# Playing Modified Flappy Bird With Deep Reinforcement Learning

**Leo Li**[*], **Zhangnan Jiang**[*], **Zichen Yang**[*]

NYU Tandon School of Engineering

## Abstract

**GitHub Repository:**
**https://github.com/SeVEnMY/DeepLearningFinal**
In this work, we demonstrate the use of deep reinforcement learning (DRL) to play a modified version of the popular mobile game Flappy Bird, which included the addition of fireballs and the widening of the gap between the pipes. To tackle this task, we proposed the use of Dueling Deep Q Networks (DDQN), a variant of the Q-learning algorithm that separates the value function and the advantage function to improve learning efficiency. Our testing results showed that as the number of training iterations increased, the model's performance in terms of the maximum and average scores also improved, and using DDQN led to better performance compared to using standard Deep Q Networks (DQN). Additionally, using a pre-trained model as a starting point for training led to better performance compared to training from scratch. Our results demonstrate the potential of DRL for playing complex games and have implications for the broader field of DRL research, although there are still many challenges and limitations to using DRL for gaming tasks, including the sample efficiency of DRL algorithms.

## Introduction

Flappy Bird is a simple but challenging mobile game that has gained widespread popularity due to its addictive gameplay and easy-to-learn controls. In the original version of Flappy Bird, the player controls a bird that must navigate through a series of obstacles by tapping the screen to make the bird fly. The goal is to fly as far as possible without crashing into an obstacle or falling to the ground.

For the purposes of this study, we have modified the Flappy Bird game by adding fireballs that generate on a random y-coordinate and fly from the right to the left in a straight line (see figure 1). If the bird touches a fireball, it dies. We have also increased the gap size between the pipes to allow the AI to navigate the bird around the fireballs. These modifications are intended to make the game more difficult for artificial intelligence (AI) algorithms to play.

Playing Flappy Bird is a difficult task for AI algorithms due to the fast-paced nature of the game and the need for precise control. This has made it an attractive challenge for researchers in the field of deep reinforcement learning (DRL), which combines the power of deep learning with the principles of reinforcement learning to allow agents to learn complex tasks by interacting with their environment.

In this paper, we present a new approach for using DRL to play the modified Flappy Bird game. We propose the use of Dueling Deep Q Networks (DDQN), a variant of the popular Q-learning algorithm that uses a neural network to approximate the optimal action-value function. We demonstrate that DDQN can learn to play the modified Flappy Bird game at a high level, outperforming previous approaches on this task.

We will provide a detailed overview of the DDQN algorithm and our implementation in the following sections. We will also discuss the challenges and limitations of using DRL to play the modified Flappy Bird game, as well as the implications of our work for the broader field of DRL.

## Literature Survey

### Deep Reinforcement Learning Gaming

One of the earliest and most influential applications of DRL has been in the domain of computer games. Games provide a rich and challenging environment for DRL algorithms to learn and adapt, as they often require a combination of decision-making, problem-solving, and strategic planning skills.

One of the pioneering works in this area was the paper "Playing Atari with Deep Reinforcement Learning" (Mnih et al. 2013), which demonstrated the use of DRL to learn how to play a variety of Atari 2600 games using raw pixel inputs. The authors used a variant of the Q-learning algorithm called the Deep Q Network (DQN), which used a convolutional neural network (CNN) to approximate the optimal action-value function. The DQN was able to learn to play several Atari games at a level comparable to or exceeding human performance, setting a new benchmark for DRL in the gaming domain.

Since the publication of this paper, there have been numerous other works on using DRL to play games. These works have explored a wide range of techniques and algorithms, including policy gradient methods, actor-critic algorithms, and model-based approaches. Some notable examples include the use of DRL to play the games of Go (Sil-

[*]These authors contributed equally.

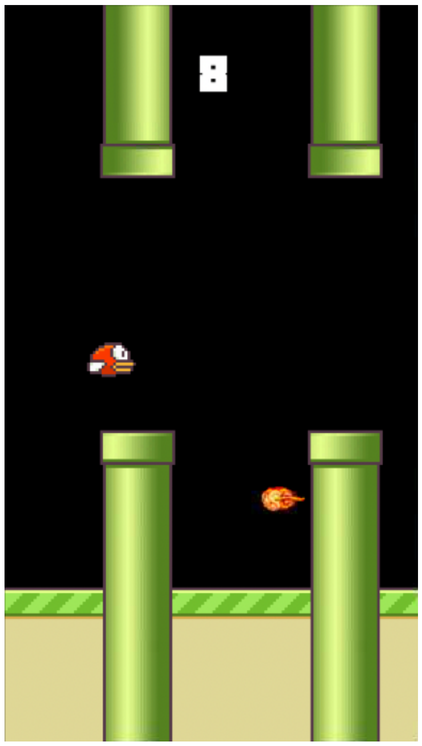


Figure 1: Modified Flappy Bird Game. There is a fireball near the bottom of the screen.

ver et al. 2016), Dota 2 (Berner et al. 2019), and complex MOBA games (Ye et al. 2019).

