# Teaching different Reinforcement Learning agents to play a game

Mite Mazgaliev

October 2022

**Abstract**

**Reinforcement learning is an area of Machine learning that is concerned with how intelligent agents take actions in a certain environment. There are a handful of techniques in Reinforcement learning but, in our case we will be using RL to teach multiple agents to play a simple game and compare their results.**

## 1 Introduction

Reinforcement learning is a powerful approach for training intelligent agents to perform a wide range of tasks. One important application of reinforcement learning is in the realm of games, where it can be used to train agents to play complex and challenging games at a high level. In this paper, we present a reinforcement learning-based approach to training an agent to play a specific game(**ChopperScape**). Our approach leverages the power of Q-learning, Deep Q-learning and modern computational resources to learn a policy that allows the agent to make intelligent decisions and achieve a high level of performance in the game.Through a combination of theory and experimental results, we will demonstrate the effectiveness of our approach and its potential to be applied to a wide range of games.

## 2 Related Work

**Q-learning** is a model-free reinforcement learning algorithm ( **it does not use the transition probability distribution**), it is a trial and error algorithm which learns by actually going through the game, and experiencing the environment and how it behaves. Q-learning computes a Q-table in which is kept for each state which actions how much Quality value it has or the expected reward from a action in a given state(**hence the Q-value name**).

**Deep learning** is a ML technique that teaches computers to do what comes naturally to humans, learning from examples, it allows computational models

that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction using deep neural networks.[3][2]

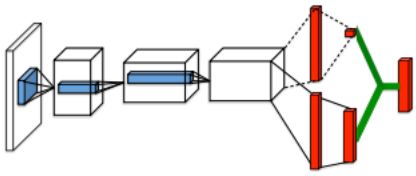
The agent starts by going through the game and for each action it takes it updates the q-table on how the action has affected him or the me (**Gets rewarded for the action or gets punished**), and given an infinite amount of exploration time and some randomness in its actions it will find the optimal policy for any environment.[6]

One of the difficulties of this task is that there are indefinite amount of states since the canvas of the game is random, and random enemies can attack our agent at random times as well as random rewards can appear on the screen at random times.

**DQN (Deep q-networks) is a combination of Q-learning and deep neural networks, instead of the brain of the agent being Q-Tables like in regular Q-learning the algorithm uses neural networks [1].**

**DoubleDQN, it is shown that the popular Deep Q-Learning algorithm is known to overestimate action values under certain conditions, more specifically it is shown that it happens quite frequently in the Atari 2600 games. In Double Q-learning two value functions are learned by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights. For each update, one set of weights is used to determine the greedy policy and the other to determine its value.** [7]

**DuelingQN, the DuelingQNetworks have a bit of a different architecture where it has two streams, one to seperate estimate state-value and the advantages for each action and the green output module combnies them. Both of the networks output Q-values for each action.** [8]



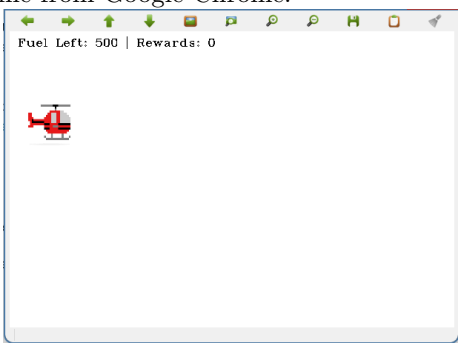
Trying to find solutions from previous work done to this exact game on this field was a dead end, so I looked into a similar game which is Dino Game and came by this article [5] which had a implementation of DQNAgent and DDQNAgent to play the game using Convolutional layers.

Similarly I looked at Atari games agents, which have a similar structure to the game I am trying to explore.

# 3 Description

## 3.1 Game Description

The game is called ChopperScape which is a game that is similar to the Dino Game from Google Chrome.




As we can see from the starting screen of the game we have a Helicopter and his total fuel, each step the Helicopter makes he loses fuel points and gains one reward point. The game contains a fuel entity as well as a bird entity they both add more complexity to the game.

### 3.1.1 State

The state of the game is a **Canvas** where all of the elements of the game are drawn, it is a **(400 x 600 x 3)** image . Each element (Chopper, Fuel or Bird) have **(x, y)** coordinate positions and are drawn around them.

### 3.1.2 Game mechanics

The Helicopter always starts at the top left side of the screen with 500 Fuel and no enemies on the board for him to face. Once the game starts for each step the agent makes he gains a reward 1 point and loses fuel. The Fuel element and the Bird element come into play after the game starts, there is a small chance that from the right side of the screen at a random coordinate a **bird** starts approaching towards our chopper **(The number of birds approaching may vary)**. The same mechanics work for the **fuel** element as well except that the fuel spawns from the top of the screen and is falling down towards the end of the screen.

