Applying Reinforcement Learning To Infinite-Play And Random Environments: Exploring Optimal Performance In Flappy Bird Through Varying Q-Learning Algorithms

Aadarsh Jha

CS 5891: Reinforcement Learning

December 10th, 2021

ABSTRACT

A popular use case of Reinforcement Learning[1] (RL) is in reaching record-breaking scores in common game environments[2–4] through leveraging existing algorithms such as Q-Learning, Monte Carlo Tree Search, Deep Neural Networks, and other related RL algorithms. Herein, implementations of the Deep Q-Learning,[5] as well as the conventional Q-Learning,[6] algorithms are applied to the game of Flappy Bird (www.flappybird.io), which has the main objective of navigating through a set of green pipes via a bird. It is found that Deep Q-Learning, in its best trial, has an average score of 72.0, and a maximal score of 125. Vanilla Q-Learning, in its best trial, has an average score of 109.0, with a maximal score of 296. Both methods are strongly competitive against a manual-play, which has an average score ranging from 4.25 to 17.625.[7,8] Furthermore, increasing the value of ϵ provided competitive performance for Vanilla Q-Learning, yet strongly decreased the overall performance of Deep Q-Learning. Future directions of this work include increasing the amount of time in which all agents trained, as well as varying the network, hyperparamters, as well introducing more advanced concepts such as prioritized experience replay, Dueling DQN networks, etc. As mentioned in the conclusion, the computational complexity of these RL algorithms applied to Flappy Bird made it infeasible top apply as many experiments as preferred. This novelty of this work is that it serves as a foundational basis by which future research may understand the effect of popular reinforcement learning algorithms amongst infinite-play, random environments like that of Flappy Bird. The code for this project, as well as all trained models and results, is open to the public at: github.com/aadarshjha/flappy. High resolution versions of the figures can further be found at: github.com/aadarshjha/flappy/tree/main/figures.

1. MOTIVATION AND PROBLEM DESCRIPTION

1.1 Motivation

As aforementioned, RL algorithms have shown superior performance relative to a manual play in games, and other learning-based approaches. Herein, two such approaches, Q-Learning and Deep Q-Learning, are applied to the difficult environment of Flappy Bird. The problem area addressed in this exploratory analysis is the application of known, and standard, RL algorithms and applying them to known, but highly unexplored, complex environments (as stipulated in 1.1 in the project specifications). In particular, the Flappy Bird game is one in which the player controls a cartoon bird that must pass through a series of random and varied pipes. For each pipe that the user passes, they gain a point to their overall score. A gameplay of Flappy Bird may be found in the following

E-mail: aadarsh.jha@vanderbilt.edu

YouTube link: Flappy Bird Gameplay. Below, screenshots of the gameplay at different stages may be seen: 1) from the start, 2) in the middle of the game; and 3) at the termination of the game.

The difficulty of the game lies within the variation of the adjacent pipes with respect to the differential y-axis positioning of the gap within the two pipes. This is an interesting problem to work on due to two main factors: 1) the infinite-play nature of the game (non-terminating until the user either quits or loses), and 2) the random nature of the obstacles within the environment (in terms of the placement of the gap between pipes). Additionally, playing the game requires fine-grained control, and since there is no limit to score, increased stamina as the game progresses. A large state space is provided in the game due to its non-terminating nature, and the random placement of pipes through which the bird flies makes learning a static policy ineffective. Considering the various positions in which the gaps may be present on the screen on the y-axis, as well as the different velocities and positions that the bird may approach a pipe (e.g., the y-distance between the player and the lower pipe, the x-distance between the player and the pipes, velocity, etc.), it is no surprise that this is a very complex game. Therefore, an adaptive learning-based approach, such as Reinforcement Learning, would prove to be effective in such an environment. Overall, an exponential state, action space primarily makes this problem very difficult for a RL algorithm.

