INM707 – Deep Reinforcement Learning
Student ID: 150010219
Name: Seyoan Santhagunam

Basic

<u>What is deep reinforcement learning</u>

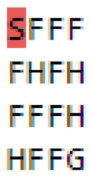
A branch of machine learning called deep reinforcement learning which combines deep learning and reinforcement learning (RL). In RL, the problem of a computational agent learning to make judgments through trial and error is taken into account [1]. As deep learning is incorporated into the solution, agents can make judgments based on input from unstructured data without having to manually engineer the state space.

The aim of reinforcement learning is to identify the best possible actions to maximise the rewards for the agent. The agent understands the environment by the actions it takes.

1) <u>Define an environment and the problem to be solved</u>

The environment is that area in which the agent is able to move around for given states and actions and for each state a corresponding reward is provided depending on the outcome of the state.

Environment: Using gym, the frozen lake environment is a (4x4) grid, which is made up of 4 different outcomes: Start(S), Frozen (F), Hole(H) and Goal(G). The aim for the agent is to travel around the grid to reach the goal. On the route the agent will face holes where if the agent lands on a hole the episode restarts, and the reward value is 0. This process is a loop where the agent carries on learning from each mistake until it reaches the goal. Here is a visual representation of the grid.



```
SFFF
FHFH
FFFH
HFFG
```

Figure 1 – The environment of Frozen Lake

Problem: The agent will be allowed to move in 4 different directions: Left, Right, Up and Down. Q-learning will be applied to this model. During learning, there can be random actions every few episodes however we need to try and reduce the probability of random actions as with deep learning we want the optimal outcome.

From the grid, there are 16 different possible positions that the agent could be at. At each state, the agent will be given 4 different directions to move in. This gives the agent a possibility of 64 (16 x 4) different direction changes in total. These movements will be our weights which will be updated according to the moves taken.

2) <u>Define a state transition function and the reward function</u>

<u>State transition function</u>

The probability of an agent changing states in Reinforcement Learning is most generally defined by the state transition function. The new state is updated depending on the actions taken by the agent. This creates the state transition function which can be shown as $s_{t+1} = \delta(s_t, a_t)$, where a_t represents the action taken (the direction of movement) and s_t represents the current state the agent is on [2].

Reward Function

The Reward Function is an incentive system that instructs the agent through rewards and punishments what is right and wrong. The agent wants to maximise overall rewards [3]. In order to maximise the total rewards, we occasionally need to forego the immediate benefits. In our environment, the reward is the agent reaching the end, 'G' where the reward is 1. However, there are no negative rewards involved as the squares labelled 'F' have a reward of 0.

3) Set up the Q-learning parameters (gamma, alpha) and policy

(This involved Python code)

Q learning parameters

- α (alpha) is the learning rate, which is a value between 0 and 1, which tells how much we should be changing the original Q value. If the value of alpha is equal to 0, this means the Q value does not change and if alpha is equal to 1, the Q value changes very quickly. In the Q learning algorithm, we are applying, we use the value of alpha to be 0.5. In principle, we already know that this will happen very quickly. This would make the reward and the maximum value will increase the value in the next state.
- γ (gamma) is the discount factor, another value which is between 0 and 1, which tells us how much weight the agent has on future rewards. If the value of gamma is equal to 0, the agent only focuses on the immediate rewards rather than the future rewards. If the value of gamma is equal to 1, the agent focuses more on future rewards instead of just the immediate rewards. We will be using the gamma value of 0.9

In the case of the Frozen Lake, we use a high discount factor as there is only 1 reward which is the end goal. We will be running this for 1000 episodes.

Epsilon Greedy Policy

Exploration vs Exploitation

The epsilon greedy policy takes an action using the greedy policy where the probability is 1-ε and the random action has a probability of ε.

An agent can profit in the long run by expanding its current understanding of each action through exploration. An agent can make better decisions in the future by increasing the estimated action-values' accuracy.

By taking use of the agent's present action-value estimates, exploitation, on the other hand, selects the action that will yield the greatest reward. However, being overly greedy when it comes to action-value estimates may prevent one from reaping the greatest benefits and result in less than ideal behaviour. [4]

An agent's assessments of action-values improve as it explores. And if it exploits, it may receive a greater prize. However, it cannot choose to perform both at once, a situation known as the exploration-exploitation dilemma.

