# Learning to be a Space Shooter Bot : Simulating a game using Deep Reinforcement Learning

Akansha Agarwal and Rishabh Garg
Department Of Computer Science And Engineering
Texas A&M University
Github and Video Link

***Abstract*—Shooter games have been a part of Reinforcement Learning research since the past few years. There have been various implementations of Shooter games where researchers have demonstrated application of Reinforcement Learning in similar games and have improved the efficiency of shooter bots using different techniques. In this work, a Deep Reinforcement Learning bot is trained to kill aliens in a custom Space Shooter game. The objective of the bot is to kill aliens using bullets, which will serve as a positive reward for the bot. If the bot dies by collision with alien or if alien crosses the game over boundary, a negative reward will be assigned to the bot. The bot is tested by tuning parameters of the model as well as the game environment. The results obtained show significant improvement in behavior of the bot and it is observed to perform comparable to a human player.**

***Index Terms*—Space Shooter Game, Deep Reinforcement Learning, Q-Learning**

## I. INTRODUCTION

Reinforcement Learning is an area of machine learning where models are trained using a system of reward and punishment by interacting with their environment in order to make them learn specific tasks. Deep Neural Networks have been combined with Reinforcement Learning to give way to special algorithms which are seen to perform better than most of the Reinforcement Learning algorithms and this sub-area is known as Deep Reinforcement Learning. Deep Q-Networks and Double Deep Q-Networks are some of the best known Deep Reinforcement Learning algorithms which integrate Q-Learning in Deep Neural Networks.

The main idea is to use Deep Reinforcement Learning algorithm for training the bot and accordingly tune the hyper-parameters with the motive of encouraging our bot to increase the game score. The game environment includes the following main features, a game agent, which is a space ship shooting bullets with motive of killing the aliens, states, which is the current situation fighter-plane is in (alive or dead), actions, which are the possible actions for space ship, i.e. move left and move right (the bullet-shooting has been kept automatic) and rewards, which are positive for killing aliens and negative for getting killed by aliens(Game Over).

In the game environment, various environment changes and tuning is performed with respect to the Deep Reinforcement Learning model as well as the game environment. A Comparison of performance and effect of Deep Q-Network [1] and Double Deep Q-Network [1] on the Reinforcement Learning bot is studied. Deep methods are preferred for bot creation as opposed to other Reinforcement Learning algorithms because it uses deep neural network as its function approximator [2] and learns directly from raw images, which results in superior performance as compared to other algorithms. For both the models, hyperparameter tuning is performed with respect to the layers of the deep leaning networks, batch size selected from the replay memory, size of the replay memory, learning rate, discount factor, epsilon for policy creation, loss functions used by model, number of iterations in training the model and number of steps after which the target model is updated. Initially the bot is given sparse rewards for killing the aliens. To further study the effect of giving rewards in different situations, the bot is given additional dense rewards. Comparison of performance of bot is performed by tuning various game environment parameters like the speed of aliens, speed of the player, speed of the bullets and the number of aliens. These parameters help in learning about the performance of bot in various game environments.

## II. RELATED WORK

Researchers have worked on building reinforcement learning bots for understanding the working and implementation of shooter games.

Game development was done using a python library, pygame. For learning pygame , [3] was referred to understand its working. For developing the learning environment, [4] was referred to learn about the structure and implementation of the game.

In [5], the authors have mentioned the importance of Deep Q-Networks. They demonstrated using 2600 Atari games that DQN can outperform all other previous reinforcement learning algorithms like SARSA, TD learning and Q-learning. The DQN was able to outperform professional human gamers for a lot of games. Since DQN performs better than all other algorithms in gaming environment, we will be using this model in our game. The authors have chosen some standard hyperparameters based on some games and used these for all other games. They observed that even without any prior knowledge, the DQN performed better than other algorithms, and if properly tuned they can give even better results. The authors introduced the algorithms for training and testing DQN as well as the usage of replay memory and various hyperparameters.