The Helicopter has five actions, (**Up, down, left, right or do nothing**) to avoid the incoming enemy birds and to catch the falling fuels to manage to maximize his survival time or reward as much as he can (When he picks up fuel he gains extra points and when he gets hits by a bird he loses points).

## 3.2 Approach Description

### 3.2.1 The model

As first given that taking the full on state from a **(400, 600, 3 )** image will be quite hard we will do some preprocessing and reduce the dimensionality of the image. We will re-scale all the elements to be adapted to a smaller state, and the state to be a **(84, 84)** and gray scale so that it takes way less time to train the agent on the state.

For training our agents we will use a similar model to the one mentioned in [4] where our input will be a **(84, 84, 3)** where we will be giving **3 frames** of the gray-scaled game state on each input.

**The first model** we will first use will consist of 2 convolutional layers with **16 (8x8)** filters with stride 4 and the second one with **32 (4x4)** filters with stride 2 with relu activations. Then the output from the last convolutional layer is flattened with a Flatten layer and is sent to the first Dense layer with relu activation and 256 units and the second one will have 128 units, and the output layer is a fully connected layer with a single output for each action.

### 3.2.2 Rewards and parameters

So in order to achieve the best possible agent we will be trying different approaches to reward the agent, or maybe take away a few key elements of the game just so that he can learn one aspect of the game rather than all at once.

**The first approach** I took was giving the agent a -1 when he hits a bird which would result in the end of the game, so that he would avoid them and a small punishment of 0.001 per movement so that he doesn't stay in place all the time and do nothing as well as a positive reward +1 for each fuel that he catches.

**The second approach** I took was giving the agent a bigger punishment when he dies so that he more quickly learns that birds should be avoided and kept the same punishment for the step and same reward for catching a fuel.

**The third approach** I took was giving the agent a bigger reward on catching fuels while keeping the punishments the same as the first approach.

**The fourth approach** I took was give the agent a positive reward on movement, so that he would be encouraged to actually stay alive for longer, and I kept the rewards for dying and catching fuel.

**The fifth approach** Is a bit different where I will actually remove the birds from the game, just so that the agent will be encouraged to learn to catch fuels and achieve higher step counts, but in order to teach it to catch the fuels we will reduce its fuel by half and make maximum steps to be 1500, so that the agent will eventually learn to achieve these 1500 steps by collecting fuel.

# 4 Results

## 4.1 First approach

For the first model we trained a DQN, DDQN and a DuelingDQN, and recieved the following results over **10000** training episodes.

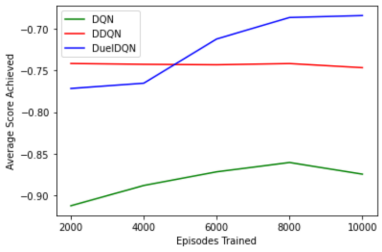



Figure 1: Comparison of mean scores achieved in first approach

As we can see from the figure above the DQN gets an average score that is close to **-1**, which is the penalty of the game for dying concluding that most of the time and even until the very end the agent just keeps on dying without doing much of anything **(which would mean that he would need much more episodes to learn the game )**. On the other hand the DoubleDQN and DuelingDQN have a better performance which would mean that they eventually started catching more fuel which rewards them **+1** for each one they catch. And we can see that in this case the DuelingDQN actually managed to make much greater progress that the other two.

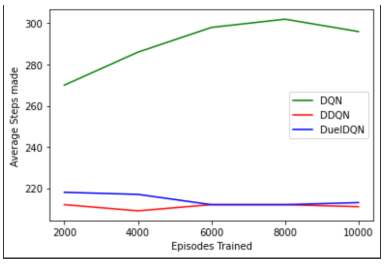



Figure 2: Comparison of mean steps taken in first approach

But, the interesting thing we can see from Figure 2 is that the DQN on average lasted around **100 steps** more than the other two which would imply that either its better at avoiding the birds but doesn't bother to pick up fuels, which in a sense is a better since the agent lasts longer in the environment which is the final goal, but it achieves a lower score by not picking up any fuels.

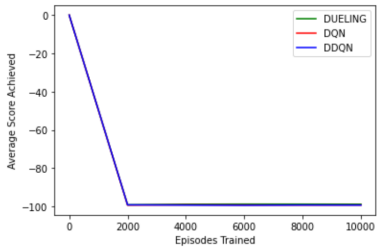



Figure 3: Average score achieved on second approach

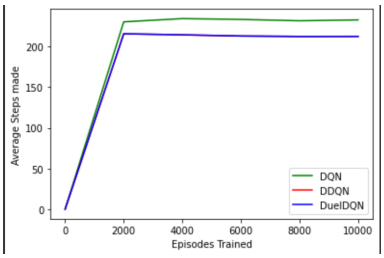



Figure 4: Average steps taken in second approach

## 4.2 Second approach

As we can see from figures **3** and **4** on the second approach where the punishments on loss is increased the agent the score is really close to the absolute minimum **-100** because the agent doesn't seem to find a way to survive for longer. The overall steps are less in the DDQN and Dueling DQN but the regular DQN has a decrease in the amount of steps it manages to take per round.