4) Run the Q-learning algorithm and represent its performance

In our Q-learning algorithm, we set our hyperparameters to be: alpha = 0.5, gamma = 0.9 and epsiodes = 1000. We create a Q-table which begins off as a (4x4) matrix where all the values are zeros and those values get updated from the Bellman equation, provided below

$$\boxed{\text{New } Q(s,a)} = \boxed{Q(s,a)} + \boxed{\alpha}\,\boxed{[R(s,a)} + \boxed{\gamma}\,\boxed{\max Q'(s',a')} - \boxed{Q(s,a)}]$$

🟥	New Q Value for that state and the action
⬛	Learning Rate
🟫	Reward for taking that action at that state
🟪	Current Q Values
🟩	Maximum expected future reward given the new state (s') and all possible actions at that new state.
🟧	Discount Rate

Figure 2: Bellman Equation [5]

In our Q learning algorithm, we have set direction values such as [0 = LEFT, 1 = DOWN, 2 = RIGHT, 3 = UP]. This helps in providing out optimal sequence for the against to reach the end goal.

After seeing how our Q-table has been updated, we train the agent and see how successful the agent is within the 1000 episodes.

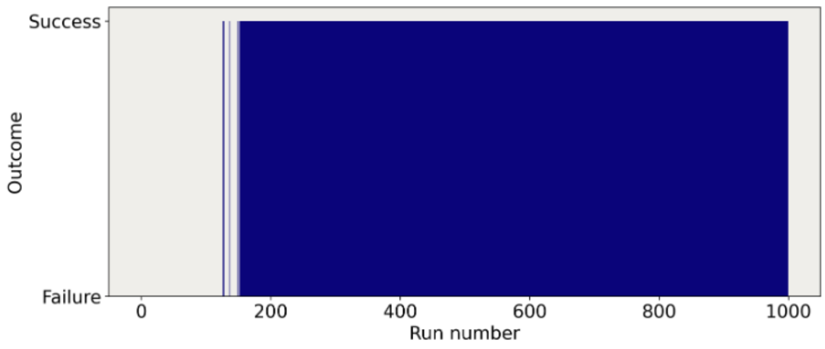


Figure 3: Q-Learning with alpha and gamma

The agent is now trained, where each blue bar indicates a win. We can see around the 120th episode is where the agent has successfully won for the first time. After a few episodes of the win, the agent understands how to win the game and it constantly wins every game afterwards.

We now check this with 100 episodes. As we see the training is complete, we have no need to update our Q-table any further. To measure the agents performance, we calculate the percentage of success.

We can identify that we have trained the agent and we have a 100% success rate.

We can then visualise this and the output provided is [2, 2, 1, 1, 1, 2], which means [RIGHT, RIGHT, DOWN, DOWN, DOWN, RIGHT].

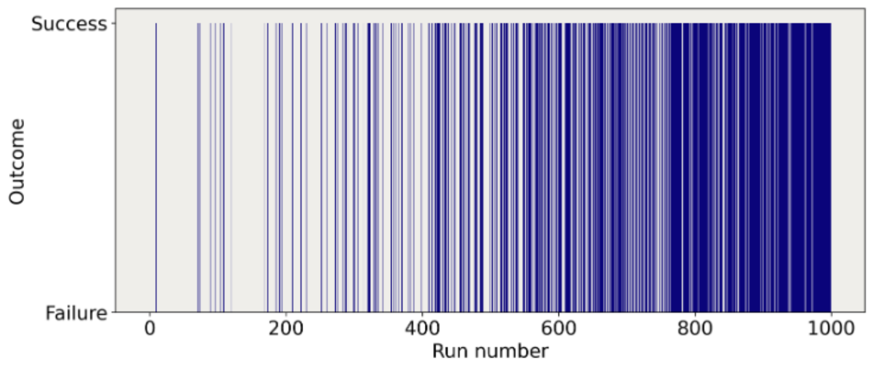


Figure 4: Q-learning with alpha, gamma, and epsilon

We then run the Q learning algorithm with the epsilon greedy policy. When this happens, we can see that more values of the Q table are updated rather than just a few. The agent begins learning the optimal route much quicker and experiments the environment much more. The success rate is not affected by this.

5) Repeat the experiment with different parameter values, and policies

(This section is Python code)

6) Analyse the results quantitatively and qualitatively

The first parameter we will be changing is alpha and maintaining both gamma and epsilon. We will looking into 2 different values of alpha: 0.8 and 1.0.

