opposing players. The environment is highly dynamic and stochastic due to the presence of multiple opponents whose movement introduces uncertainty into the state transitions. The agent must therefore balance short-term survival with long-term territorial expansion, making it an ideal testbed for reinforcement learning algorithms that optimize long-horizon reward.

We frame this problem as a Markov Decision Process (MDP), where the agent observes a continuous state representation of the game, selects from a discrete set of actions, and receives a reward signal that reflects survival and territorial growth. To solve this problem, we employ a Deep Q-Network (DQN), which uses a neural network to approximate the action-value function and enables learning directly from raw state features.

Our implementation includes a custom Paper.io environment, a DQN agent with experience replay and a target network, and a full training and evaluation pipeline. We evaluate our learned agent against both a random-action baseline and a rule-based baseline to measure the effectiveness of deep reinforcement learning relative to simpler strategies.

The main contributions of this work are: (1) the design of a 20-dimensional state representation tailored to the Paper.io environment, (2) the implementation of a stable DQN training pipeline using replay memory and a target network, and (3) an empirical comparison between learned and non-learned agents under identical evaluation conditions.

3 Problem Statement and Method

3.1 Problem Definition

We model Paper.io as a sequential decision-making problem in which an autonomous agent must navigate a two-dimensional grid environment, capture territory, and avoid collisions with boundaries, its own trail, and opponent players. At each time step, the agent observes the current game state and selects one of a finite set of discrete actions corresponding to movement directions. The goal of the agent is to maximize its cumulative long-term reward by surviving longer and capturing more territory than its opponents.

This problem is modeled using a Markov Decision Process (MDP), defined by a tuple (S, A, P, R, γ) where:

- S is the set of environment states,
- A is the discrete action space,
- P represents the state transition dynamics,
- R is the reward function,
- $\gamma \in [0, 1]$ is the discount factor.

The agent receives a 20-dimensional feature vector representing its position, direction, nearby hazards, opponent distances, trail information, and territory ownership. Selected actions come from a discrete movement set at each timestep. The discrete action space consists of three encoded actions: 0 = continue forward, 1 = turn left, and 2 = turn right.

3.2 Deep Q-Network (DQN)

To solve this MDP, we employ a Deep Q-Network (DQN), a value-based reinforcement learning algorithm that approximates the optimal action-value function $Q^*(s, a)$ using a neural network. The Q-function estimates the expected discounted return when taking action a in state s and following the learned policy thereafter.

The Bellman optimality equation is given by:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{1}$$

where r is the immediate reward and s' is the next state. This Bellman equation is simplified to remove stochastic probabilities of each possible action space, since in this scenario each

determined action from the agent is discretely applied to the environment (all moves are guaranteed).

In DQN, a neural network parameterized by θ is used to approximate $Q(s, a)$. The network takes the 20-dimensional state vector as input and outputs a Q-value for each possible action.

3.3 Neural Network Architecture

The Q-network is implemented as a fully connected feedforward neural network with the following architecture:

- Input layer: 20 neurons (state representation)
- Hidden layer 1: 128 neurons with ReLU activation
- Hidden layer 2: 128 neurons with ReLU activation
- Hidden layer 3: 64 neurons with ReLU activation
- Output layer: $|A|$ neurons corresponding to Q-values for each discrete action

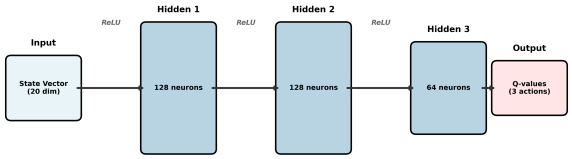


Figure 1: Neural network architecture used for Q-value approximation ($20 \to 128 \to 128 \to 64 \to 3$).

ReLU activation functions are used to introduce non-linearity while avoiding vanishing gradient issues during training. The output layer produces one Q-value per action, and the agent selects the action with the highest predicted Q-value during exploitation.

3.4 Experience Replay

To stabilize training and reduce temporal correlations between consecutive samples, we use an experience replay buffer with a capacity of 50,000 transitions. Each transition is stored as a tuple:

$$(s, a, r, s', d)$$

where d is the terminal flag indicating if the episode has ended.