In addition to these approaches, there have also been a number of works on using DRL to play simpler, more arcade-style games such as Flappy Bird (Vu and Tran 2020), and Snake (Sebastianelli et al. 2021). These games offer a more focused and constrained learning environment but still require the agent to learn complex skills and strategies in order to succeed.

Overall, the use of DRL to play games has provided a rich and exciting research area that has helped to advance the state of the art in DRL and AI more broadly. It has also demonstrated the potential of DRL to solve complex and challenging tasks in a wide range of domains.

### Dueling Deep Q Networks

The Dueling Deep Q Network (DDQN) is a variant of the popular Q-learning algorithm that was introduced in the paper "Dueling Network Architectures for Deep Reinforcement Learning" (Wang, de Freitas, and Lanctot 2015). It is designed to improve the stability and efficiency of Q-learning in deep reinforcement learning (DRL) by decoupling the estimation of the value function and the advantage function.

In traditional Q-learning, the action-value function (Q-function) is used to estimate the expected future reward for each action in a given state. The Q-function is typically approximated using a neural network, which is trained to minimize the difference between the predicted Q-values and the true Q-values through the use of a loss function such as the mean squared error.

The DDQN algorithm introduces a novel architecture that separates the estimation of the value function (V) and the advantage function (A) into two separate streams. The V stream estimates the value of being in a given state, while the A stream estimates the relative advantage of each action in that state. The final Q-values are then computed as the sum of the V and A streams.

This architecture has been shown to improve the stability and convergence of the Q-learning algorithm, as well as reduce the sensitivity to the choice of the learning rate and the discount factor. It has also been demonstrated to outperform traditional Q-learning on a variety of tasks, including the Atari 2600 games and the game of Go.

Overall, the DDQN algorithm represents an important contribution to the field of DRL and has demonstrated its effectiveness in a wide range of tasks and environments. It has also inspired a number of follow-up works that have further explored and refined the use of dueling architectures in DRL.

## Methodology

### Pre-processing

Flappy Bird is an RGB game with multi-dimension game frames. To reduce the dimension, we perform the pre-processing procedure to reduce the number of dimensions. The original game frame input is $288 \times 512$ with 3 channels R, G and B. We simply convert the image input to grayscale and rescale it to $84 \times 84$. Above that, the input is then normalized to $[0, 1]$.

### Memory Buffer

When training DQN with continuous game frames, there are strong correlations between these consecutive inputs. In order to reduce the bond, a memory buffer is used. In each state memory, we simply store the quadruple $(s, a, r, s')$ into a buffer (implemented as a list). The size of the buffer is another hyperparameter to be selected. Most recent states will be included. The mini-batch used to update weights in DQN consists of sampled states from the buffer.

### Q-Learning

In reinforcement learning problems, we are aiming to make the best sequence of actions in order to maximize a given objective. In this Flappy Bird game, we need to figure out take the action "flap" or not use the learning module. For all-time game states, we have:

$$s_t = x_{t-n+1,a_{t-n+1},\ldots,x_{t-1},a_{t-1},x_t} \tag{1}$$

where $s_t$ is the game state at time $t$, $a_t$ is the action to take at time $t$, and $x_t$ is the input, in this question is a game frame at time $t$. $n$ is the length of the memory buffer, which is a hyperparameter to be set. The memory buffer will be introduced in the following section. Q-Learning could give an estimate of anticipated action at by time step, it is efficient in this control task. In Q-Learning, we have the Bellman equation:

$$Q^*(s_t, a_t) = r + \gamma \max_{a'} Q(s_{t+1}, a') \tag{2}$$

where $Q^*(s_t, a_t)$ is the $Q$-value at current iteration, $s_t$ and $s_{t+1}$ are the current and next state, $r$ is the reward value from the environment. Since there cannot be infinite rewards, we set a discount for future rewards, and $\gamma$ is the decay factor. It is aiming to update and converge to the optimal $Q$-value. In order to obtain the ability to generalization to the unseen data, we are trying to make use of a convolutional neural network in our approach, which is called a Deep Q Network(DQN). In each iteration, for the loss calculation we need to get a target $y_t$, which is called mini-batch in the training:

$$y_t = r + \gamma \max_{a'} Q(s, a'; w_{t-1}) \tag{3}$$

for a state $(s, a, r, s')$ defined before. Where $w_{t-1}$ are parameters of the DQN at iteration $t - 1$. a state point is a data point added to the memory buffer. We need to update the gradient of the loss function with respect to the weights using backpropagation.