When alpha is 0.8,

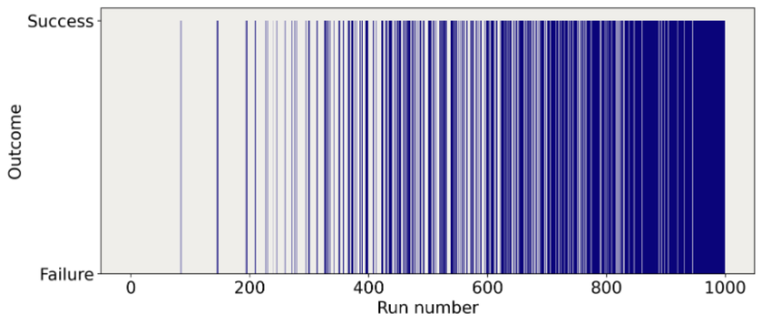


Figure 5: Q-learning when alpha = 0.8

When alpha is 1,

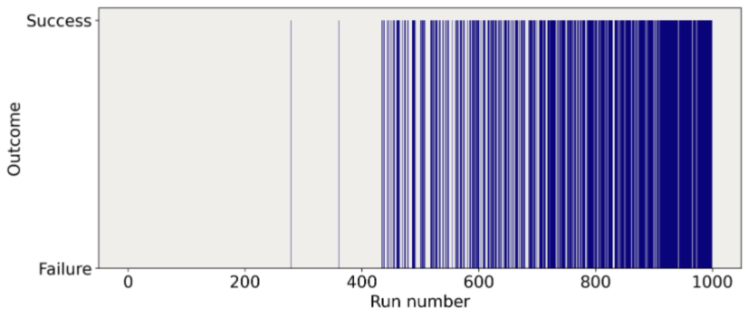


Figure 6: Q-learning when alpha = 1

From the diagrams above we can see that when the learning rate is increased, it takes the agent much more episodes to successful reach the goal. The success rate for both trials is 100%.

The second parameter we will be changing is gamma and maintaining both alpha and epsilon. We will looking into 2 different values of gamma: 0.6 and 0.1.

When gamma is 0.6,

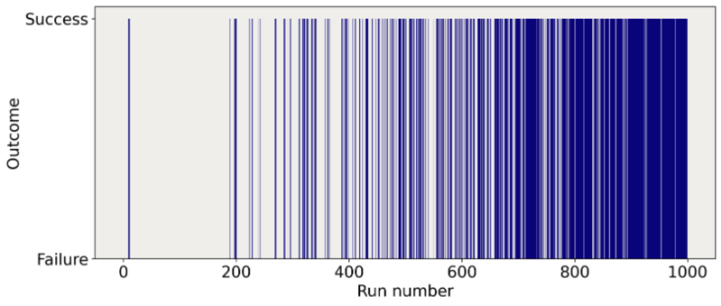


Figure 7: Q-learning when gamma = 0.6

When gamma is 0.1,

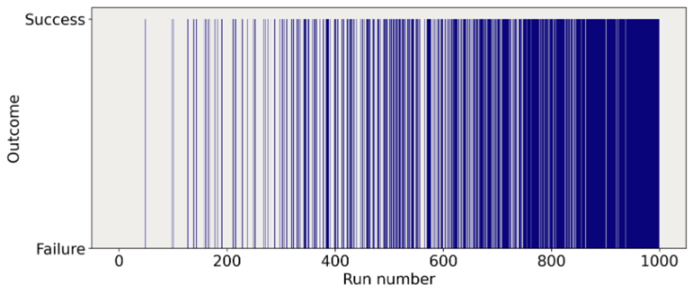


Figure 8: Q-learning when gamma = 0.6

From the diagrams we can see that when the discount factor is decreased, there are episodes of success.

The final parameter we will be changing is epsilon and maintaining both alpha and gamma. We will looking into 2 different values of gamma: 0.8, 0.7 and 0.4