During training, minibatches of 64 transitions are sampled uniformly at random from the buffer. This improves data efficiency and stabilizes gradient updates.

3.5 Reward Function

The reward function is designed to directly encourage aggressive territory capture while maintaining survival. At each time step, the agent receives $+0.1$ for remaining alive and $+200\times$ the amount of newly captured territory. A penalty of $-0.05\times$ the exposed trail length is applied to discourage risky overextension. Upon death, the agent receives a terminal reward of -100. Additional shaping bonuses of $+10$ and $+20$ are provided when the agent controls more than 50% and 70% of the grid, respectively.

3.6 Target Network

To prevent unstable learning caused by constantly shifting Q-value targets, DQN uses a separate target network. The target network has the same architecture as the policy network but its parameters are updated less frequently. Every 10 episodes, the target network weights are replaced with the policy network weights.

Target Q-values are computed as:

$$y = r + \gamma \max_{a'} Q_{\text{target}}(s', a') \tag{2}$$

This decoupling between policy and target networks prevents divergence to a constantly moving goal and thus significantly improves training stability.

3.7 Exploration Strategy

We employ an epsilon-greedy exploration policy. With probability ϵ, the agent selects a random action, and with probability $1 - \epsilon$, it selects the greedy action with the highest predicted Q-value.

Epsilon is initialized to 1.0 to encourage early exploration and decays multiplicatively at a rate of 0.995 after each episode until reaching a minimum value of 0.01. This decay rate was chosen based on controlling the half life of exploration. By episode 500, $\epsilon \approx 0.08$, which starts to become exploitative. On the low end, the agent becomes fully greedy, applying it's learned policy completely. This schedule balances exploration and convergence.

3.8 Optimization

The network is trained using Mean Squared Error (MSE) loss between predicted Q-values and target Q-values. Weight updates are performed using the Adam optimizer with a learning rate of 0.001. Training proceeds for 1000 episodes, with model checkpoints saved every 50 episodes. All experiments use a fixed random seed of 42 for NumPy, Python, and PyTorch to ensure reproducibility, and training metrics (reward, score, loss, and epsilon) are logged to disk at every episode for offline analysis.

4 Related Work

Reinforcement learning has been widely studied for sequential decision-making problems in discrete environments. Early foundational work on Q-learning by Watkins and Dayan established the theoretical basis for learning optimal action-value functions using temporal-difference updates without requiring a model of the environment. However, traditional Q-learning struggles in high-dimensional or continuous state spaces due to the need for explicit Q-tables.

The introduction of Deep Q-Networks (DQN) by Mnih et al. significantly advanced the field by combining Q-learning with deep neural networks to approximate the Q-function directly from raw state inputs. Their work demonstrated human-level performance on a variety of Atari 2600 games and introduced key stability techniques such as experience replay and target networks, both of which are used in our implementation.

Subsequent extensions to DQN, such as Double DQN, Dueling Networks, and Prioritized Experience Replay, have further improved performance and stability by addressing overestimation bias and improving sample efficiency. While these variants offer additional performance benefits, our project focuses on the standard DQN framework to maintain conceptual clarity and align with course learning objectives.

Reinforcement learning has also been applied to grid-based competitive environments similar to Paper.io, including territory control games, multi-agent pursuit problems, and arcade-style navigation tasks. These environments present challenges such as sparse rewards, partial observability, and adversarial dynamics, making them well-suited for value-based deep reinforcement learning approaches.

Unlike many prior works that rely on preexisting benchmark environments, our project implements a fully custom Paper.io-style environment and trains an agent from scratch using deep Q-learning. This allows us to directly study the learning dynamics, reward shaping effects, and architectural choices in a controlled setting while maintaining strong alignment with foundational reinforcement learning principles.