## Loss function

In this problem, the Bellman error is used to calculate the loss to be backpropagated. The Bellman error is defined as the mean-squared error between the current estimated $Q$-value and the predicted mini-batch(target) $y_t$:

$$L = \frac{1}{2}(y_t - Q(s, a; w_t))^2 \tag{4}$$

where $y_t$ is the mini batch(target) at time $t$. Once we got the MSE loss, we can use Adam(Kingma and Ba 2014) optimizer to update the weights of the model. Then backpropagate updated weights to the model.

## Game Action Reward

In this modified flappy bird game, the score will be added by one of the birds passing either a pipe or a fireball successfully. But for the reward used by the Deep Q-Network, we need to reward every action the model takes. In the beginning, the reward is initialized as 0.1, after a sequence of actions, the bird successfully passed a pipe, and the reward would be added by 0.5. Same for the fireball, if the bird passed a fireball, the reward would be added by 0.5. Avoiding a fireball and passing a pipe are equally important. If the bird collided with a pipe or a fireball, the reward would be set to -1 to punish these actions.

## Method Pipeline

As illustrated in Figure2, at first we get game frames and pre-process them using the methods stated above. The memory buffer is initialized as an empty list. And we initialize the Deep Q-Network in Table 1 after taking the input of $84 \times 84 \times n$. In each iteration, we take game frames $x_t$ from the running game and update the state $s_t$ with $x_t$. The quadruple $(s, a, r, s')$ is added to the memory buffer. And we take the best action by:

$$a_t = argmin(Q(s_t, a)) \tag{5}$$

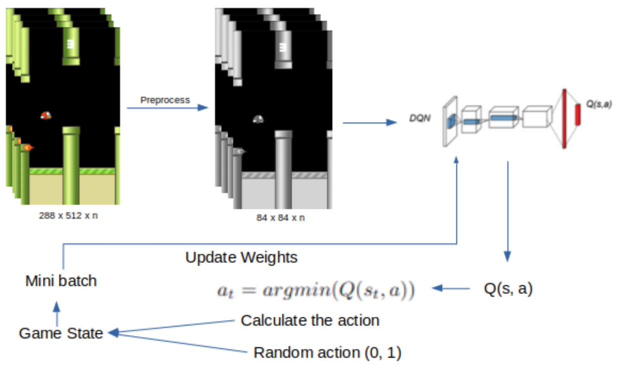


Figure 2: The pipeline of our method

In Q-Learning, there will be an exploration-exploitation problem, to avoid it, we add a random action with probability $\epsilon$. There is probability $\epsilon$ for the model to choose to flap the bird upwards. Otherwise, it will take the best-estimated action. The probability $\epsilon$ differs from 0.1 to 1 along with the training progress. Then the loss is calculated and backpropagated with the mini-batch. $s_t$ is updated with $a_t$ and the reward $r$ is updated. The game state will be updated at the end. Inside each iteration, the process is repeated until the bird collides with a pipe or a fireball.

| **Conv Name** | **Block type, Count** |
|---|---|
| conv1 | $[8 \times 8, 32]$ |
| ReLU | |
| conv2 | $[4 \times 4, 64]$ |
| ReLU | |
| conv3 | $[3 \times 3, 64]$ |
| ReLU | |
| Fully-connected | 512 outputs |
| Output(Fully-connected) | a single output for each action |

Table 1: Structure of the Deep Q-Network. There are 3 convolution layers and one nonlinear layer after each convolution. For the output, there is a fully-connected layer to compute one single output for each action

## Dueling Deep Q Network

The network structure of the Dueling Deep Q Network used here can be illustrated in Figure 2. The main idea behind the Dueling network structure in reinforcement learning is unnecessary for estimating the value of each action taken by the model. As in our situation of Flappy Bird, it is sometimes not quite important to know if a flap should be taken to avoid collision with pipes or fireballs, while in some states it is critical to take the right flap to keep the bird alive. This important insight is implemented by using the structure demonstrated in Figure 2. Instead of directly using a fully connected layer to transfer the convolution layers, two fully connected layers are used as the state value function and state-dependant action advantage function. These two functions would be combined to calculate a single output