Double Q-learning was introduced as an improvement over Q-learning which tends to overestimate action values in certain conditions. After the introduction of DQN, in [6], the authors integrated Double Q-learning with Deep Reinforcement Learning. The authors have proposed a specific adaptation to the DQN algorithm which not only reduces the observed overestimations but also leads to much better performance on several games. In standard DQN, the max operator uses the same values to select as well as evaluate the action, which results in overestimation of values. In Double DQN, the authors used two value functions, out of which one is used to determine the greedy policy and other is used to calculate the weight. These functions tend to have different set of weights, which help in preventing overfitting. The authors have implemented Double DQN which finds better policies and therefore obtaining new state-of-the-art results on 2600 Atari games. From this paper, we can clearly understand the advantage of using Double DQN over DQN algorithm.

In [7], the authors have implemented a simple virtual environment with walls, health items, ammo and spawn points. They aimed to investigate the behaviour of Reinforcement Learning algorithms for training the first person shooter bot. Since we are developing a shooter game, this paper helped us in understanding the basics of implementing such games. They applied tabular SARSA algorithm which performed better as it learns the action-selection mechanism in the environment. According to them algorithms like TD-lambda are good at learning state transitions but generally require an extrinsic state action mapping mechanism. The authors have implemented tabular SARSA, with epsilon-greedy policy, varying eligibility traces and linearly decreasing learning rate. A small penalty was assigned if bot collided with walls, small reward was given when bot moved and large reward was given when bot collected an item. The objective of reward function was to minimize collision, maximize distance travelled and maximize the number of items collected. The authors learned that SARSA can be successfully applied to shooter games where smaller values for eligibility traces gave better results since the best solutions do not require a lot of planning, the discount factor did not cause any significant change in the results.

As a extension of [7], the authors published [8], where they incorporated certain methods using which a combination of tasks can be controlled using the bot, which include navigation, collection of items as well as killing the enemies in first person shooter games. For this they used three reinforcement learning setups - Flat RL (State Machine bot), Hierarchical RL and Rule based RL. In State Machine bot the objective is to fight the enemy when they are in sight with continuous navigation. In hierarchical RL, the motive is to assign rewards to the bot such that it learns to choose between fighting and navigating. In Rule based RL, bot tries to eliminate the enemy when it is in range else it continues to navigate the environment. Similar to [7], tabular SARSA was used. They observed that hierarchical and rule based RL performed better than flat RL since the flat RL was not using any navigation intelligence. Rule based RL performed better than hierarchical RL in terms of number of kills however hierarchical RL performed better in complex situations such as fleeing the scenario in case of multiple enemies wherein rule based RL would aim to kill all the enemies instead. The paper helped in analyzing the importance of navigation as well as fighting in similar shooter games. Our model is based mainly on the number of kills, for which rule based RL can be analyzed and implemented.

In [9], the authors have explained the mathematical principles which lead to success of DQN models over other RL algorithms. They have used a linear representation for Deep Q-Networks to evaluate it's performance. Using these equations, a feature set is formed which achieves a performance comparable to DQN. Using the feature sets extracted by the authors, they build a model called Blob-PROST which performs almost as good as DQN. The authors suggest that it is possible to implement certain linear methods which can perform as good as DQN, and are simple as well as faster compared to these deep reinforcement learning algorithms.

In [10], authors discuss playing first person shooter game using the pixels on screen. It compares the baseline DRQN model (Deep Recurrent Q-Network) with a DQN(Deep Q-Network) model which consists of individual network for each phase of the game. The DRQN model performed well in simple tasks which require enemies to be in line of sight but fail to provide good results in complex scenarios such as a death match. Then the authors switched to a DQN model which consisted of DRQN model for action phase and a separate DQN model for the navigation phase. The new model was seen to perform better than DRQN in all the simple as well as complex scenarios. The authors discussed the tuning of frame rates, in which higher frame rates tend to hurt the performance of models. From this paper, we get an understanding of the performance of deep recurrent networks and integration of separate deep networks dedicated to each task to form a resultant bot which can handle difficult scenarios in shooter games.

## III. METHODS

### *A. Model 1: Deep Q-Network*

Q-Learning is a well known Reinforcement Learning algorithm that makes use of expected reward of each action at every step. It used the maximum Q-value from the next state to calculate the current Q-value. It is a powerful algorithm which has applications in various domains of machine learning. Deep Neural Networks and Q-Learning are two good algorithms of their respective domains which were combined to built a more powerful model called Deep-Q Network (DQN).