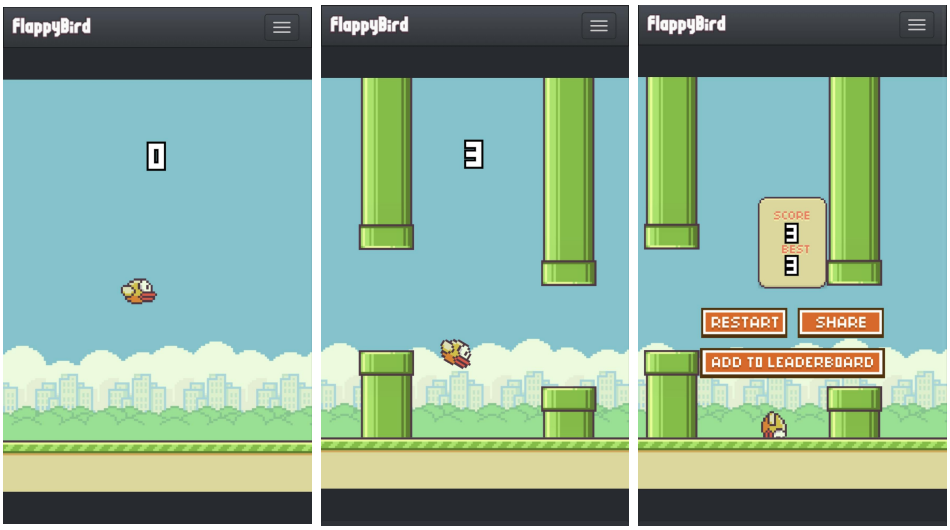

Figure 1. Three main states in Flappy Bird: Initialization state, Gameplay, and Termination.

1.2 Proposed Approach

In this paper, Q-Learning is primarily used as a test-bed upon which the effectiveness of RL techniques are analyzed for environments like Flappy Bird. The experimental design in this paper is to utilize an existing effective Deep Q-Learning and Q-Learning implementation, and adapt it to the Flappy Bird environment. More specifically, initial hyperparameters are adapted from previous implementations of Deep Q-Learning and Q-Learning algorithms available on-line (these sources are credited). Then, 5 seeds {0, 123, 457, 999, 1001} are used to evaluate Deep Q-Learning and Q-Learning based on the selected hyperparameters, for a set number of episodes. The reward, average score, and other salient factors are then compared so as to conclude and analyze the different advantages of each applied and studied framework. Finally, a follow up study of hyperparamter

optimization, focused on ϵ, is given. As mentioned, Q-Learning[1] and the subsequent Deep Q-Learning[5] algorithms are used within this work. Below, an enumeration describes the details and differences between the two:

1. **Q-Learning**: The Q-Learning[1] algorithm is considered to be an model-free reinforcement learning algorithm, which is off-policy and temporal difference based. Namely, a Q-Table caches the Q-Value associated with a (state, action) pair. Leveraging this table, the agent can interact with the enviornment and make updates to the (state, action) pairs in the Q-table. The agent will interact with the environment in two main ways: 1) randomly (exploratory), or 2) exploiting the Q-table to select a maximum value associated with a (state, action) pair. Exploitation and exploration is often controlled through an ϵ threshold hyperparameter. Updating the Q-table enables the Q-Learning algorithm to learn the optimal Q-Values.
2. **Deep Q-Learning**: The Deep Q-Learning algorithm can be seen as an extension to Q-Learning, with particular advantages when dealing with large state action spaces. Thus, a Neural Network is used to approximate the Q-value based on the state as the input (by generating all possible actions as the output, and taking the maximum — of course, depending on the value of ϵ). Past experiences are typically stored in a replay buffer. In such a sense, handling environments with a continuous state action space become more accessible.

1.3 Report Structure

The report is delineated into the following sections: 1) Background; 2) Approach; 3) Experimental Studies And Results; and 4) Discussions and Conclusions.

2. BACKGROUND

Herein, previous works to address the application of RL methods, as well as other learning-based methods, are discussed to further contextualize achieving optimal scores in Flappy Bird in an automated fashion. In particular, previous works have implemented[7,8] a Deep Q-Learning Network to achieve refined and competitive performance to human play. Fewer works explore the application of vanilla Q-Learning to Flappy Bird, but are also able to achieve competitive performance with a notably unique, and arguably simple, heuristic to Deep Q-Learning.[9] Other approaches such as Actor-Critic methods,[10] and Genetic Algorithms,[11] have also been applied as an initial study. The problem of solving Flappy Bird optimally has yet to be fully explored, relative to other games such as Go and Atari, and a comparative analysis of differential Q-Learning implementations is valuable to understand optimal approaches in automating such a game.