When epsilon is 0.8,

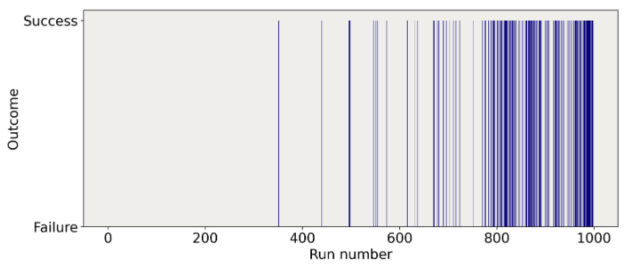


Figure 9: Q-learning when epsilon = 0.8

When epsilon is 0.7,

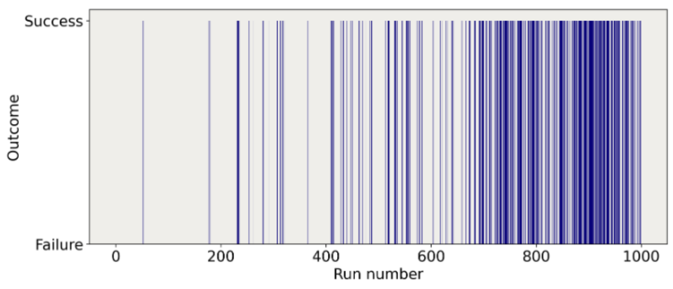


Figure 10: Q-learning when epsilon = 0.8

When epsilon is 0.4,

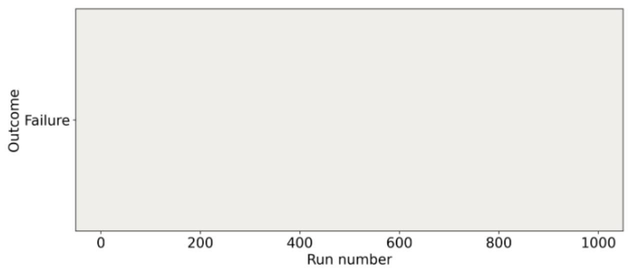


Figure 11: Q-learning when epsilon = 0.8

Here we can see that changing the epsilon value has a huge effect. The lower the epsilon value, the lower the rate of success. When epsilon is 0.8, the success rate is 77%. At 0.7 the success rate is 62% and when epsilon is 0.4, there is actually no success, so the agent fails to reach the end goal.

Advanced

7) <u>Implement DQN with two improvements</u>

What is Deep Q-Network (DQN)?

DQN is a model-free RL algorithm that makes use of current deep learning technology. DQN techniques use a deep neural network or convolutional neural network to estimate the Q value function and Q-learning to determine the optimum course of action to take in the current situation.

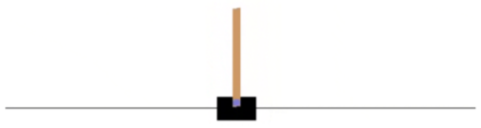


Figure 12: Cartpole v0 environment

We will be using the Cartpole-v0 environment for this DQN task. In the Cartpole environment the agent is given 2 directional movements, either left or right. The aim of the Cartpole, is very straight forward, balance the pole on the cart without making it fall off. There are 4 states that we have: position, velocity, angle of the pole and the velocity of the base of the pole. A reward of +1 is provided to the agent for every incremental time step.

In this we will apply the DQN to Cartpole-v0. As well as applying DQN with will be applying 2 improvements in the form of DDQN and Dueling. We use DDQN and Dueling as these seem to be the most popular choices of improvements for DQN's.

<u>DDQN</u>

A Double Deep Q-Network, Double DQN, uses Double Q-learning to break down the maximum operation in the target into action selection and action evaluation, which reduces overestimation. We assess the greedy strategy using the online network, but we quantify its value using the target network. DQN's are known for overestimating rewards, and we will apply DDQN to prevent his from happening

<u>Dueling</u>

For Dueling DQN, the algorithm separates the Q values into value functions and advantage functions. The value function provides the reward from states and the advantage functions provides us with the improved efficiency of an action.

8) <u>Analysis quantitatively and qualitatively</u>

DDQN

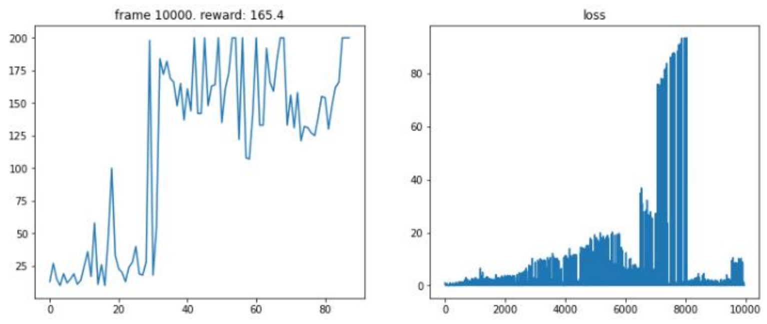


Figure 13: The reward and loss obtained after 10,000 frames for DDQN

In figure 13, we can see that the DDQN algorithm provides us with a reward of 165.4. We have also included its loss over the time steps beside it.