In [5], the authors introduced DQN and demonstrated it's importance by demonstrating it on 2600 Atari games DQN was able to outperform all other previous reinforcement learning algorithms like SARSA, TD learning and Q-learning.

DQN uses deep neural network as its function approximator [2] and learns directly from raw images, which results in superior performance as compared to other algorithms. One of the main features of DQN is it's use of replay memory which

randomly samples data from all previous transitions and trains the network on these samples. This helps in smoother training distribution [1] because of decorrelation of data due to random sampling from a larger set of transitions.

**Algorithm 1:** Deep Q-Network

**Result:** Fully Trained Deep Q-Network
initialize a replay memory R to capacity N
initialize the model $Q$ with random weights
initialize the target model $Q'$ with random weights
initialize a state sequence $s$
**while** $i <$ *total iterations* **do**
  choose an action using $\epsilon-$greedy policy
  Execute the action and store the reward $r$ and next state $s'$
  Store the transition in replay memory
  Sample random minibatch of transitions from R
  Calculate actual labels $y$ as $r$ **if** the transition is for terminal state **else** $r + \gamma * max_{a'}\{Q'(s')\}$
  Calculate training data $q$ as $Q(s)$.
  Perform optimization for the model $Q$.
  After C steps, update the target model as $Q' = Q$
**end**

### B. Model 2: Double Deep Q-Network

Double Q-learning was introduced as an improvement over Q-learning which tends to overestimate action values in certain conditions. After the introduction of DQN, in [6], the authors integrated Double Q-learning with Deep Reinforcement Learning and proposed the Double Deep Q-Networks. The authors have proposed a specific adaptation to the DQN algorithm which not only reduces the observed overestimations but also lead to much better performance. In DQN, the maximum operator uses the same values to select as well as evaluate the action, which can result in overestimation of values. In Double DQN, the authors used two value functions, out of which one is used to determine the greedy policy and other is used to calculate the weight. These functions tend to have different set of weights, which prevents overfitting.

### C. Methodology

Our work consists of two main parts - the game environment and the Deep Q-Networks. The game environment is a custom space shooter game where our agent which is a space ship has to kill as many aliens as possible before it crashes. This work focuses more on the training part rather than building a sophisticated game, hence the game environment is such that it allows proper learning of the effect various game parameters have on the learning of bot.

*1) The Space Game Environment:* The game environment is developed using pygame, a python game development library. Figure 1 shows the game environment built for this study. The main components of the game include a space ship which shoots automatic bullets and few aliens which should be killed to gain reward. The number of aliens defined in the game can be tuned to evaluate it's effect on the bot's performance. The current score is displayed on the screen and the game gets over when either one of the alien collides with the space ship or the aliens cross the game over boundary.

**Algorithm 2:** Double Deep Q-Network

**Result:** Fully Trained Double Deep Q-Network
initialize a replay memory R to capacity N
initialize the model $Q$ with random weights
initialize the target model $Q'$ with random weights
initialize a state sequence $s$
**while** $i <$ *total iterations* **do**
  choose an action using $\epsilon-$greedy policy
  Execute the action and store the reward $r$ and next state $s'$
  Store the transition in replay memory
  Sample random minibatch of transitions from R
  Calculate actual labels $y$ as $r$ **if** the transition is for terminal state **else** $r + \gamma * Q'(s', max_{a'}\{Q(s')\})$
  Calculate training data $q$ as $Q(s)$.
  Perform optimization for the model $Q$.
  After C steps, update the target model as $Q' = Q$
**end**

Specific speeds are defined for the player which is the space ship, the bullet that is automatically fired and for the moving aliens. These can be tuned to observe the effect of change in speed of various elements of the game on the learning of the bot. Additionally, various functions are defined to control the effect of collision of bullet with the alien and for executing one step of the game when required by the Deep Q-Networks.

