Cassy Cormier

Capstone Project: Train 1an AI Agent to Play Flappy Bird

15698 Computer Vision 1378

Professor Anna Devarakonda

April 20th, 2025

Evaluating Reinforcement Learning Performance in a Flappy Bird Environment

Reinforcement learning (RL) is a machine learning (ML) task where an AI agent interacts with its environment and learns in the form of rewards or penalties. An AI agent is an autonomous entity that interacts with its environment to achieve a specific goal. The agent will learn to perform tasks through a trial, error, and reward system (Murel Ph.D. & Kavlakoglu, 2024). One specific problem tackled in this project was training an AI agent to play Flappy Bird using a Deep Q-Network (DQN). The goal was to enable the agent to learn how to navigate the game's pipe obstacles through trial and error by maximizing its reward. Key challenges faced during this project include game development issues, coding errors, exploration strategy failures, time constraints, and no learning evidence. This project implements RL using a PyGame Learning Environment through experience replay and target network updates. A DQN agent is a core component of my approach to solving the Flappy Bird challenge. The agent was trained using a combo of exploration-exploitation techniques and reward shaping. This reflection will explore how well the agent performed over time and what challenges and breakthroughs emerged along the way.

Computer vision (CV) is a subset of Deep Learning (DL). Computer vision played a central role in allowing the agent to understand and manipulate images in the environment using its pixel data. CV empowers the agent to make better decisions and perform enough jumps to gain more rewards as if a human was playing. The computer vision methods used include preprocessing, feature extraction, scene understanding and object detection techniques (Reinforcement Learning, 2025).

Image processing algorithms were used for the purpose of interpreting images to allow the agent to extract meaningful information from the environment. One of those preprocessing techniques included resizing the frame to reduce computation while preserving key game features (Patel, 2023). Also, I scaled down the pixel values through normalization to speed up training and improve stability. I chose to run the models both with and without grayscale conversion for the purpose of seeing how the agent would perform. Feature extraction allows for the relevant information in the scene, such as edges, corners, textures, and so much more, to be identified by the agent. Enabling this lowers the model complexity and allows the agent to focus on the structure, and not the color (Feature Extraction, n.d.). Object detection algorithms were implemented so that the agent can start locating and classifying objects within the game environment.

To begin the implementation, I installed necessary libraries and set up the Flappy Bird environment. To support the development of my project, I relied upon a combination of machine learning and simulation libraries such as NumPy, TensorFlow, PyGame, and OpenCV. The project

leveraged Keras for its high-level neural network processing abilities and Gym for building and interacting with the Flappy Bird environment. A local package was installed in editable mode (-e) to streamline iterative development. I utilized ChatGPT and ClaudeAI to assist me with coding and debugging. Once data acquisition was complete, I implemented a preprocessing function to resize and normalize game frames. Next, I loaded a pre-trained MobileNetV2 model after researching this model would be best for the computational constraints I have. I modified the model for feature extraction by removing the top layer and adding custom layers for the Q-network.

This was followed by the implementation of a DQNAgent with an initial setup that included initializing replay memory, setting the exploration rate, and building the Q-network using a pre-trained model. The agent was further equipped with vital methods such as remembering to store experiences, acting to select actions via an epsilon-greedy policy, and replaying to enable experience replies during training. Additionally, a target network was implemented with periodic updates to stabilize learning. After completing that, I set up the training loop to interact with the Flappy Bird environment with frame skipping for efficiency. This step included storing experience, performing learning steps, and periodically updating the target network. I implemented code for a decay schedule for the exploration rate and to save the model checkpoints periodically. Throughout the process, metrics were recorded such as average score, survival time, and average steps. This was implemented alongside a separate testing script to evaluate the trained agent. By following these steps, I was able to visualize the agent's performance using Matplotlib and compare the performance of various models.

To evaluate the performance of my AI agent, I employed a set of quantitative metrics that reflect both effectiveness and consistency. The primary metrics include the average score across multiple episodes, maximum score achieved, inference time per frame, and frame skips. These measures provide insight into the agent's ability to consistently achieve high performance and its stability during inference. The agent evaluated over 500 independent episodes using a fixed environment configuration. All episodes were initialized with the same game settings, and no training updates were applied during evaluation to ensure unbiased performance. The results of all 5 trained models are summarized in Table 1.