Dueling

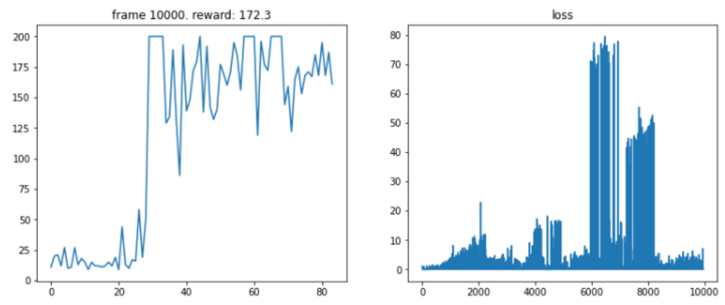


Figure 14: The reward and loss obtained after 10,000 frames for Dueling

In figure 13, we can see that the DDQN algorithm provides us with a reward of 172.3 We have also included its loss over the time steps beside it.

Looking at the 2 algorithms, Dueling has performed better than the DDQN.

9) <u>Apply the RL algorithm from rllib to an Atari Learning Environment</u>
(This is section is Python code)

Atari is a producer of interactive entertainment that oversees a portfolio of intellectual property that includes more than 200 games and franchises.

From our Atari environment, we will implementing the game SpaceInvaders-v4.

We will using an reinforcement algorithm from RLlib to an Atari Learning environment

What is RLlib?

Built on Ray, RLlib is the open source, industry-recognized Python framework for reinforcement learning. It comprises more than 25 of the newest algorithms, all of which are constructed to run at scale and in multi-agent mode and are designed for quick iteration and a fast road to production. [6]

10) <u>Analyse the results quantitively and qualitatively</u>

When we create the tensorboard, we are able to plot many graphs but we have the graph of episode reward mean against the steps. From which we are able to see that by the end of 20,000 steps, including both dueling and double q, as well as a high learning rate of 0.1 returns the highest reward. The lowest reward returned is by the variable which does not include double q, so we can see that double q plays a big part in the return of the reward.

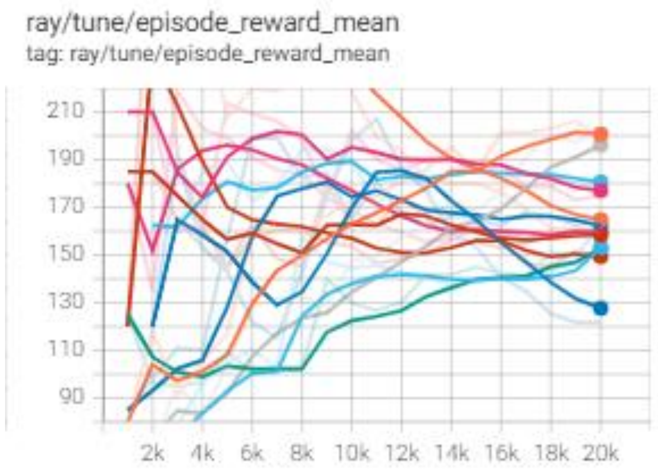
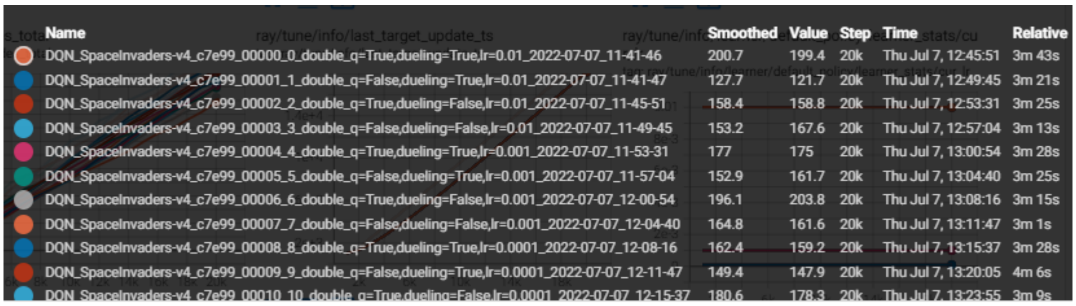


