I implemented and refined a Deep Q-Network (DQN) agent for the Atari Bowling environment across two major iterations. In both implementations, I used a convolutional neural network to process raw pixel inputs, converting game frames to grayscale, resizing to 84×84 pixels, and stacking 4 consecutive frames to capture motion. For my first implementation, I used a standard DQN architecture with three convolutional layers followed by fully connected layers, experience replay with a buffer size of 100,000 transitions, and a target network updated periodically. Despite coding an epsilon-greedy strategy with a decay schedule (starting at 1.0 and decaying to 0.1), a bug resulted in a fixed epsilon value of 0.10 throughout training. After analyzing the initial results, I modified several hyperparameters for my second implementation, including adjustments to the learning rate, batch size, and target network update frequency, while still working with the fixed epsilon constraint. The agent was trained for 200 episodes across two implementation iterations.

My approach evolved based on the challenges of the Bowling environment and observations from early training. I initially chose DQN because it is specifically designed for high-dimensional visual inputs, allowing the agent to learn directly from pixels without manual feature engineering. The key components (convolutional layers, experience replay, target network) address specific challenges in reinforcement learning for complex environments. After observing limited learning progress in my first implementation, I recognized that the bug fixing epsilon at 0.10 was limiting exploration– thus, even though I had coded a standard epsilon-greedy strategy with a decay schedule starting at 1.0 (100% exploration) and gradually decreasing to 0.1 with a decay rate of 0.995 per episode–epsilon remained fixed at 0.10 throughout the entire training process. This meant the agent consistently selected random actions only 10% of the time and followed its learned policy 90% of the time from the very beginning. My initial implementation demonstrated limited learning progress, as shown in Figure 1. The learning curve displayed a relatively flat moving average around 27-30 points across 200 episodes, with no clear improvement trend. Despite the high variance in individual episode rewards (ranging from 10 to approximately 60 points), the moving average remained stagnant. The loss value quickly stabilized around 0.011-0.012, and the reward distribution had a mean of 28.14.

After discovering this issue, I had two choices: restart training with a corrected decay schedule or adjust other parameters while maintaining the fixed epsilon to investigate whether learning could improve despite the limited exploration. I attempted to correct the decay schedule code, but ultimately struggled to correct this issue and chose the latter approach to better understand how different components of the DQN algorithm influence learning under exploration constraints. Heavily favoring exploitation from the start created a challenging learning scenario that forced the agent to extract maximum value from its limited exploratory actions. My decision to investigate whether hyperparameter tuning could improve performance even under this constraint was motivated by the observation that even with limited exploration, the agent occasionally achieved high scores, suggesting that the bowling environment has some learnable patterns accessible even with minimal exploration. This two-phase approach allowed me to better understand how different components of the DQN algorithm affect performance in precision-timing tasks like bowling.