The implementation details utilized in this study primarily come from previous works that have attempted to apply RL to Flappy Bird. In particular, the Deep Q-Learning and Q-Learning implementations are largely inspired by the following GitHub repositories: DeepLearningFlappyBird as well as FlapPy-Bird-RL-Q-Learning-Bot, respectively. A technical overview of these implementations are summarized later on in this paper.

3. APPROACH

Within this section, a detailed look at the implementation details as well as experimentation are provided. In summary, hyperparameters are selected from previous works and a series of 10 trials (5 for each algorithm) of training runs are conducted so as to explore the effectiveness of each algorithm. Differential features of the game frame compose of the state, as well as actions guide the bird through the series of pipes. Both experiments are

run on a standard Google Colab environment, with Intel Xeon CPU 2.2 GHz, 13 GB RAM, 33 GB Disk Space, 12 GB NVIDIA Tesla K80 GPU, and CUDA 10.1. Additionally, both portions of the experimentation utilize the Flappy Bird clone github.com/sourabhv/FlapPyBird as the environment in which the agent learns, due to the creator of the game taking down, citing its addictive nature. The Deep Q-Learning algorithm is chosen as it has shown to achieve superior performance in games;[5] as a natural extension, the Q-Learning algorithm is also evaluated to see if a Deep Q-Learning really provides a conferred advantage given the state, action space of Flappy Bird.

3.1 Applying Deep Q-Learning Networks

Herein, a modified implementation of the aforementioned code base of DeepLearningFlappyBird is used. To begin, drawing from the existing work, the following hyperparameters are selected: Gamma = 0.9, Replay Memory = 50000. An adaptive epsilon-varying approach is implemented, wherein the initial epsilon value is set as 0.1, and the final value of epsilon is considered to be: 0.0001. The update rule of ϵ is defined as, where $E_i = 0.1$ and $E_f = 0.0001$:

$$E = E - \frac{(E_i - E_f)}{T}$$

The mini-batch size is set to be 32. In this implementation, a observation period is set as 10,000 in order to initialize the replay memory, whereas the exploratory/training experimentation is set to 1,000,000 (denoted as **T** in the above expression). In reality, all experiments are terminated at 1,300,000 frames due to the computational complexity of the problem, as discussed in the conclusion. An Adam optimizer is used with a learning rate of 1×10^{-6}.

This pipeline will convert the game-play frame into an image, which is in gray-scale (for computational effectiveness). The initial image, which is defined as 512x288, is also downsampled to a 28x28 image before being fed into the network. The network itself consists of the following components: 1) an 80x80x4 input layer; 2) a convolutional hidden layer with an 8x8x4x32 kernel and a stride of 4; 3) a 2x2 max pooling layer; 4) a convolutional layer with an 4x4x32x64 kernel at a stride of 2; 5) a 2x2 max pooling layer; 6) a convolutional layer with a 3x3x64x64 kernel at a stride of 1; 7) a 2x2 max pooling layer; 8) a hidden layer with 356 fully connected ReLU nodes; and 9) an output layer. In sum, there are three convolutional components of differential strides and dimensionality, max pooling layers, and a fully connected layer to produce a readout. The output of the network is a selection of actions that one can choose; of course, either a random action or a maximal action is taken dependent on the value of ϵ. The salient information of the game state is given by the state of the game (defined as an image of the current frame), the action, reward, next state (also defined as an image), and a boolean flag indicating if the episode has terminated (e.g., the user loses the game). Sampling the replay memory updates the gradients of the network, as in conventional Deep Q-Learning. The reward in this approach is defined in the following manner: 1) for a death, -1; 2) for simply continuing to exist (e.g, the user flaps), +0.1; and 3) for crossing a pipe, +1. After a standard baseline is set via 5 training sessions, further experimentation of hyperparameters is conducted and reported.

3.2 Applying Vanilla Q-Learning

Then, an implementation of a conventional (Vanilla) Q-learning algorithm is tested in a similar fashion as listed above: fives trials are run, and additional experimentation with hyperparameters are presented. Within this experimentation, the implementation used is inspired by that of FlapPy-Bird-RL-Q-Learning-Bot. In particular,

the learning rate is set at 0.7, the epsilon value is initially set to 0.001 and decayed with a factor of 0.8. The discount rate is set to a value of 0.9. Two unique implementation details of this particular approach is: 1) the use of reverse logic in the Q-table, as well as 2) penalizing upper-pipe crashes. To discuss (1), reversing the experience tuple allows for a faster propagation of Q-values in the table, since future data is updated first. Secondly, adding a penalty to upper-pipe crashes allows for the learnt policy to avoid the common case of the bird jumping too high. This implementation was initially proposed in the following repository: flappybird-qlearning-bot. A discretization of the state space into NxN grids is implemented in order to reduce the state space to reduce the runtime complexity; not all information of the game is needed and this allows for faster convergence.