Figure 15: episode reward mean for Atari



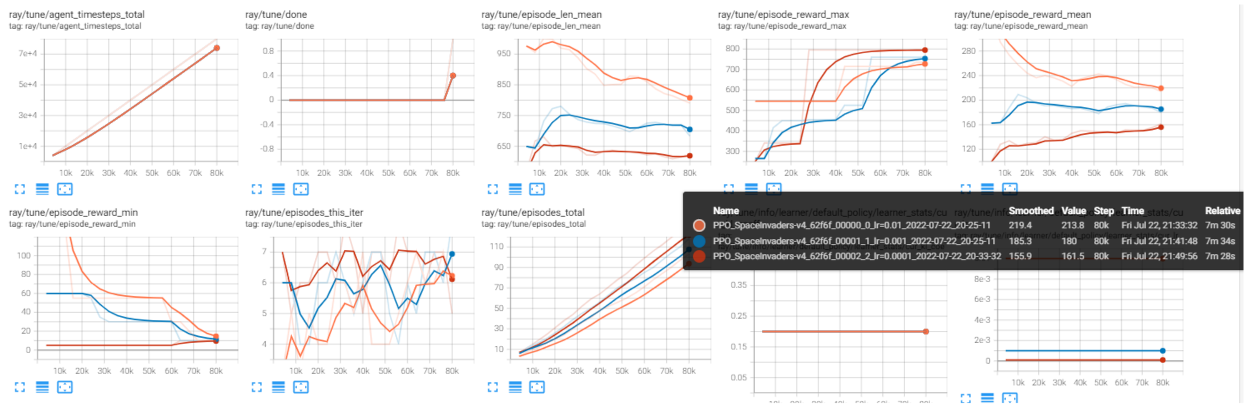
	Name	Smoothed	Value	Step	Time	Relative
●	DQN_SpaceInvaders-v4_c7e99_00000_0_double_q=True,dueling=True,lr=0.01_2022-07-07_11-41-46	200.7	199.4	20k	Thu Jul 7, 12:45:51	3m 43s
●	DQN_SpaceInvaders-v4_c7e99_00001_1_double_q=False,dueling=True,lr=0.01_2022-07-07_11-41-47	127.7	121.7	20k	Thu Jul 7, 12:49:45	3m 21s
●	DQN_SpaceInvaders-v4_c7e99_00002_2_double_q=True,dueling=False,lr=0.01_2022-07-07_11-45-51	158.4	158.8	20k	Thu Jul 7, 12:53:31	3m 25s
●	DQN_SpaceInvaders-v4_c7e99_00003_3_double_q=False,dueling=False,lr=0.01_2022-07-07_11-49-45	153.2	167.6	20k	Thu Jul 7, 12:57:04	3m 13s
●	DQN_SpaceInvaders-v4_c7e99_00004_4_double_q=True,dueling=True,lr=0.001_2022-07-07_11-53-31	177	175	20k	Thu Jul 7, 13:00:54	3m 28s
●	DQN_SpaceInvaders-v4_c7e99_00005_5_double_q=False,dueling=True,lr=0.001_2022-07-07_11-57-04	152.9	161.7	20k	Thu Jul 7, 13:04:40	3m 25s
●	DQN_SpaceInvaders-v4_c7e99_00006_6_double_q=True,dueling=False,lr=0.001_2022-07-07_12-00-54	196.1	203.8	20k	Thu Jul 7, 13:08:16	3m 15s
●	DQN_SpaceInvaders-v4_c7e99_00007_7_double_q=False,dueling=False,lr=0.001_2022-07-07_12-04-40	164.8	161.6	20k	Thu Jul 7, 13:11:47	3m 1s
●	DQN_SpaceInvaders-v4_c7e99_00008_8_double_q=True,dueling=True,lr=0.0001_2022-07-07_12-08-16	162.4	159.2	20k	Thu Jul 7, 13:15:37	3m 28s
●	DQN_SpaceInvaders-v4_c7e99_00009_9_double_q=False,dueling=True,lr=0.0001_2022-07-07_12-11-47	149.4	147.9	20k	Thu Jul 7, 13:20:05	4m 6s
●	DQN_SpaceInvaders-v4_c7e99_00010_10_double_q=True,dueling=False,lr=0.0001_2022-07-07_12-15-37	180.6	178.3	20k	Thu Jul 7, 13:23:55	3m 9s

Figure 16: Performance with other variables

Extra

11) Implementation PPO

The Proximal Policy Optimisation (PPO) algorithm is a policy gradient method used for reinforcement learning. PPO collects small batches of experiences that get involved with the environment and the decision-making policy is updated depending on the batches [7]. From running the Space Invaders environment, we are able to see the plot of Mean Reward against number of Episodes with 3 different learning rates of 0.1, 0.01 and 0.001. The figure below shows our results:



