CS F317 – REINFORCEMENT LEARNING

Connect Four Agent using Deep Q Learning

Ruban S[1], Tanmay Chowhan[2], Neerukonda Harsha[3], Indrashis Das[4], and Bharadwaz Rushi[5]

[1]2019A7PS0097H
[2]2018B5A71056H
[3]2019AAPS1489H
[4]2019AAPS0248H
[5]2019A7PS0111H

Table of Contents

Introduction 2
The Game and Rules 2
Reinforcement Learning Algorithm Used 2
Tools and Frameworks Used 3
Process of Training the Agent 4
Results 5
Observations and Explanations 6
Plots 7
Proposed Improvements 10
Conclusion 11
References 12
Glossary 13
Contributions of the Team 14

Introduction

This project is an implementation of a research paper in the field of reinforcement learning

Reinforcement Learning for Connect Four: https://web.stanford.edu/class/aa228/reports/2019/final106.pdf

We have employed Deep Q Learning in our agent to play against itself and learn with each move. The agent takes turns as player 1 (red) and player 2 (black), where each player plays the starting move every alternate game.

The GitHub repository link for this project is https://github.com/S-Ruban/cs-f317-project

The Game and Rules

Connect Four is a two-player connection board game, in which the players choose a colour and then take turns dropping coloured discs into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs.

Reinforcement Learning Algorithm Used

The Reinforcement Learning algorithm used to train the agent is Deep Q Learning. Deep Q Learning is a blend of the classic RL algorithm "Q Learning", and Deep Learning. Deep Q Learning is employed in this environment, since the number of state-action pairs is extremely high, and it is practically not feasible to use Q Learning to train the agent. Hence, a Deep Q Network is used to approximate the Q table.

Tools and Frameworks Used

Python is the programming language used in this project. The game environment is coded from scratch in Python. Keras framework is used for the Deep Q Network. `matplotlib` library is used to plot the graphs of the results.

The project contains four python files, viz, `c4.py`, `c4_game.py`, `c4_model.py` and `plot.py`.

`c4.py` is the main program and is to be executed to either train the agent, play against the agent as a human player, or play with another human player. This program controls the flow of the game and collects statistics of the game during training of the agent.

`c4_game.py` is the game environment that contains the rules of the game and ensures that the rules are not violated. Unlike typical environments in reinforcement learning that are imported from OpenAI Gym or other sources, this environment was coded from scratch in python.

`c4_model.py` is the code for the Deep Q Network and the Q Learning algorithm using the convolutional neural network (CNN). Here the CNN is implemented using the Keras framework. Adam optimizer is used in the CNN.

`plot.py` is the code to read the `.csv` file containing the statistics of each game played and plot the results using `matplotlib`.

Process of Training the Agent

While training the agent, we have explicitly varied hyperparameters such as discount factor(γ) and learning rate(α). The hyperparameter (ε) is implicitly varied in the code, where it starts at 1.0 and gradually decreases after each move in very small timesteps (which is another hyperparameter that can be varied). α takes on values $\{0.0001, 0.001, 0.01, 0.1\}$ for γ values $\{0, 0.5, 1\}$. For every other set of hyperparameters, $\alpha = 0.001$. γ is varied from 0 to 1 in intervals of 0.1.

The agent plays 624 games against itself for every set of hyperparameters, 312 as red, and 312 as black. After every move, the agent "inverts" its perspective of the game, allowing it to see the game from both players' perspective. The player that wins a game gets a reward of +1, and a reward of -1 on losing a game. There is 0 reward for every other move (including the moves that result in a tie). The weights of the various edges in the Deep Q Network are stored and updated in a file. There is also a file that has a memory of moves played which is used in the training period.

After every game, the program records the result of the game, number of moves played, and other statistics related to the neural networks and dumps it into a `.csv` file. This file is then read from another program to extract information such as win percentage when playing first and number of moves played every game, and plots it as a function of the number of games played.