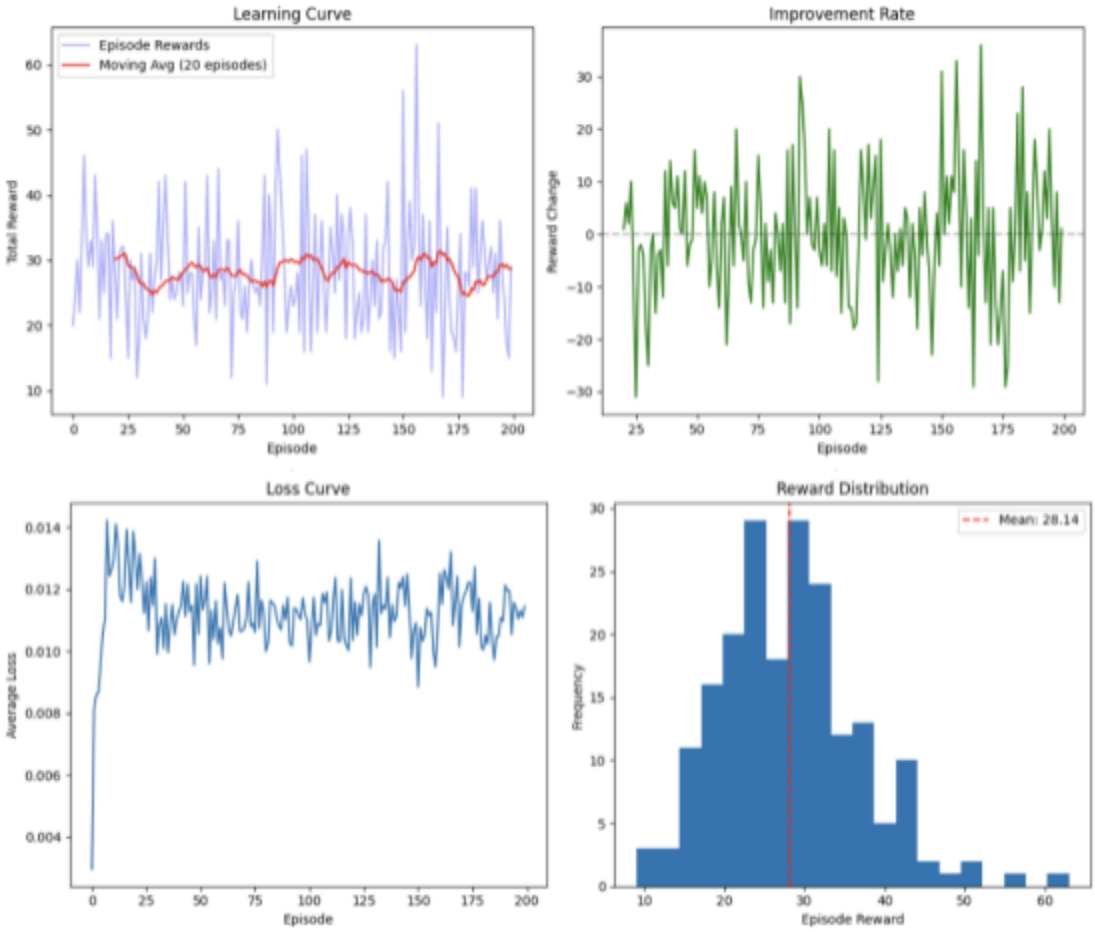

Figure 1. Initial DQN

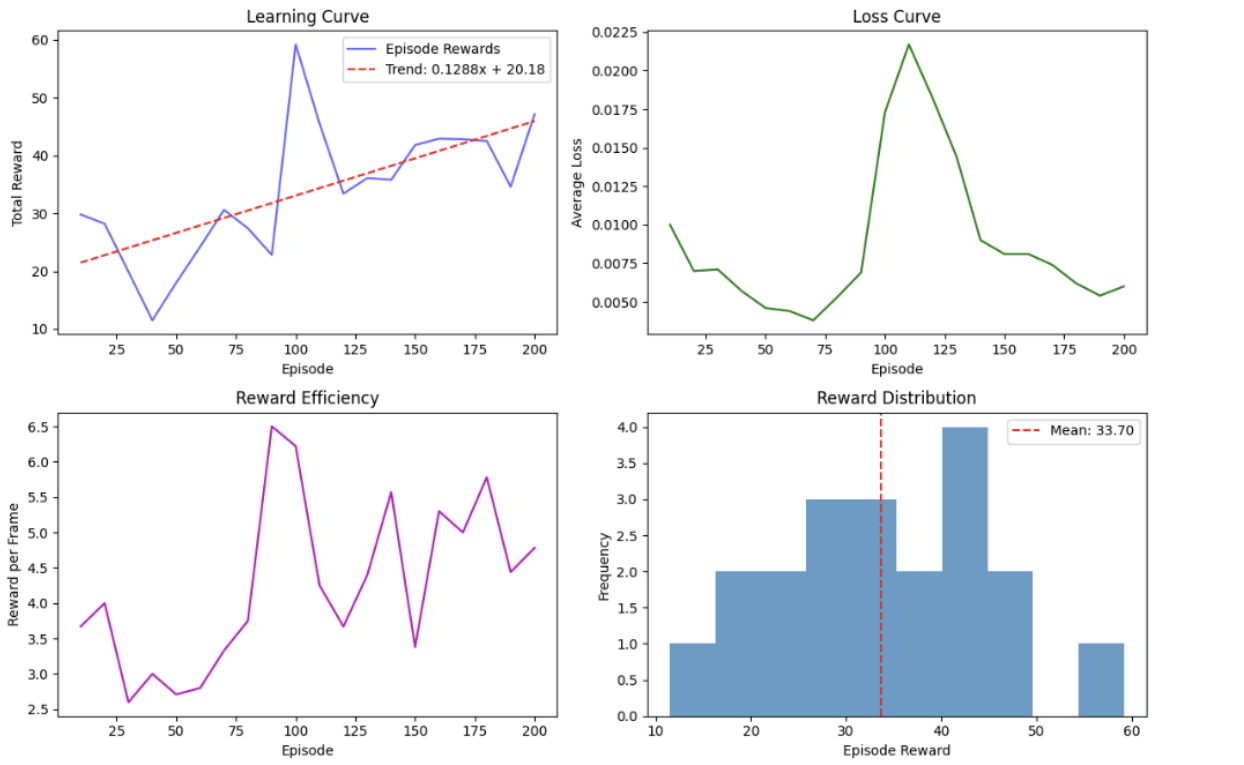

Figure 2. Final DQN

After hyperparameter modifications in my second implementation, I observed markedly different results. The agent showed clear improvement over 200 episodes, with a positive trend line having a slope of 0.1288. The agent progressed from scores around 20 points in early episodes to consistently achieving 40-47 points in later episodes, with a notable performance spike around episode 100 reaching nearly a score of 60. The mean reward improved to 33.70, and the reward efficiency (points per frame) increased from approximately 3 in early episodes to over 5 in later episodes. The loss curve showed a more dynamic pattern, with an initial decrease followed by a significant spike coinciding with the performance breakthrough, and eventual stabilization at a lower level.