Table 1

Test	Episodes	Best Reward	Avg Reward (10)	Avg Steps (10)
Test 1	500	0.00	0.00	0
Test 2	65	-3.00	-	16
Test 3	100	5.60	-	—
Test 4	100	17.10	1.91	22
Test 5	500	15.10	2.26	18.80
Test 6	500	15.30	2.32	9.50

The agent received an average reward of 2% with a maximum of 17.30, demonstrating a robust performance. Overall, the training trajectory reflects a successful learning process, with clear signs of convergence and improved performance. These trends suggest that the model effectively adapted to the environment achieving reliable performance over time. The first test showed concerning signs of issues with the Epsilon decay, with it being stuck at 1.0 and the agent

gaining no rewards. There were zero steps recorded which means the episodes were terminating immediately. There were coding bugs in the step function.

The second model initially struggled due to bugs in the step function, which required immediate attention. Most episodes resulted in a reward of -5.00, with occasional improvements to -4.00 or -3.00, likely outcomes of random exploration rather than effective learning. Although epsilon successfully decayed to 0.1, indicating that the exploration phase completed as intended, the overall improvement was not meaningful. This suggests the need to revisit the reward function for further refinement.

In contrast, the third test marked a significant turning point. Early in training, the agent achieved a reward of 5.60—a substantial leap that indicated it managed to pass four or five pipes and survive longer. At this point, I introduced a system to track performance trends and automatically save the best-performing model based on reward metrics. This shift in approach marked the transition from random behavior to purposeful gameplay, as the agent began to learn the dynamics of the environment more effectively.

To support this progress, I increased the number of episodes from 100 to 500 and adjusted the learning rate and batch size to enhance neural network performance. Following these changes, the model began to stabilize, with reward values fluctuating between 1.10 and highs of 7.90, 10.10, and even 15.00, as documented in the table below. Despite these gains, the model still occasionally reversed to a reward of 1.10, indicating that further tuning is required to improve consistency and performance.

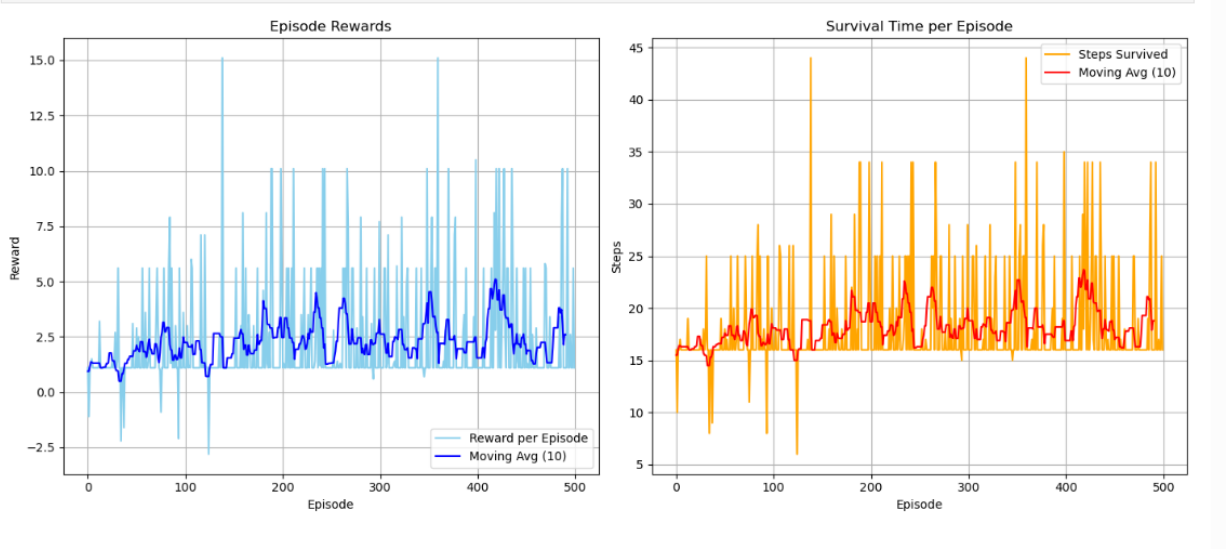
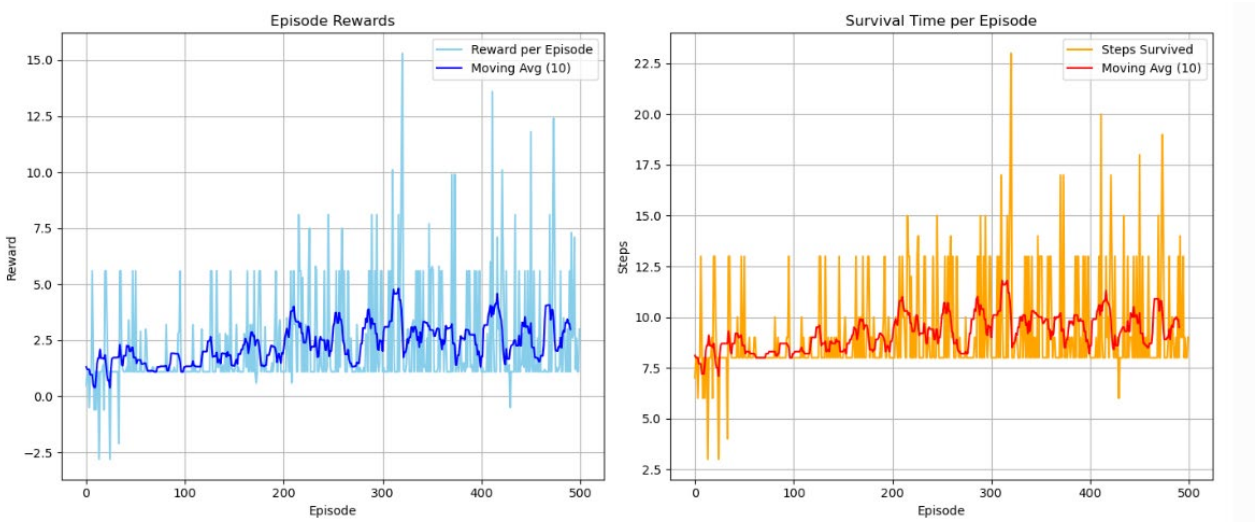