Given the resources we had access to, it took quite some time to train the agent, with the training time for each set of hyperparameters going up to 3 hours. Due to this, we have limited information about the variation of hyperparameters. Nevertheless, it is enough to arrive at a few conclusions.

Results

γ	α	Red Wins	Black Wins	Red Win (First Move) %	Black Win (First Move) %	Average Number of Moves
0	0.0001	326	299	54.81%	50.32%	19.5808
	0.001	304	321	46.80%	49.68%	19.9664
	0.01	320	305	54.17%	51.61%	21.3696
	0.1	322	303	49.36%	46.47%	22.024
0.5	0.0001	311	314	47.76%	48.40%	20.8336
	0.001	318	307	51.60%	50.00%	19.9552
	0.01	327	298	55.45%	50.64%	21.1328
	0.1	302	323	48.08%	51.60%	21.3952
1	0.0001	330	295	50.32%	44.55%	20.5648
	0.001	305	320	53.21%	55.45%	20.6544
	0.01	323	302	51.92%	48.72%	22.0752
	0.1	297	328	47.76%	52.89%	21.5568
0.9	0.0001	314	311	48.40%	48.08%	21.2256
0.8	0.0001	312	313	54.49%	54.49%	22.0048
0.7	0.0001	320	305	44.55%	51.28%	21.1072
0.6	0.0001	291	334	43.91%	51.60%	21.024
0.4	0.0001	320	305	50.64%	48.40%	21.5904
0.3	0.0001	310	315	50.96%	51.60%	20.7824
0.2	0.0001	305	320	46.80%	49.36%	21.4992
0.1	0.0001	294	331	50.64%	56.41%	21.4528

Observations and Explanations

From the above table, it is noticed that both the win rate of player 1 and player 2 when playing the starting move, and the average number of moves per game converges to a constant value.

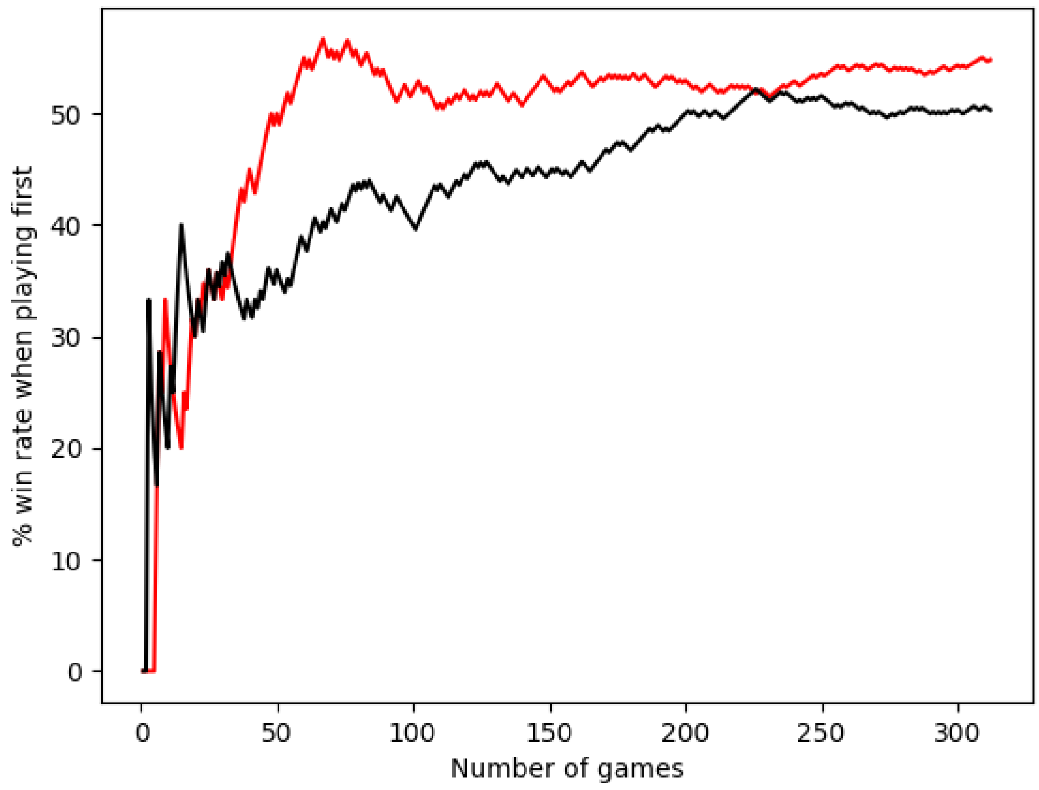
The win rate of the agent when playing the starting move is distributed between 45% and 55%. The average number of moves per game is distributed between 19 and 21. This varies from the above-mentioned paper, where the win rate is distributed between 60% and 70%, while the average number of moves per game is distributed between 18 and 19. The win rate and average number of moves per game saturates after a few games.