The stark contrast between my two implementations reveals important insights about DQN for precision-timing tasks like bowling– and show the challenges the agent faced during training. My initial implementation's flat performance curve despite high variance suggests that the agent was discovering effective strategies but failing to consistently reproduce them. The agent only received meaningful feedback after the ball reached the pins, creating a substantial delay between the critical ball release action and the resulting score. This delay made it extremely difficult for the agent to identify which specific actions led to successful outcomes, as evidenced by the high variance in performance between episodes. The fixed epsilon value of 0.10 further compounded this problem by severely restricting the agent's ability to explore the state-action space, particularly in early training. This was especially problematic because bowling requires discovering precise timing and positioning that are unlikely to be found through

limited random actions. Unlike many Atari games where approximate actions often yield reasonable results, bowling demands exact timing for optimal performance. The pixel-based representation made this precision challenge even harder, as the agent had to learn subtle visual cues indicating the optimal release point. Additionally, the agent would occasionally discover highly effective strategies (as seen in the episode 100 spike in my second implementation) but then "forget" these strategies as it incorporated new experiences. This catastrophic forgetting occurred because in DQN, updating Q-values for some state-action pairs can unintentionally alter the policy for other states–a problem exacerbated by bowling's sparse reward structure where successful experiences were rare.

Despite these challenges, several factors made successful training feasible. The convolutional neural network architecture was crucial, as it enabled the agent to extract relevant spatial features from the raw pixel inputs. Without this capability, the agent would have been unable to identify important visual patterns like ball position, lane orientation, and pin layout. The convolution operations allowed the agent to focus on relevant game elements while ignoring background details. Experience replay was essential for efficient learning in a sparse reward environment like bowling. By storing and reusing rare successful experiences, the agent could learn from its infrequent wins multiple times—particularly important given the limited exploration that reduced the discovery of new successful strategies. In my second implementation, hyperparameter tuning proved critical for extracting value from limited exploration. A lower learning rate provided more stable updates, while more frequent target network updates allowed the agent to incorporate successful strategies more quickly. These adjustments explain why my second implementation showed clear improvement despite the fixed epsilon constraint. Frame stacking was another key factor, as combining four consecutive frames provided temporal information essential for understanding ball trajectory and movement. Without this, the agent would have seen only static images, making it impossible to learn the dynamics necessary for effective bowling.

Several approaches could have potentially improved performance but were not implemented. Most critically, I couldn't implement a proper epsilon decay schedule due to the bug in my code. This significantly limited the agent's ability to explore in early training when it needed to discover fundamental strategies from scratch. Proper exploration is essential in reinforcement learning because the agent needs to discover effective strategies before it can exploit them, and the fixed epsilon greatly hindered this discovery process. I also couldn't implement prioritized experience replay due to implementation complexity and computational constraints. This technique would have helped the agent learn more efficiently from important transitions by sampling them more frequently, which would have been particularly valuable given the sparse successes in bowling.

More sophisticated network architectures might have improved performance by separately modeling state values and action advantages or by learning the distribution of returns rather than just the mean, but I couldn't implement these due to time constraints and the significant increase in computational requirements they would have introduced. Finally, I couldn't implement reward shaping to provide intermediate rewards that would have guided the agent toward good bowling techniques. This approach would have required domain knowledge to design appropriate shaping rewards and would have moved away from the pure end-to-end learning paradigm of DQN. Despite these limitations, my second implementation demonstrated

that even a simple DQN with proper hyperparameter tuning can learn reasonably effective bowling strategies, as evidenced by the positive trend line in the learning curve and the improvement from approximately 20 points in early episodes to 40-47 points in later episodes.

In conclusion, this project demonstrated both the capabilities and limitations of deep reinforcement learning for tasks requiring precise timing and actions with delayed rewards. The DQN agent's ability to show meaningful improvement despite suboptimal exploration highlights the robustness of the algorithm when properly tuned. However, the challenges encountered, particularly the difficulty in consistently reproducing peak performance, illustrate why precision tasks like bowling remain difficult for reinforcement learning agents. The most valuable insight from this implementation was that while exploration is crucial for reinforcement learning, thoughtful hyperparameter tuning can partially compensate for exploration limitations. Moving forward, combining proper exploration schedules with the hyperparameter optimizations discovered in this project would likely yield even stronger performance.