	Name	Smoothed	Value	Step	Time	Relative
●	PPO_SpaceInvaders-v4_62f6f_00000_0_lr=0.01_2022-07-22_20-25-11	219.4	213.8	80k	Fri Jul 22, 21:33:32	7m 30s
●	PPO_SpaceInvaders-v4_62f6f_00001_1_lr=0.001_2022-07-22_20-25-11	185.3	180	80k	Fri Jul 22, 21:41:48	7m 34s
●	PPO_SpaceInvaders-v4_62f6f_00002_2_lr=0.0001_2022-07-22_20-33-32	155.9	161.5	80k	Fri Jul 22, 21:49:56	7m 28s

Figure 17: performance with PPO

From looking at our results we can see that the agent was not able to outperform the DQN algorithm which we looked at earlier due to having less iterations to train this PPO. If provided with more iterations and episodes PPO would be able to understand and learn the game better due to a better relationship with the environment. This will allow the agent for the best way to win the game as we can see from the figure that each curve increases its mean reward when the number of iterations increase.

References

[1] - En.wikipedia.org. 2022. *Deep reinforcement learning - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Deep_reinforcement_learning>.

[2] - Myers, A., 2006. *State Machines*. [online] Cs.cornell.edu. Available at: <https://www.cs.cornell.edu/courses/cs211/2006sp/Lectures/L26-MoreGraphs/state_mach.html#:~:text=The%20state%20transition%20function%20says,%CE%B4(s%2Ce) .

[3] - Anon, 2022, Reinforcement learning *Wikipedia* Available at https://en.wikipedia.org/wiki/Reinforcement_learning

[4] - Shawl, S., 2020. *Epsilon-Greedy Algorithm in Reinforcement Learning - GeeksforGeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/> [Accessed 12 August 2022].

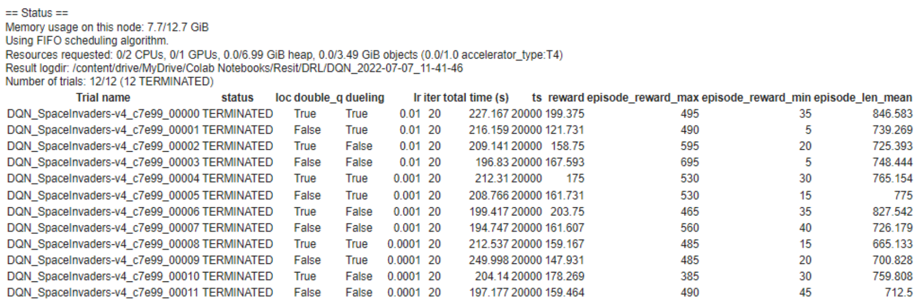
[5] - freeCodeCamp.org. 2018. *An introduction to Q-Learning: reinforcement learning*. [online] Available at: <https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/#:~:text=Q-Learning%20is%20a%20value-based%20reinforcement%20learning%20algorithm%20which,by%20selecting%20the%20best%20of%20all%20possible%20actions.>.

[6] - Ray. 2022. *RLlib - Scalable, state of the art reinforcement learning in Python*. [online] Available at: <https://www.ray.io/rllib>.

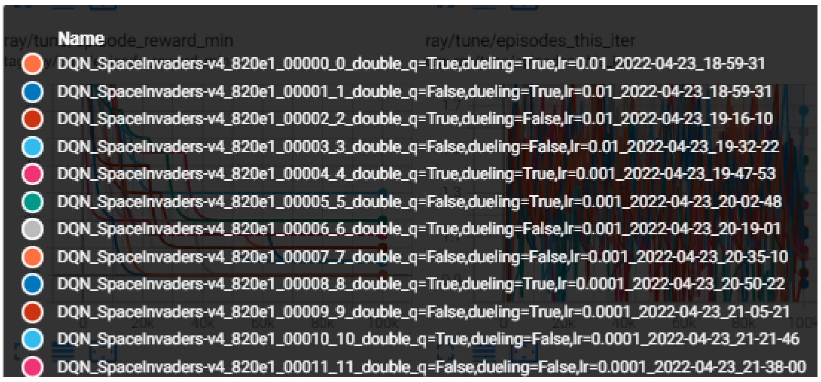
[7] - Paperswithcode.com. 2022. *Papers with Code - PPO Explained*. [online] Available at: <https://paperswithcode.com/method/ppo#:~:text=Proximal%20Policy%20Optimization%2C%20or%20PPO,using%20only%20first%2Dorder%20optimization.>.