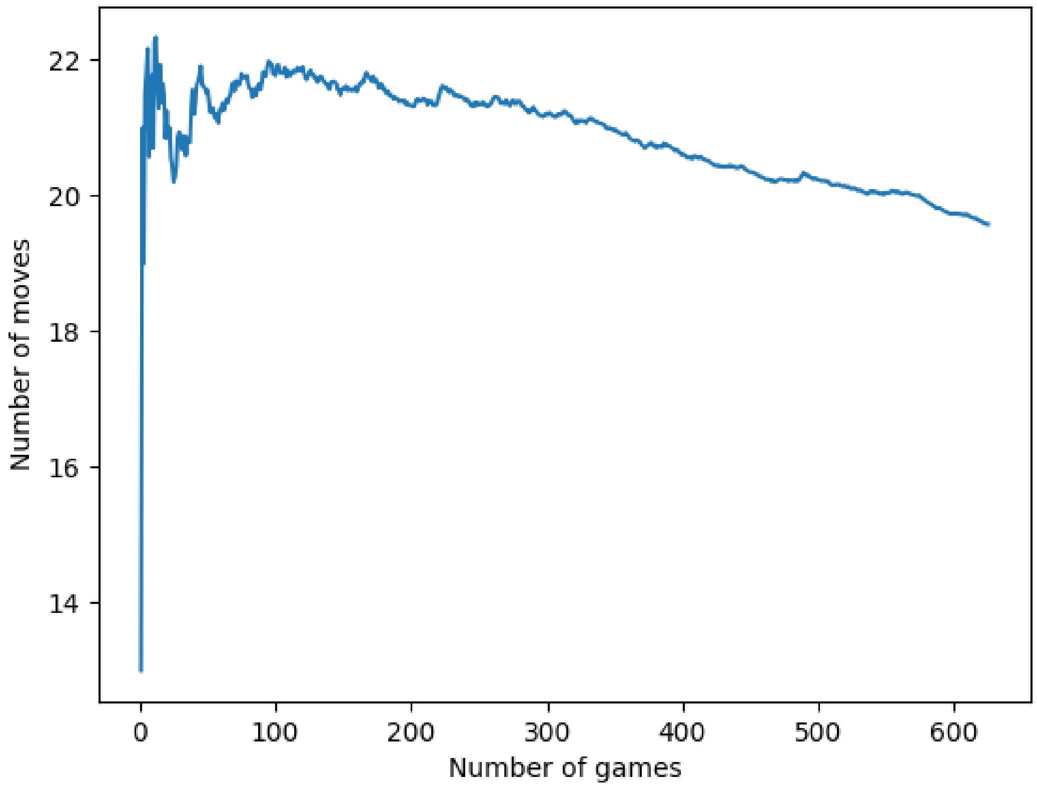
One possible explanation is the computing constraint our team is facing. Since we are using a CNN to approximate our Q function for Q Learning, it requires more computing power, which we did not have access to.