The salient features of the state space are considered to be: 1) the y-distance between the player and the lower pipe; 2) the x-distance between the player and the pipes; and 3) the y-velocity of the agent. Reward is denoted in the following manner: 1) death is -10; 2) cross a series of pipes is +1; and 3) an upper-pipe crash is -15. Similar to Deep Q-Learning, the action space is either to jump or do nothing.

4. EXPERIMENTAL STUDIES AND RESULTS

Within this section, the experimental procedures to evaluate Deep Q-Learning as well as Q-Learning are presented. The dataset set and testbed environment used is aforementioned, and can be found in the following link: FlapPyBird. Code for this section may be found at: github.com/aadarshjha/flappy. High resolution figurers may be found at: github.com/aadarshjha/flappy/tree/main/figures.

4.1 Deep Q-Learning Experimentation

As shown in Figure 2 and Table 1 (on Pages 5 and 6), the results of the Deep Q-Learning experimentation are shown. To begin, all experimentation were run across 5 distinct seeds. It is very clear that the Deep Q-Learning is very consistent by way of its reward growth over time. In particular, all trials demonstrated a similar trend in average reward growth, and demonstrated a high standard deviation in reward near the termination of the training. Specifically, near 200,000 processed frames, there is a quasi-linear growth in both the average value and standard deviation of reward, wherein the agent learns at a steady rate. Overall, seed 0, 457, and 999 have the highest average reward, but also the highest standard deviation of reward.

The table demonstrates that the highest performing seed is 1001, with an average performance of 72.0, and a maximal performance of 125 over a testbed of 11 games. All trials fall under the score range of 27 to 72, with the minimum maximal score value being 61.

Table 1. **Deep Q-Learning**: The average, and maximal, performance of the algorithm across 11 trials of the game.

Seed Value	Average Performance	Maximal Performance
0	28.63	61
123	71.81	115
457	31.18	100
999	27.81	117
1001	72.0	125

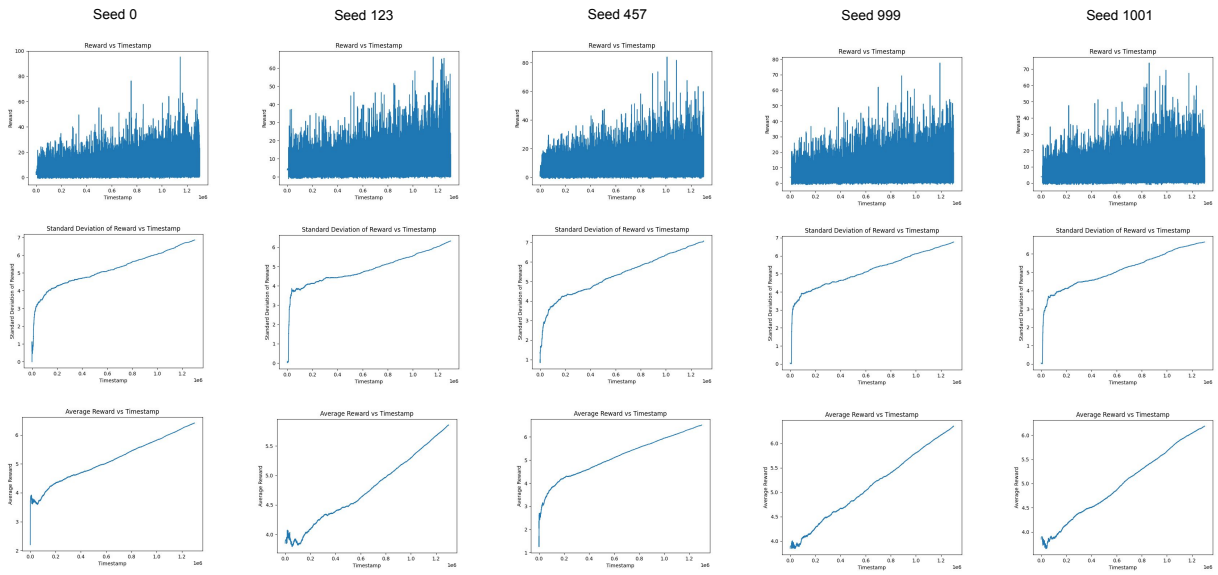

Figure 2. The results of executing **Deep Q-Learning** on Flappy Bird for 1,300,000 frames. The seed value is varied to ensure consistency. The first row is the raw reward value for each frame. The second row is the standard deviation of reward. The concluding row is the average value of the reward.