Q function. For an agent behaving based on stochastic policy $\pi$, the values of the state-action pair(s, a) and the state s could be defined as the following As in (Wang et al., 2015):

$$Q^{\pi}(s,a) = E[R_t|s_t = s, a_t = a, \pi] \tag{6}$$

$$V^{\pi}(s) = E_{a\ \pi(s)}[Q^{\pi}(s',a')]|s,a,\pi] \tag{7}$$

Where s is the image from the game, and a is the action it takes. The advantage function as equation 7 relates the value and Q functions:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{8}$$

The value function V is used to measure how good the action is in a certain state, and the Q function is used to measure what would be the outcome when choosing a certain action in the state, while the advantage function A is used to get a relative measure of the importance of each action.
Therefore, using the definition of advantage, we could construct the aggregating module as:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{9}$$

In order to distinguish two actions obviously, in our implementation, the returned Q value would also abstract the mean of action score:

```
x=val+adv-adv.mean(1,keepdim=True)
```

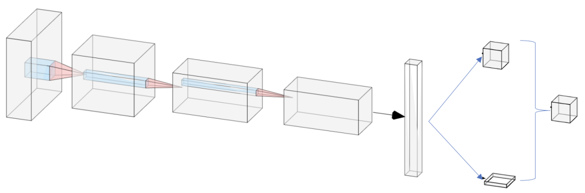


Figure 3: Dueling Deep Q Network Structure

## Results

### Training

The number of trainable parameters in DQN is 1,685,154. The Number of trainable parameters in Dueling DQN is 3,292,131. In our experiments, we tested different models (DQN and Dueling DQN). Various approaches of reward are also tested:
a. Reward +0.1 when the bird is initiated and alive, reward +1.0 when the center of the model passes the top center of one pipe, and reward -1.0 when the bird crashes with a pipe or a fireball (game over).
b. Reward +0.1 when the bird is initiated and alive, reward +0.5 when the center of the model passes the top center of one pipe, reward +0.5 when the center of the bird totally passes the x coordinate of one fireball, and reward just once for each fireball passed, reward -0.1 when the bird crashes with a pipe or a fireball (game over).
We also conducted experiments over training based on the pre-trained network. The pre-trained network is a Deep Q-network trained through playing the original Flappy Bird game (without fireballs), with the same hyperparameters set. The pre-trained model continues to be trained through playing the fireball Flappy Game. Intermediate models are also recorded when the number of iterations reaches 1,000,000. When the training is in progress, tensorboard is embedded to monitor the changes in loss, Q-value, and rewards.

### Hyperparameter selection

The Hyperparameters we used in all experiments are listed as follows:

- **Image size**: 84. The pre-processing we have done transfer the input image to 84*84 size.
- **Learning Rate**: 1e-6
- **Number of Iterations**: 2,000,000. Here the number of iterations also means the number of game frames that feeds into the model.
- **Memory buffer**: 48,000. We tuned different memory buffer sizes to avoid crashing due to running out of memory.
- **Gamma**: 0.99. Here gamma is used to model that future rewards are worth less than immediate rewards.
- **Optimizer**: Adam.

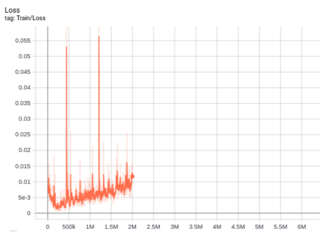
In this part, we basically keep these hyper-parameters as the ones we are based on, since the purpose of this project is to compare if a dueling deep Q network would work better than the original deep Q network if the pre-trained model would work better, and the difference between two reward methods.

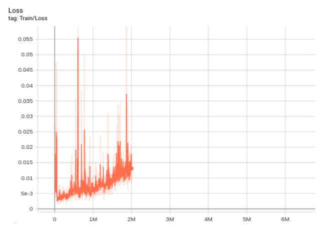
### Testing results

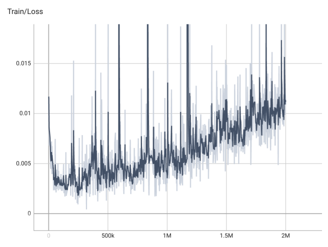
For testing, we run each trained model 100 times, record all the scores the certain model gets from 100-time games, and compare the highest score and the average score in the 100-time games.

| Model Name | Max Score | Avg Score |
|---|---|---|
| Pre-trained DQN, reward b. fireball, 1,000,000 iters | 32 | 11.42 |
| Pre-trained DQN, reward b. fireball, 2,000,000 iters | 97 | 21.22 |
| DQN, reward b., 1,000,000 iters | 12 | 3.71 |
| DQN, reward b. 2,000,000 iters | 47 | 8.28 |
| DDQN, reward a. fireball, 1,000,000 iters | 17 | 5.24 |
| DDQN, reward a. fireball, 2,000,000 iters | 58 | 14.59 |
| DDQN, reward b. fireball, 1,000,000 iters | 8 | 3.19 |
| DDQN, reward b. fireball, 2,000,000 iters | 25 | 7.18 |

Based on the testing results, we could see that when the number of iterations goes up, the maximum score and the average score would be better. Compared with DQN trained from scratch, DQN trained based on a pre-trained no-fireball-reward model would have better performance. In


(a) DQN, reward method b.