Another possible explanation is the algorithm itself. Having a look at the plots, one would notice that the win rate starts saturating at around 200 games. This would mean that there is a limitation in the code itself. While Q Learning is an effective reinforcement learning algorithm, there exist improvements to this algorithm, which could potentially increase the win rate of the algorithm.

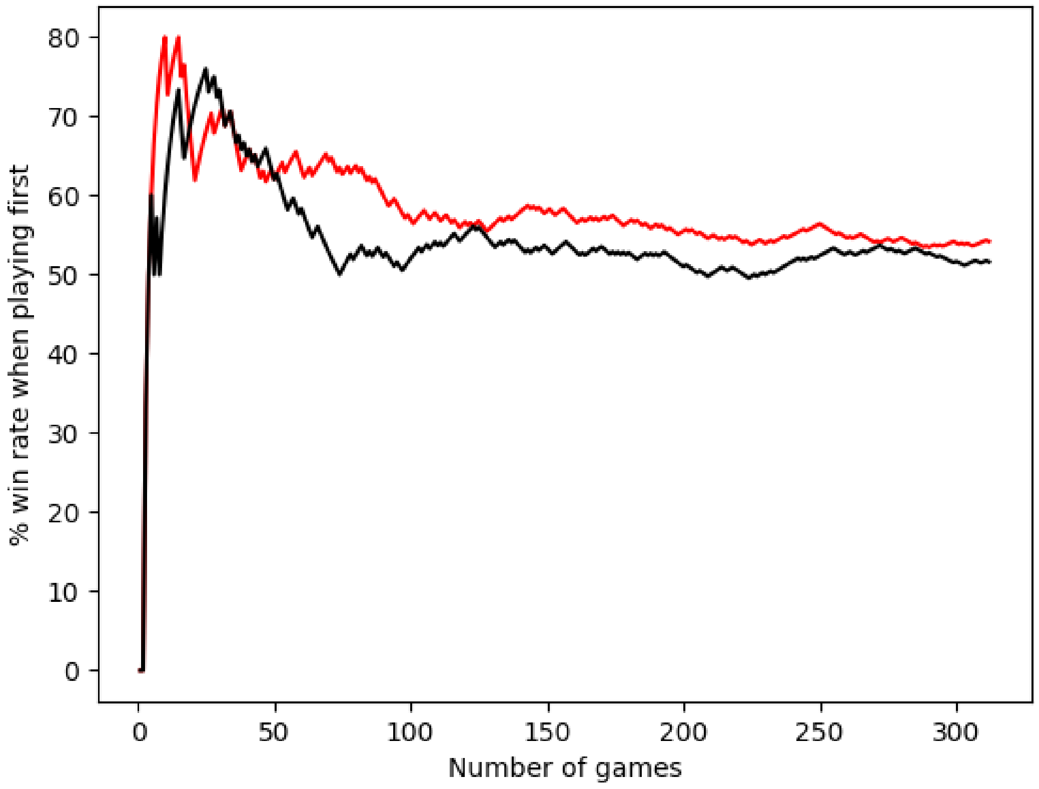
Plots

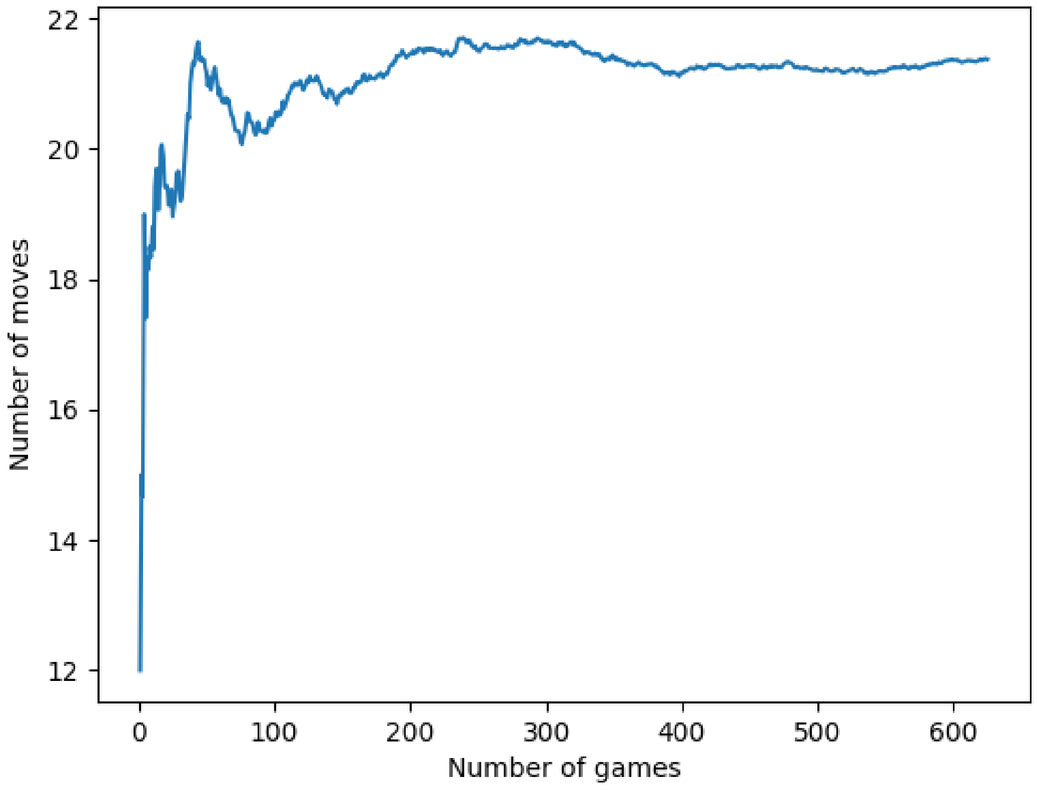
$\gamma = 0, \qquad \alpha = 0.0001$



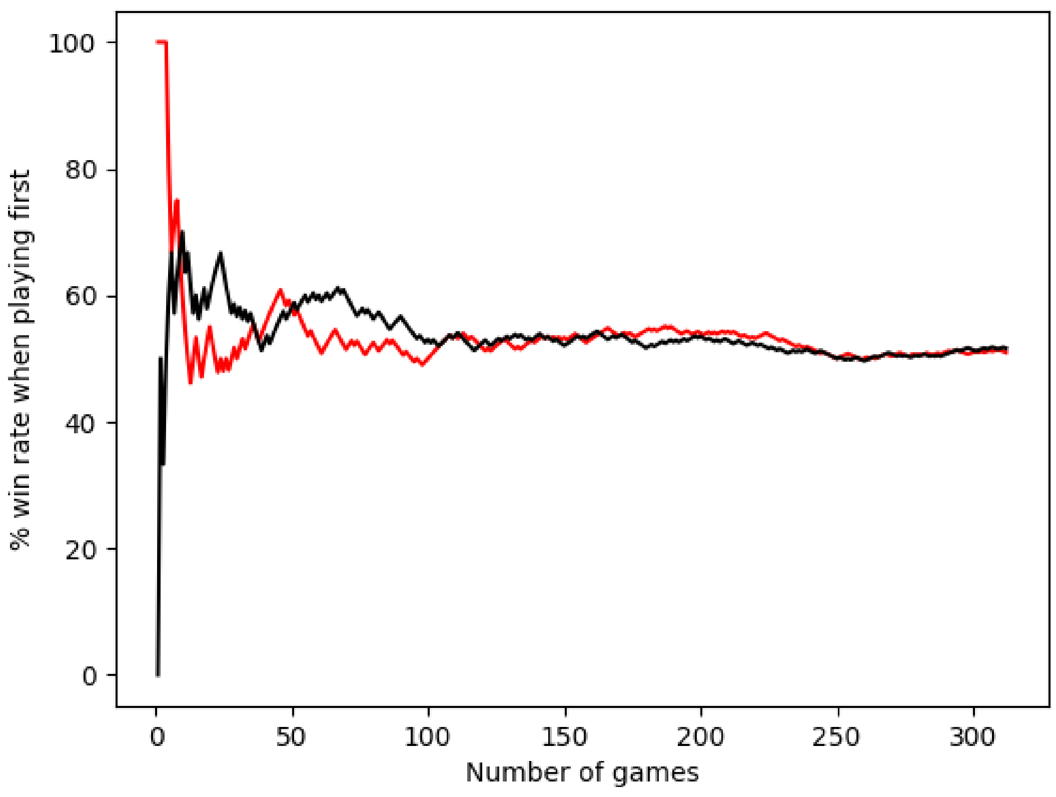