4.2 Q-Learning Experimentation

A similar procedure for training the Q-learning implementation is followed. In particular, the results of Figure 3 and Table 2 (on pages 6 and 7) demonstrate the performance of Q-Learning across 10,000 episodes. It is immediately clear that relative to the Deep Q-Learning approach, this approach is much more seed-dependent and has variable results across the board. For instance, seeds 0, 999, and 1001 demonstrate a high standard deviation, indicating a significant lack of consistency in performance. However, they also demonstrate the highest average reward near the termination of the training.

From the table, it is further clear that the seed value of 999 performed the best, with an average performance of 109.9, and a maximal performance of 296. All trials fall in the average range of 16.5 - 109.9. The minimum maximal performance is considered to be 32.

Table 2. **Q-Learning**: The average, and maximal, performance of the algorithm across 11 trials of the game.

Seed Value	Average Performance	Maximal Performance
0	16.5	32
123	33.6	146
457	21.2	39
999	109.9	296
1001	37.5	128

4.3 Hyperparameter Search And Additional Experimentation

Further experimentation leverage the existing implementations to conduct a small hyperparameter search. For the Deep Q-Learning experimentation, this study was expanded so as to further understand the optimal ϵ value.

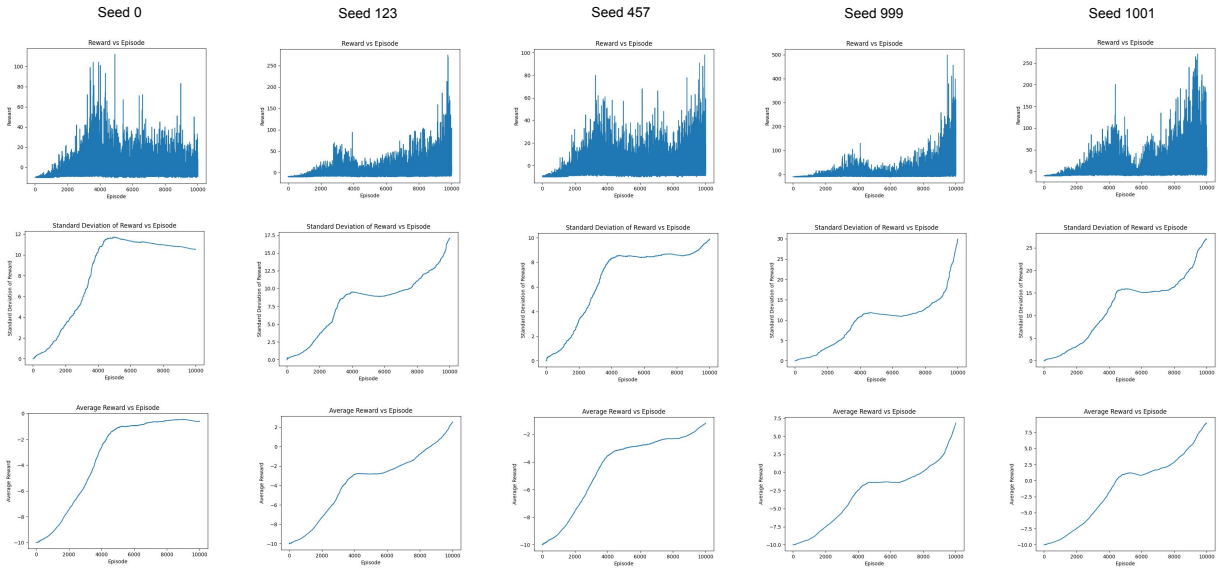

Figure 3. The results of executing **Q-Learning** on Flappy Bird for 10,000 games. The seed value is varied to ensure consistency. The first row is the raw reward value for each frame. The second row is the standard deviation of reward. The concluding row is the average value of the reward.