(b) Pre-trained DQN, reward method b.


(c) DDQN, reward method a.

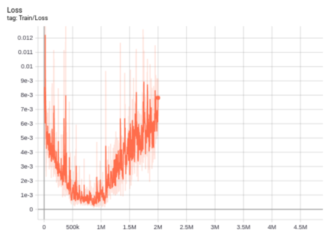
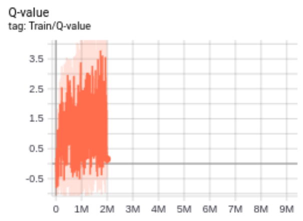
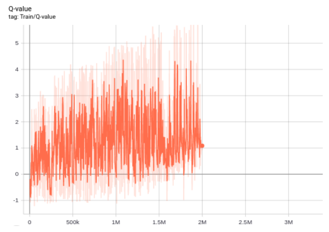
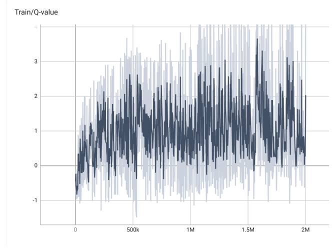

(d) DDQN, reward method b.

Figure 4: plots of training losses


(a) DQN, reward method b.


(b) Pre-trained DQN, reward method b.


(c) DDQN, reward method a.

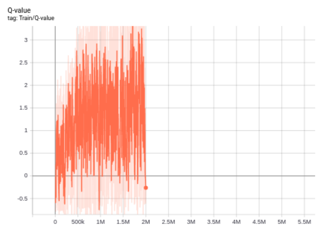

(d) DDQN, reward method b.

Figure 5: plots of Q-values

our opinion, this is because DQN requires a great number of iterations to improve, if we could train our scratch DQN with more game frames, the performance would be better. In general, we could see that as the number of iterations goes up, the maximum score and the average score would increase. Compared with DQN, Dueling DQN works better under the same condition. Both the maximum score and average score are higher than DQN when the iteration number is 1,000,000 and 2,000,000. However, compared with the DDQN without reward on avoiding fireball, the result is not as good as the one without reward. In our understanding, as we look at the training loss for the DDQN with reward, the training loss decrease at first, but it goes up after a certain iteration, the model is not converging, which may cause the decrease in performance.

## Conclusion

In this work, we aimed to demonstrate the use of deep reinforcement learning (DRL) to play a modified version of the popular mobile game Flappy Bird. The modifications to the game included the addition of fireballs and the widening of the gap between the pipes, which made the game more challenging for artificial intelligence algorithms to play.

To tackle this task, we proposed the use of Dueling Deep Q Networks (DDQN), a variant of the Q-learning algorithm that separates the value function and the advantage function to improve learning efficiency. We implemented DDQN using the PyTorch library and trained the model using a combination of exploration and exploitation to maximize the reward.

Our testing results showed that as the number of training iterations increased, the model's performance in terms of the maximum and average scores also improved. We also found that using DDQN led to better performance compared to using standard Deep Q Networks (DQN). Additionally, using a pre-trained model as a starting point for training led to better performance compared to training from scratch.

Overall, these results suggest that both the number of training iterations and the choice of algorithm can have a significant impact on the model's performance in playing the modified Flappy Bird game. Further research may be needed to identify optimal hyperparameter settings and training strategies for improving the model's performance.

## References

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *CoRR*, abs/1912.06680.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Sebastianelli, A.; Tipaldi, M.; Ullo, S.; and Glielmo, L. 2021. A Deep Q-Learning based approach applied to the Snake game.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Vu, T.; and Tran, L. 2020. FlapAI Bird: Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques. *CoRR*, abs/2003.09579.

Wang, Z.; de Freitas, N.; and Lanctot, M. 2015. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR*, abs/1511.06581.

Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; Chen, Q.; Yin, Y.; Zhang, H.; Shi, T.; Wang, L.; Fu, Q.; Yang, W.; and Huang, L. 2019. Mastering Complex Control in MOBA Games with Deep Reinforcement Learning. *CoRR*, abs/1912.09729.