Appendix

Space Invader results from rllib



```
== Status ==
Memory usage on this node: 7.7/12.7 GiB
Using FIFO scheduling algorithm.
Resources requested: 0/2 CPUs, 0/1 GPUs, 0.0/6.99 GiB heap, 0.0/3.49 GiB objects (0.0/1.0 accelerator_type:T4)
Result logdir: /content/drive/MyDrive/Colab Notebooks/Resit/DRL/DQN_2022-07-07_11-41-46
Number of trials: 12/12 (12 TERMINATED)
           Trial name                   status    loc double_q dueling      lr iter total time (s)     ts  reward episode_reward_max episode_reward_min episode_len_mean
DQN_SpaceInvaders-v4_c7e99_00000 TERMINATED     True     True      0.01  20     227.167 20000 199.375                495                 35          846.583
DQN_SpaceInvaders-v4_c7e99_00001 TERMINATED     False    True      0.01  20     216.159 20000 121.731                490                  5          739.269
DQN_SpaceInvaders-v4_c7e99_00002 TERMINATED     True     False     0.01  20     209.141 20000  158.75                595                 20          725.393
DQN_SpaceInvaders-v4_c7e99_00003 TERMINATED     False    False     0.01  20      196.83 20000 167.593                695                  5          748.444
DQN_SpaceInvaders-v4_c7e99_00004 TERMINATED     True     True     0.001  20      212.31 20000     175                530                 30          765.154
DQN_SpaceInvaders-v4_c7e99_00005 TERMINATED     False    True     0.001  20     208.766 20000 161.731                530                 15              775
DQN_SpaceInvaders-v4_c7e99_00006 TERMINATED     True     False    0.001  20     199.417 20000  203.75                465                 35          827.542
DQN_SpaceInvaders-v4_c7e99_00007 TERMINATED     False    False    0.001  20     194.747 20000 161.607                560                 40          726.179
DQN_SpaceInvaders-v4_c7e99_00008 TERMINATED     True     True    0.0001  20     212.537 20000 159.167                485                 15          665.133
DQN_SpaceInvaders-v4_c7e99_00009 TERMINATED     False    True    0.0001  20     249.998 20000 147.931                485                 20          700.828
DQN_SpaceInvaders-v4_c7e99_00010 TERMINATED     True     False   0.0001  20      204.14 20000 178.269                385                 30          759.808
DQN_SpaceInvaders-v4_c7e99_00011 TERMINATED     False    False   0.0001  20     197.177 20000 159.464                490                 45            712.5
```



Name
● DQN_SpaceInvaders-v4_820e1_00000_0_double_q=True,dueling=True,lr=0.01_2022-04-23_18-59-31
● DQN_SpaceInvaders-v4_820e1_00001_1_double_q=False,dueling=True,lr=0.01_2022-04-23_18-59-31
● DQN_SpaceInvaders-v4_820e1_00002_2_double_q=True,dueling=False,lr=0.01_2022-04-23_19-16-10
● DQN_SpaceInvaders-v4_820e1_00003_3_double_q=False,dueling=False,lr=0.01_2022-04-23_19-32-22
● DQN_SpaceInvaders-v4_820e1_00004_4_double_q=True,dueling=True,lr=0.001_2022-04-23_19-47-53
● DQN_SpaceInvaders-v4_820e1_00005_5_double_q=False,dueling=True,lr=0.001_2022-04-23_20-02-48
● DQN_SpaceInvaders-v4_820e1_00006_6_double_q=True,dueling=False,lr=0.001_2022-04-23_20-19-01
● DQN_SpaceInvaders-v4_820e1_00007_7_double_q=False,dueling=False,lr=0.001_2022-04-23_20-35-10
● DQN_SpaceInvaders-v4_820e1_00008_8_double_q=True,dueling=True,lr=0.0001_2022-04-23_20-50-22
● DQN_SpaceInvaders-v4_820e1_00009_9_double_q=False,dueling=True,lr=0.0001_2022-04-23_21-05-21
● DQN_SpaceInvaders-v4_820e1_00010_10_double_q=True,dueling=False,lr=0.0001_2022-04-23_21-21-46
● DQN_SpaceInvaders-v4_820e1_00011_11_double_q=False,dueling=False,lr=0.0001_2022-04-23_21-38-00