In particular, another trial wherein $\epsilon = 0.2$, and $\epsilon = 0.4$ are further explored, as shown in Figure 4. The central logic behind increasing the value of epsilon is to allow for more random actions to add to the memory of seen states, thereby increasing the probability of exercising more "deangerous", yet potentially strategic moves, such as jumping when already near a top pipe, or doing nothing when near a bottom pipe. However, this logic does not translate well when considering the training and testing scores associated with the aforementioned values of ϵ. When considering $\epsilon = 0.2$ and 0.4, it is apparent that there is far more variation and noise within the first 400,000 frames, whereas when $\epsilon = 0.1$ there was clear, albeit slow, growth. While average reward seems to recover and grow in $\epsilon = 0.2$, it seems to falter when $\epsilon = 0.4$. Due to the alteration of ϵ, it is suspected that more training frames were necessary to see proper convergence, as Table 3 demonstrates a great drop in performance, with average performance decreasing with an increase in ϵ. Additionally, increasing the replay memory size would also be beneficial in recalling such dangerous actions and penalizing the policy from taking it later on in the training cycle.

To continue the hyperparameter optimization in the Vanilla Q-Learning approach, a similar logic was applied — herein, the value of ϵ is set to 0.1, and 0.2. While Deep Q-Learning seems to suffer in performance, Q-Learning with a raised ϵ seems to beat seeds 0 and 457 in average performance. Additionally, the learning pattern with respect to the growth of reward, by way of average and standard deviation, is in-tune with the initial trials. It is also suspected that further increasing the training time would allow for competitive performance; but overall, increaesing the ϵ value seems promising. The results may be seen in Table 4 and Figure 5 on page 9.

Table 3. Performance metrics of 11 trials of the game via a **Deep Q-Learning** Strategy.

Epsilon Value	Average Performance	Maximal Performance
0.2	7.54	19
0.4	1.36	3

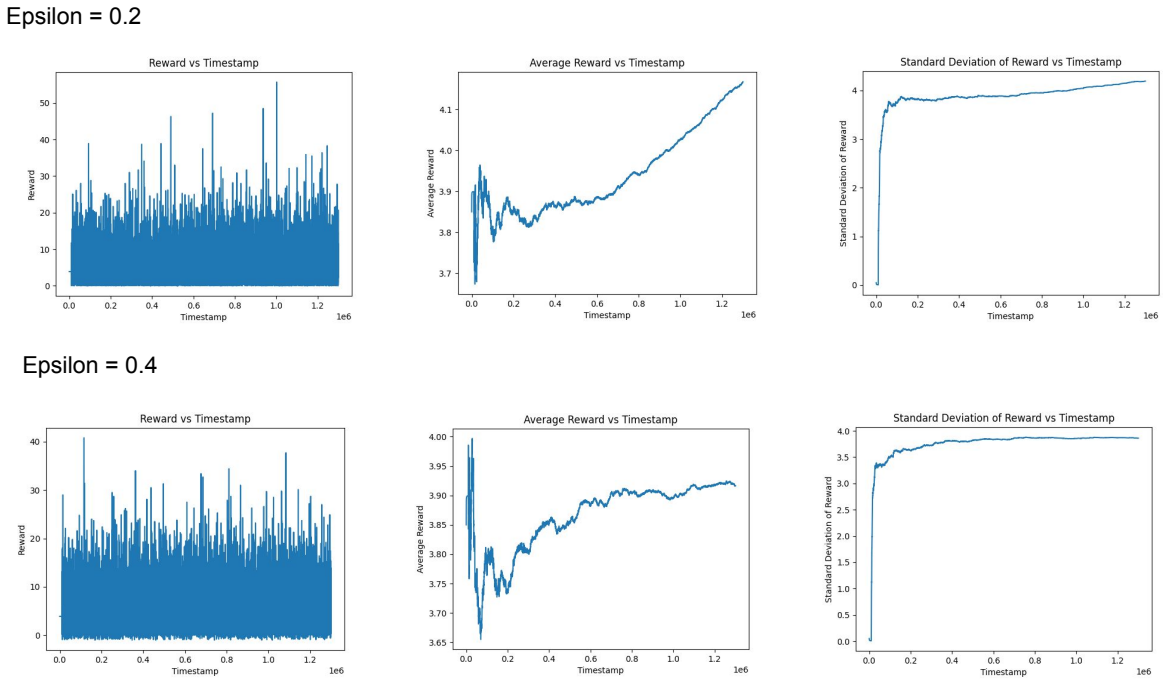

Figure 4. ϵ set to 0.2 and 0.4 for the **Deep Q-Learning** approach, trained for 1,300,000 frames.