## 4.3 Third approach

From the figures of the third approach we can see clearly that the agents have a increase in their score as the episodes go on and they experience the environment more. As we can see the DDQN score is a steady increase and not as chaotic as the DQN and the DDQN, this means that the DQN and DuelingDQN got better states and managed to explore the environments in a much better way. Out of the 3 algorithms in this case the DQN seems to have made the best improvement over time as it had started at a way lower score achievemnts tan the DuelingDQN but managed to almost even out the achievements by the end of the episodes.

As for the steps we can see that the Dueling and the DDQN actually hold up a bit over the DQN in durability.

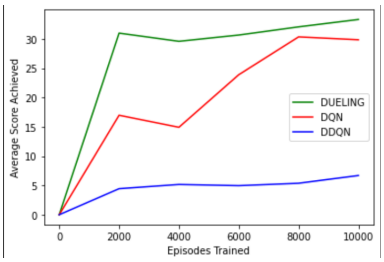



Figure 5: Average score achieved on third approach

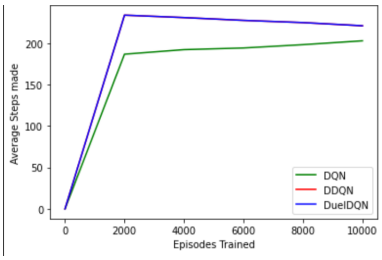



Figure 6: Average steps taken in third approach

## 4.4 Fourth approach

Nothing much to add on the fourth approach as the results are almost identical to the second approach, not much improvement in any shape of form.

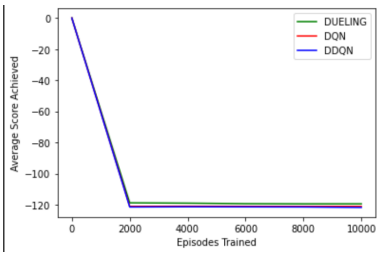



Figure 7: Average score achieved on fourth approach

## 4.5 Fifth approach

The fifth approach being a bit different from the others we would expect a difference in the results. As previously mentioned the fifth approach agents will start with a max fuel of 150 and try to achieve as much as they can with that.

From the figures we can see that the DuelingDQN has the overall highest score which means that he would catch a fuel and manage to stay alive for a bit

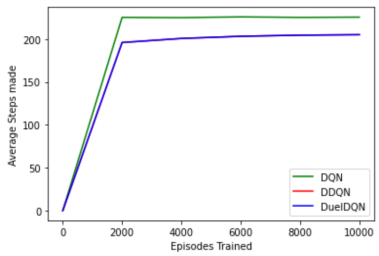



Figure 8: Average steps taken in fourth approach

longer than the others far more often.

The DQN has similar performances to the DuelingDQN but with less frequent action to catch the fuels.

And the DDQN had the least amount of fuels caught overall.

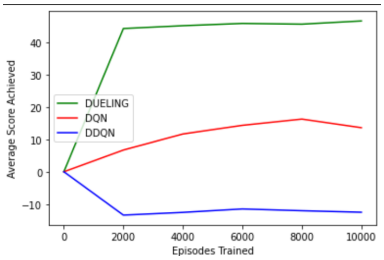



Figure 9: Average score achieved on fifth approach

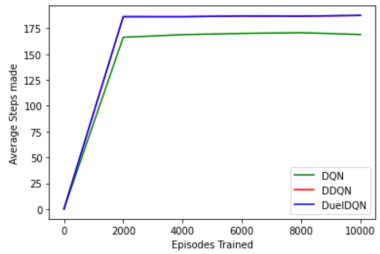



Figure 10: Average steps taken in fifth approach

# 5 Conclusion

Out of all the approaches the most promising results were from the **first** and **third** approach, that is because as the training episodes went on, the epsilon for random actions went down and their average scores kept rising at a steady

pace with the episodes played, giving that they were improving at a greater rate than the other approaches where almost always the models hit a plateau and don't make any forward progress. For further progress first it should be looked into training the agent on more episodes. For better ways to reward/punish the agent on its' actions.

# References


[1] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] István Szita. Reinforcement learning in games. In *Reinforcement learning*, pages 539–577. Springer, 2012.

[7] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[8] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.