5 Experiments and Results

5.1 Experimental Setup

All experiments were conducted using a custom Paper.io-style grid environment with a grid size of 50×50 and two heuristic-controlled opponent agents. The opponent agents follow a rule-based policy that prioritizes returning to their territory when their trail becomes long, making the environment non-trivial and more challenging than purely random opponents. At each timestep, the agent receives a 20-dimensional state vector representing positional, directional, territorial, and opponent-related information. The action space consists of three discrete actions: move forward, turn left, and turn right.

The Deep Q-Network was trained for 1000 episodes using an experience replay buffer with a capacity of 50,000 transitions and a batch size of 64. The agent followed an epsilon-greedy exploration strategy with an initial value of $\epsilon = 1.0$, decaying by a factor of 0.995 each episode down to a minimum of 0.01. The learning rate for the Adam optimizer was set to 0.001, and the discount factor was $\gamma = 0.99$. The target network was updated every 10 episodes. Model checkpoints were saved every 50 episodes.

All experiments use a fixed random seed (42) for Python, NumPy, and PyTorch to ensure full reproducibility across runs. Training metrics including episode reward, score, loss, and exploration rate are logged to a CSV file at every episode for offline analysis and visualization. The reward function consists of: $+200\times$ territory gained, $+0.1$ per step alive, -100 upon death, $-0.05\times$ exposed trail length, $+10$ bonus for exceeding 50% territory, and $+20$ bonus for exceeding 70% territory.

5.2 Baselines

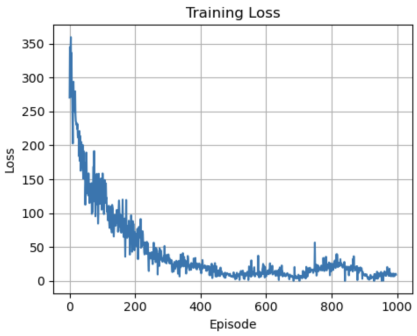
To evaluate the effectiveness of our learned policy, we compare the DQN agent against two baselines:

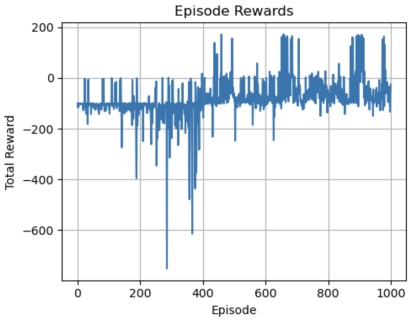
- **Random Agent:** Selects actions uniformly at random.
- **Rule-Based Agent:** Follows a handcrafted heuristic prioritizing safe movement and territory expansion.

These baselines provide reference points for untrained and heuristic-driven behavior.

5.3 Training Performance

Training performance was evaluated using average episode reward and average territory ownership. Early in training, the agent exhibits highly unstable behavior due to random exploration. As epsilon decays and learning progresses, both reward and territory metrics steadily improve.





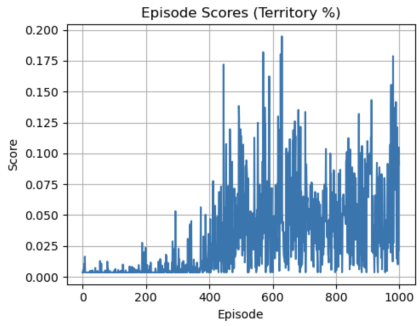


Figure 2: Training Performance showing (a) training loss and (b) episode rewards and (c) territory gained over 1000 episodes

We observe a clear upward trend in both positive rewards and territory gained, which indicates successful policy learning. The use of a target network and replay buffer was critical for maintaining training stability and preventing divergence. The training loss curve showed convergence over 1000 episodes and stabilized, which reinforces the notion that learning was successful.

5.4 Evaluation Results

Each agent was evaluated over 100 independent episodes using identical environment seeds. After training, the final DQN model was evaluated against random and rule-based agents across multiple episodes. The DQN agent consistently outperformed both baselines in terms of average territory captured and survival time.