Table 4. Performance metrics of 11 trials of the game via a Vanilla **Q-Learning** Strategy.

Epsilon Value	Average Performance	Maximal Performance
0.1	18.1	51
0.2	36.0	70

4.4 Comparative Analysis Of Applied Q-Learning Approaches

Both Q-Learning and Deep Q-Learning performed very well. Citing previous works, it can be seen that Deep Q-Learning has empirically[7,8] achieved a range of scores from 16.4 to 82.2. The results of this study are directly competitive with this range, with initial experimentation revealing an average Deep Q-Learning performance of 72.0. A singular previous work cites a high-score value of 169[9] for Vanilla Q-Learning, which is exceed in this paper with a value of 296.

With respect to this study, some major conclusions can be drawn: 1) Deep Q-Learning provides a more reproducible and consistently strong performance; 2) additional tuning to ϵ makes training on Deep Q-Networks unstable; and 3) the computational simplicity of Vanilla Q-Learning provides an overall competitive edge. To address 1), it can be seen that across all seed values, the convergence of average reward was highly standard, and no significant outliers were found in the testing phase of the Deep Q-Network (average values ranged from 27.81 to 72.0). On the other hand, an initial training of Q-Learning across 5 seeds demonstrated noisy performance and differential convergence, with average test values ranging from 16.5 to 109.9. To address 2), tuning ϵ to higher values so as to allow for more dangerous techniques to be explored only decreased the performance of the Deep Q-Learning approach, but provided promising and relatively consistent results for Q-Learning. The

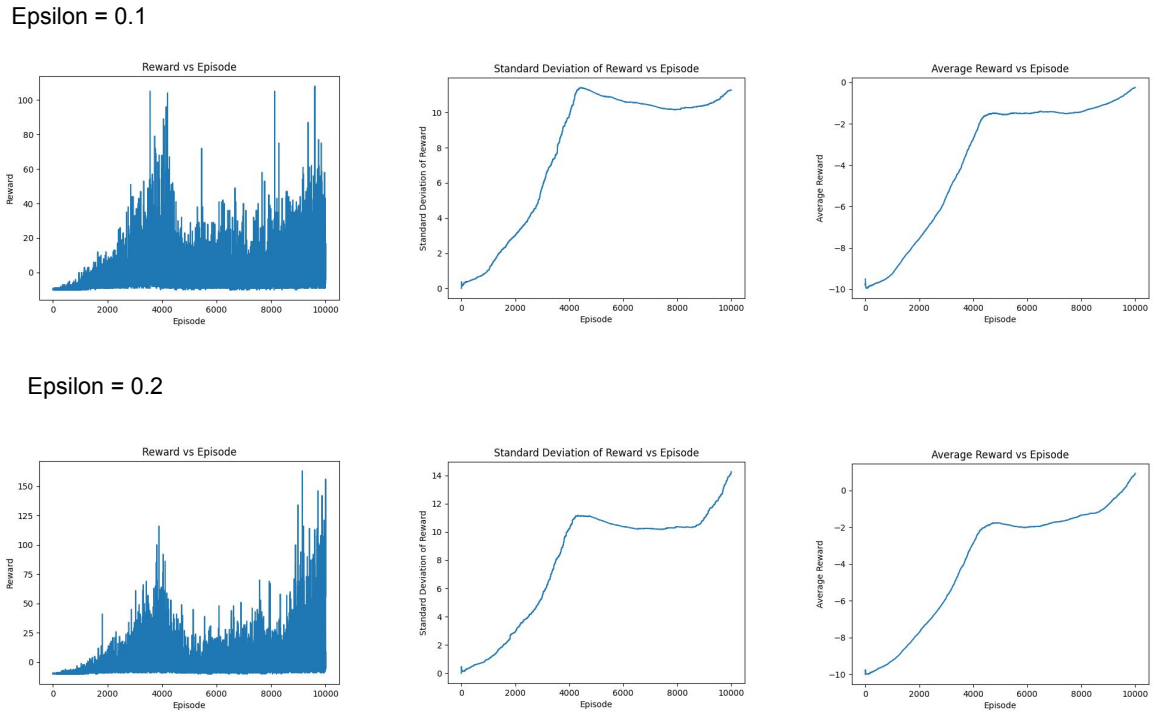

Figure 5. ϵ set to 0.1 and 0.2 for the **Q-Learning** approach, trained for 10,000 games.