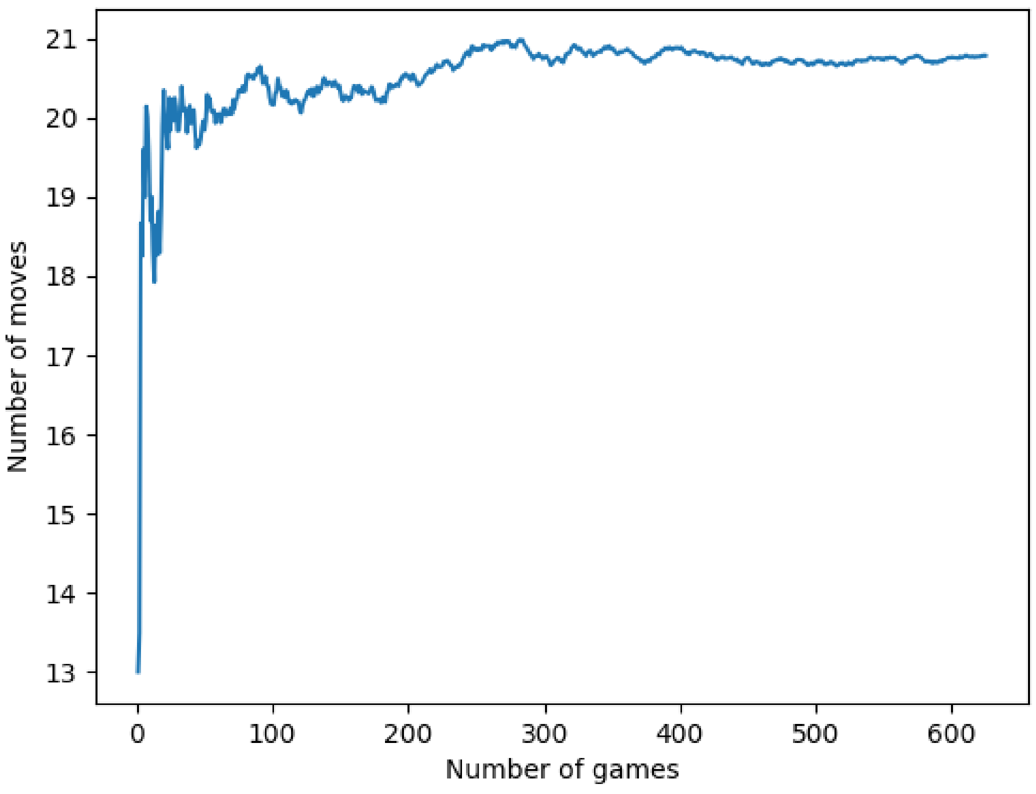
$\gamma = 0, \qquad \alpha = 0.01$





$\gamma = 0.3, \qquad \alpha = 0.0001$





Proposed Improvements

Given that an improvement in the algorithm can improve the win rate, we propose the following improvement suggestions:

1. Fixed Q Targets: Instead of updating the same DQN, we have a duplicate DQN, and update it the same way we do to our original network, after a specific number of timesteps.
2. Double Deep Q Networks: Instead of having one Q function that is updated after every timestep, we use two Q functions. We use one Q function to select the optimal action, but at the same time update the Q value of the other function.
3. Dueling DQN: A Dueling Network is a type of Q-Network that has two streams to separately estimate (scalar) state-value and the advantages for each action. Both streams share a common convolutional feature learning module. The two streams are combined via a special aggregating layer to produce an estimate of the state-action value function Q.
4. Prioritized Experience Replay: Prioritized Experience Replay is a type of experience replay in reinforcement learning where we more frequently replay transitions with high expected learning progress, as measured by the magnitude of their temporal-difference (TD) error.
5. Rainbow DQN is an extended DQN that combines several improvements into a single learner. Specifically:
    - Double Q-Learning to tackle overestimation bias.
    - Prioritized Experience Replay to prioritize important transitions.
    - Dueling networks.
    - Multi-step learning.
    - Distributional reinforcement learning
    - Noisy linear layers for exploration.

However, given the computing constraints, time constraints and knowledge constraints, it was a challenging task to implement improvements and hence, our team was unable to implement any of the above-mentioned improvements. However, this makes for an interesting project that can be done at our own pace during the holidays.

Conclusion

After running the code for various set of hyperparameters, we have arrived to the conclusion that our model achieves a win rate of 50% and each game, on average, takes 20 moves.

Given that both player 1 and player 2 have similar win rates for most of the sets of hyperparameters, this implies that the model isn’t biased towards the player that makes the first move.

The hyperparameters themselves seem to have little effect to the result, especially ε. The gradient of ε does not have much of an effect on the outcomes as well.

The current learning algorithm (Deep Q Learning) is good, but can be improved by few methods.

References

- Reinforcement Learning for Connect Four: https://web.stanford.edu/class/aa228/reports/2019/final106.pdf
- Wikipedia
- Geeks for Geeks
- https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc
- Keras CNN Documentation: https://keras.io/api/layers/convolution_layers/
- https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/
- https://towardsdatascience.com/double-deep-q-networks-905dd8325412
- https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751

Glossary

Name	Description
Action	Actions are the Agent's methods which allow it to interact and change its environment, and thus transfer between states.
Agent	The learning and acting part of a Reinforcement Learning problem, which tries to maximize the rewards it is given by the Environment.
Convolutional Neural Network	A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other.
Deep Q Learning	Deep convolutional neural network, with layers of tiled convolutional filters to mimic the effects of receptive fields.
Deep Q Network	Deep Q Networks (DQN) are neural networks (and/or related tools) that utilize deep Q learning in order to provide models such as the simulation of intelligent video game play.
Discount Factor	The discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.
Environment	Environment is the Agent's world in which it lives and interacts.
Keras	Keras is an open-source software library that provides a Python interface for artificial neural networks.
Learning Rate	Learning rate is defined as how much you accept the new value vs the old value.
Neural Network	A neural network is a network or circuit of neurons.
Q Learning	Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state.
Q Table	Q-Table is a simple lookup table where we calculate the maximum expected future rewards for action at each state.
Reinforcement Learning	Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.
Reward	A reward in RL is part of the feedback from the environment.
State	States are a representation of the current world or environment of the task.

Contributions of the Team

Team Member	Contribution
Ruban S	Writing code for the game environment, preparing the report
Tanmay Chowhan	Writing code for the Deep Q Network
Neerukonda Harsha	Testing of the code
Indrashis Das	Testing of the code
Bharadwaz Rushi	Helped in preparing the report