Table 5

Compared to the other models, test four, five, and six demonstrated faster convergence and higher reward stability, indicating learning efficiency. As demonstrated in table 2, model 5 appears to have a higher and more consistent survival time, with a clear upward trend in the moving average. Taken together, these results suggest that while both models demonstrated learning progression, the final few models exhibited the strongest signs of convergence, with more stability and sustained improvements in survival time.

Table 3, Test 6



Early on, even setting up the game environment posed issues—requiring a full restart of the runtime session in Google Colab to resolve execution errors. I relied heavily on ChatGPT as a coding assistant, especially after choosing the custom implementation path, which inevitably brought with it several coding errors. These bugs often forced me to rerun the entire program from scratch—a frustrating obstacle when working under tight deadlines. Still, challenges like reward stagnation, sparse feedback, and inconsistent episode performance remained, especially in the mid-training phases. Nonetheless, the agent has clearly moved past the "just dying" phase into one of genuine strategic play, capable of consistent pipe-passing, survival, and even recovery after setbacks—proving that progress, while non-linear, is certainly taking flight.

Comparing test one through six, we see a clear learning curve, where test one (the baseline) starts at 0 reward, meaning the agent was failing quickly. By test six, reward performance and consistency greatly improved. Test six shows the highest average reward (2.32) and the highest best reward (15.30), indicating a consistent learning agent. It achieves this with relatively few steps (9.5), implying efficient gameplay. Test two had negative rewards, suggesting poor agent performance or potentially a punishment-based reward system. No average reward was recorded for tests two and three due to incomplete testing. Test four had the highest single reward (17.10) but a lower average, meaning it had high potential but low consistency.

Through this project, I gained a deep understanding of RL and CV, especially how they can be integrated to create adaptive systems. RL taught me how agents learn through trial and error, using feedback from rewards to refine their decisions over time. I saw how vital factors like epsilon decay, reward shaping, and episode structure impact learning outcomes.

Computer vision, on the other hand, played a vital role in enabling the agent to interpret and react to its environment through pixel data, using techniques like frame resizing, grayscale conversion, and object detection to simplify complex visual inputs. This project significantly enriched my overall knowledge of AI and ML not just in theory but in real-world applications. I learned how to train models, debug code under pressure, and interpret learning curves. Personally, I grew in my ability to troubleshoot creatively, stay patient through setbacks, and celebrate iterative progress—especially when the agent first succeeded in passing multiple pipes. Lastly, modifying the game environment—such as introducing moving obstacles, wind resistance, or adaptive difficulty—could create a richer and more challenging learning space for future experiments.

This project demonstrates the promising capabilities of reinforcement learning in mastering gameplay mechanics, as seen through the Flappy Bird AI agents. Future goals include expanding this research into more engaging, interactive environments, particularly in fashion and shopping, where reinforcement learning could be used to personalize style recommendations, or simulate creative decision-making. Beyond gaming, the same learning techniques can have practical applications in fields like robotics, where agents learn to make complex decisions in dynamic environments. Most excitingly, in the realm of property management and multifamily housing, these methods could be used to optimize leasing strategies, automate maintenance scheduling, or simulate tenant behavior for better service design—turning insights from simple games into smart, scalable business tools.