addition of unexplored states for Q-Learning seems to be more beneficial in this case, though more episodes should be ran for a stronger conclusion, across multiple seeds. See the Discussion and Conclusion section as to why further exploration was infeasible due the computational complexity of the project. To address 3), it was found that the average compute time to run a singular training session for a Deep Q-Learning approach was roughly 22 hours, whereas running running the Vanilla Q-Learning would only take roughly 2 hours. The sheer difference in computational complexity between the two methods makes the Vanilla Q-Learning approach preferable due to its competitive performance, with such a short training time. The total time spent on this project, computationally, was 191 hours through Google Colab and personal CPU compute.

5. DISCUSSION AND CONCLUSIONS

To conclude, in this work, both Q-Learning and Deep Q-Learning methods are explored in order to understand effective Off-Policy, Temporal Difference learning approaches in reaching optimal performance in the Flappy Bird game. In particular, it was found that, on average, Deep Q-Learning provided consistently strong results, but came with a high training time and was more computationally complex. On the other hand, Q-Learning, while also competitive to Deep Q-Learning, suffered from high variance in the results, though provided much faster convergence. This study provides a further exploration of increasing the value of ϵ so as to include more random, and potentially strategic, (state, action) pairs. While showing competitive results for $\epsilon = \{0.1, 0.2\}$ for Vanilla Q-Learning, the values of $\epsilon = \{0.2, 0.4\}$ actually decrease the training performance for a Deep Q-Learning approach.

Many limitations, computationally, were present in this project that made it difficult to fully realize. When starting this project, it was revealed that Deep Q-Learning has an incredibly long training — with the implementation used, it would take 22 hours to train at least 1,300,000 frames. As such, many experiments presented in this study are terminated early and truly do not demonstrate the full performance of the particular network. The same is true for Q-Learning, which would take, on average, 2 hours to train. Though a best-effort for this project was made, and a total of 191 hours of personal cloud-based and personal compute was dedicated to this project. This significantly limited the ability to experiment with hyperparameters or tune the network architecture. Future directions of this work would include: 1) allowing for the network to train until maximal performance; 2) distributing the training across a large suite of RL approaches; 3) further exploring the effects of hyperparamters and network architectures on the result of optimal performance; and 4) reducing the differences between the implementation details of the Q-Learning and Deep Q-Learning approaches.

The significance of this studies lies in its critical analysis and comparison of two *de-facto* standard methods in RL. In particular, several seeds are provided as a testbed for performance to analyze how well RL can be genearalized to infinite-play (never-ending games), and random environments. Additionally, follow-up studies are performed to demonstrate the particular power of ϵ with respect to Q-Learning and Deep Q-Learning.

REFERENCES

[1] Sutton, R. S. and Barto, A. G., [*Reinforcement learning: An introduction*], MIT press (2018).

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., "Mastering the game of go with deep neural networks and tree search," *nature* **529**(7587), 484–489 (2016).

[3] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815* (2017).

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "Openai gym," *arXiv preprint arXiv:1606.01540* (2016).

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602* (2013).

[6] Jang, B., Kim, M., Harerimana, G., and Kim, J. W., "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access* **7**, 133653–133667 (2019).

[7] Chen, K., "Deep reinforcement learning for flappy bird," (2015).

[8] Pilcer, L.-S., Hoorelbeke, A., and Andigne, A., "Playing flappy bird with deep reinforcement learning [c]," *IEEE Transactions on Neural Networks* **16**(1), 285–286 (2015).

[9] Ebeling-Rump, M., Kao, M., and Hervieux-Moore, Z., "Applying q-learning to flappy bird," *Department Of Mathematics And Statistics, Queen's University* (2016).

[10] Alp, E. C. and Guzel, M. S., "Playing flappy bird via asynchronous advantage actor critic algorithm," *arXiv preprint arXiv:1907.03098* (2019).

[11] Xu, Q., "A multi-gene genetic algorithm for the flappy bird game based on neural network," in [*2021 International Conference on Neural Networks, Information and Communication Engineering*], **11933**, 7–14, SPIE (2021).