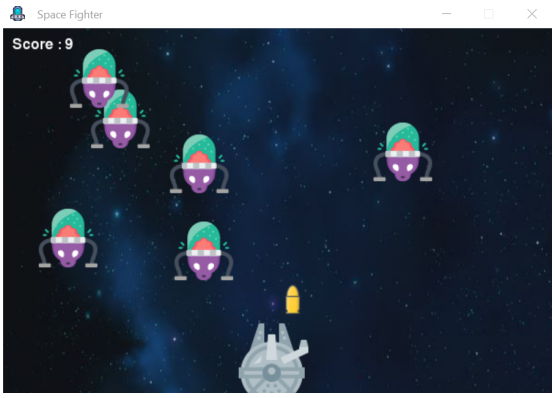


Fig. 1. Game Environment

*2) The Deep Reinforcement Learning Networks:* The Deep Q-Networks are implemented using python's torch library. Two implementations of Deep Reinforcement Learning Networks have been done - Deep Q-Network and Double Deep Q-Network. Each implementation used two Neural Network models - one current Network which is optimized with each transition in the game environment and one target network used to estimate the future result. The target network is updated after certain steps with the weights of current model.

The models take the screenshot of the game as input and by processing the pixels of the image try to learn the pattern in the game. Various parameters are defined for the models - replay memory for choosing random mini-batch of transitions in each iteration to train the current model, epsilon for selecting the action using epsilon-greedy policy, gamma for keeping track of the decay in rewards, step size for updating the target model and the mini-batch size to be selected from replay memory.

While training the model, scores and high score are tracked to measure the performance of the model. In testing, the saved model is used to predict the action the bot should take given the current state as input to the target model. The performance of the bot is measured using the averaged scores the bot is able to obtain during testing of these models.

## IV. EXPERIMENTS AND RESULT

### A. Hyperparameter Tuning : With Respect to Model

- **Iterations**: Iterations correspond to the duration the model was trained for. For DQN, highest scores were achieved for iterations = 150000, whereas for DDQN highest scores were achieved for iterations = 200000. This also holds if average scores are considered. The other parameters were assumed to be on their default chosen values where $\epsilon = 0.1, C = 100, \gamma = 0.99$ and batch size =32. After 150000 iterations, the DQN model started giving poor scores due to overfitting whereas the DDQN model provided best results for 200000 iterations.

- **Gamma**: Gamma refers to the discount factor used in the Q-Learning formula. For DDQN, highest scores were achieved $\gamma$= 0.99.This also holds if average scores are considered. Note that these values are taken for iterations =150000 and iterations =200000 since these number of iterations performed the best as stated above.

- **Batch Size**: Batch Size refers to the number of states passed as input to the Neural Network(DDQN). For DDQN, highest scores were achieved for batch size=32.This also holds if average scores are considered. Note that these values are taken for iterations =150000 and iterations =200000 since these number of iterations performed the best as stated above.

- **C**: C refers to the number of iterations after model parameters are copied to the target model. For DDQN, highest scores were achieved for C= 100 for iterations = 200000 and C=200 for iterations = 150000.This also holds if average scores are considered. Note that these values are taken for iterations =150000 and iterations =200000 since these number of iterations performed the best as stated above.

- **Epsilon**: Epsilon refers to the value of probability used in epsilon greedy policy which is proportional to the probability of choosing exploration over exploitation. For DDQN, highest scores were achieved for $\epsilon$= 0.1.This also holds if average scores are considered. Note that these values are taken for iterations =150000 and iterations =200000 since these number of iterations performed the best as stated above.

TABLE I
NUMBER OF ITERATIONS (DDQN)

| Method | Iterations | Average Score | High Score |
|---|---|---|---|
| DQN | 50000 | 38.58 | 137 |
| DQN | 100000 | 34.72 | 164 |
| DQN | 150000 | 60.64 | **398** |
| DQN | 200000 | 9.63 | 32 |
| DDQN | 50000 | 45.09 | 207 |
| DDQN | 100000 | 7.62 | 19 |
| DDQN | 150000 | 4.14 | 9 |
| DDQN | 200000 | **71.96** | 343 |

In the above table, iterations count is tuned (where default parameters are $\epsilon = 0.1, C = 100, \gamma = 0.95$ and batch size =32, and the scores are averaged over 100 test game plays of all models)