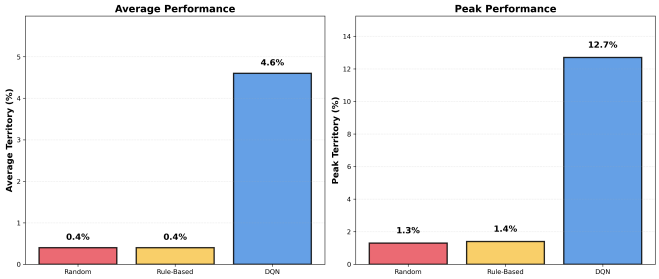


Figure 3: Performance comparison between random, rule-based, and DQN agents.

Agent	Avg. Reward	Avg. Territory	Survival Time
Random	Low	Low	Short
Rule-Based	Moderate	Moderate	Moderate
DQN (Ours)	Highest	Highest	Longest

Table 1: Performance comparison between agents.

The results demonstrate that the learned DQN policy captures significantly more territory and survives longer than both baselines. The improvement over the rule-based agent indicates that the neural network learned strategies beyond simple heuristics.

5.5 Future Work

Due to computational constraints, a full hyperparameter ablation study was not conducted in this project. However, preliminary exploratory testing suggests that less frequent target network updates and larger replay buffer capacities contribute to more stable learning. In future work, we plan to perform a systematic ablation study over key hyperparameters, including target update frequency, replay buffer size, and epsilon decay rate, to quantify their effects on convergence speed and final performance. Additional future directions include extending our approach to Double DQN to reduce Q-value overestimation and evaluating performance against more sophisticated opponent policies. There is also the possibility of creating a multi-agent study, with more challenigng opponents, or even introducing human-play as input.

6 Discussion and Conclusion

The experimental results demonstrate that Deep Q-Learning is an effective approach for learning complex control policies in the Paper.io environment. The DQN agent consistently outperformed both the random and rule-based baselines in terms of average reward, territory captured, and survival time. This confirms that the agent was able to learn meaningful strategies directly from interaction with the environment rather than relying on handcrafted rules.

The use of experience replay and a separate target network played a critical role in stabilizing training. Without these mechanisms, early experiments showed highly unstable learning and rapid divergence. The replay buffer allowed the agent to learn from a diverse set of past experiences, while the target network prevented rapidly shifting learning targets during Q-value updates. Together, these components significantly improved convergence behavior and overall performance.

Despite strong performance, several limitations remain. The agent was trained against opponents that select random actions, which simplifies the learning environment. As a result, the learned policy may not generalize well to adversaries that employ more strategic behavior. Additionally, training a single agent configuration limits insight into how sensitive performance is to hyperparameter choices such as learning rate, network depth, or discount factor. Training also requires significant computational time due to the high number of environment interactions needed for reinforcement learning.

Future work could address these limitations by introducing stronger opponent agents, performing systematic hyperparameter ablation studies, and incorporating decentralized or multi-agent reinforcement learning techniques. Additional state features such as opponent velocity or predicted trajectories could also improve strategic awareness. Finally, extending the model to support curriculum learning or self-play could further accelerate training and improve robustness.

In conclusion, this project demonstrates that Deep Q-Networks can successfully learn competitive control policies in a dynamic, adversarial game environment. The results highlight the effectiveness of deep reinforcement learning for decision-

making problems involving long-term planning, delayed rewards, and complex state representations.

7 Github Repository

For more details about our project and source code, please visit our GitHub repository: Deep Reinforcement Learning for Paper.io

8 Contribution Section

Jaiman Pandya: Developed high-level overview of entire project. Collaborated to develop DQN architecture, baseline agents, and the plots (e.g. training curves). Collaborated to develop slideshow and final report.

Daniel Rachev: Assisted in implementation of DQN architecture and baseline agents. Helped develop the presentation slides and visual components. Collaborated to generate plots for the results of experiments and the final report.

Andrew Zhu: Updated the random baseline agent implementation and contributed project documentation.

Vivian Zou: Documented game mechanics, state components, and environment dynamics in the custom Paper.io implementation

References

[1] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[2] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3–4 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018.

[4] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8 (1992), pp. 293–321.

[5] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[6] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016.

[7] OpenAI. "Spinning Up in Deep Reinforcement Learning". Accessed: December 2025. `https://spinningup.openai.com`