References

Feature Extraction. (n.d.). Retrieved from Domino AI: https://domino.ai/data-science-dictionary/feature-extraction

Murel Ph.D., J., & Kavlakoglu, E. (2024, March 25). *What is reinforcement learning?* Retrieved from IBM: https://www.ibm.com/think/topics/reinforcement-learning

Patel, M. (2023, October 23). *The Complete Guide to Preprocessing Techniques in Python*. Retrieved from Medium: https://medium.com/@maahip1304/the-complete-guide-to-image-preprocessing-techniques-in-python-dca30804550c

Reinforcement Learning. (2025, February 24). Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/what-is-reinforcement-learning/

Cassy Cormier

Learning Log: Train an AI Agent to Play Flappy Bird

15698 Computer Vision 1378

Professor Anna Devarakonda

April 20th, 2025

Learning Log: Flappy Bird AI Agent Project

- **Challenges Faced & Solutions Implemented**

1. Environment Setup Failures
    a. Early attempts to run the Flappy Bird environment in Google Colab often resulted in broken runtimes and execution errors.
    b. Solution: Restarting the Colab runtime session consistently helped resolve these errors. Eventually, setting up the environment locally with editable package installation improved reliability.
2. 2. Epsilon Decay & Exploration Strategy Flaws
    a. Challenge: In Test 1, the agent's epsilon value was stuck at 1.0, meaning it kept making random moves even after 500 episodes. No learning or rewards were recorded.
    b. Solution: The epsilon decay schedule was rechecked and implemented correctly, leading to a gradual shift from exploration to exploitation. By Test 4, epsilon had decayed to 0.1 and the agent showed improved behavior.
3. Sparse Reward Signal
    a. Challenge: With the default reward system, the agent mostly received -5.0 for dying and almost never survived long enough to get positive feedback.
    b. Solution: Introduced survival rewards (+0.1 per frame) to provide denser feedback and allow the agent to learn incrementally. This drastically improved learning stability.
4. Coding Bugs & Debugging Under Pressure
    a. Challenge: Frequent bugs in the step function and inconsistent environment behavior made debugging difficult, especially with time constraints.
    b. Solution: Leveraged ChatGPT and ClaudeAI for live debugging help, which saved time and supported deeper understanding of Python and ML code structures.
5. No Evidence of Learning
    a. Challenge: In early tests, the agent showed no meaningful improvement in performance.
    b. Solution: Introduced logging, performance tracking, and a reward plot function. Saved best-performing models automatically to monitor learning progress.

- **Key Learnings & Insights**

1. On Reinforcement Learning
    a. Gained firsthand experience with Deep Q-Learning (DQN), epsilon-greedy strategies, and how crucial reward design is for agent performance.
    b. Learned the impact of exploration-exploitation balance, reward sparsity, and target network updates in stabilizing training.
2. On Computer Vision

   a. Used image preprocessing, normalization, and grayscale conversion to reduce complexity and focus learning on relevant features like pipe edges and gaps.
   b. Applied feature extraction and object detection techniques to enhance the agent's decision-making capability.
3. Personal Growth
   a. Discovered patience and persistence are key in machine learning experiments. Debugging, especially when rewards plateau, taught resilience.
   b. Found joy in seeing meaningful results after solving a challenging problem—like when the agent first survived multiple pipes in Test 3.
4. Improvements to Consider
   a. Optimization Ideas
      i. Incorporate hyperparameter tuning to optimize learning rate, discount factor, and batch size.
      ii. Try Prioritized Experience Replay to speed up learning from high-value experiences.
   b. Technical Enhancements
      i. Explore Transfer Learning by fine-tuning pre-trained networks like MobileNetV2 for different environments.
      ii. Add multi-agent training to simulate cooperation or competition.
5. Game Environment Enhancements
   a. Introduce dynamic difficulty adjustment, like increasing pipe speed or spacing over time. Add more feedback types—like small positive rewards for flapping without dying—to reduce sparse rewards early on.

- Resources Used

1. ChatGPT (OpenAI): Assisted with Python debugging and reinforcement learning concept clarification.
2. ClaudeAI (Anthropic): Provided additional help with environment setup and code optimization.
3. MobileNetV2 (TensorFlow/Keras): Used as a pre-trained model for feature extraction in the DQN agent.
4. PyGame: Game engine for simulating the Flappy Bird environment.
5. OpenCV: Used for image preprocessing (grayscale conversion, normalization, feature extraction).
6. TensorFlow/Keras: Built and trained the neural network models.
7. JupyterLab & Google Colab / Local Setup: Used for coding, debugging, and model training.