TABLE II
NUMBER OF ITERATIONS = 150000 (DDQN)

| $\epsilon$ | Batch | $\gamma$ | C | Avg Score | High Score |
|---|---|---|---|---|---|
| 0.1 | 32 | 0.99 | 100 | 4.14 | 9 |
| 0.1 | 64 | 0.99 | 100 | 17.7 | 56 |
| 0.1 | 32 | 0.99 | 200 | **62.09** | **289** |
| 0.1 | 64 | 0.99 | 200 | 26.49 | 122 |
| 0.1 | 32 | 0.95 | 100 | 19.08 | 41 |
| 0.1 | 64 | 0.95 | 100 | 55.6 | 263 |
| 0.2 | 32 | 0.99 | 100 | 38.49 | 179 |
| 0.2 | 64 | 0.99 | 100 | 49.05 | 284 |

In the above table, $\epsilon, C, \gamma$ and batch size are tuned for iterations = 150000 using DDQN. (The scores are averaged over 100 test game plays of all models and the high score is the maximum score obtained out of these 100 test game plays.)

TABLE III
NUMBER OF ITERATIONS = 200000 (DDQN)

| $\epsilon$ | Batch | $\gamma$ | C | Avg Score | High Score |
|---|---|---|---|---|---|
| 0.1 | 32 | 0.99 | 100 | **71.96** | **343** |
| 0.1 | 64 | 0.99 | 100 | 59.4 | 199 |
| 0.1 | 32 | 0.99 | 200 | 37.97 | 160 |
| 0.1 | 64 | 0.99 | 200 | 37.48 | 192 |
| 0.1 | 32 | 0.95 | 100 | 23.93 | 224 |
| 0.1 | 64 | 0.95 | 100 | 8 | 18 |
| 0.2 | 32 | 0.99 | 100 | 15.12 | 112 |
| 0.2 | 64 | 0.99 | 100 | 4.26 | 12 |

In the above table, $\epsilon, C, \gamma$ and batch size are tuned for iterations = 200000 using DDQN.(The scores are averaged over 100 test game plays of all models and the high score is the maximum score obtained out of these 100 test game plays.)

### B. Hyperparameter Tuning : With Respect to Game

- **Fixed Reward on killing aliens [R1]**: In the approach, a fixed reward = 1 was given for killing each alien irrespective of the position of the alien in the game frame and punishment was given in the form of reward

= -100 when the bot dies. This performed poorly with respect to the below approach as the bot was unable to learn about the priority of killing an alien closer to it compared to an alien far away, and hence was getting killed.

- **Reward on killing based on alien's position [R2]**: In the approach, a reward proportional to the y coordinate of the alien in the current game frame was given. The logic used was that the space-shooter should kill the aliens closer to it with more probability than the aliens which are farther away to increase the score and decrease the probability of itself getting killed. This performed way better than the above approach as the bot was able to perceive the danger from an alien that was closer to it compared to an alien far away. It focused on killing the alien posing greater danger compared to others.

- **Dense Reward based on bot's sustenance [R3]**: The model was tuned with and without giving a reward = 0.1 when the bot was alive in addition to the reward based on killing the aliens. The results were not as good as expected when this dense reward was given.

- **Number of aliens**: The model was tuned for the number of aliens in the frame at any given time, since each time an alien is killed a new alien appears in the frame randomly. This was tuned for alien count =6 and alien count =10. Increasing the number of aliens does not provide good results as the bot is not able to handle multiple aliens at high speed due to speed and bullet restrictions.

- **Speed Of Bullet [B-spd]**: The model was tuned for the speed of bullet in the y-axis in the frame at any given time. This was tuned for speed of bullet =10 and speed of bullet = 20. As expected, increasing the speed of bullet increased our game scores since it killed the aliens at a much greater rate. In the tables provided, higher average score can be observed when only the bullet speed is increased since now the bot is able to kill more aliens compared to slower bullet speed.

- **Speed Of Aliens [A-spd]**: The model was tuned for the speed of aliens in the x-axis in the frame at any given time. This was tuned for speed of aliens =5 and speed of bullet = 10.We observed that increasing the speed of aliens increased our game scores.

- **Speed Of Space-Shooter Bot [P-spd]**: The model was tuned for the speed of space-shooter bot in the x-axis in the frame at any given time. This was tune for speed of space-shooter = 5 and speed of space-shooter = 10. As expected, increasing the speed of bot increased our game scores since the bot's movement rate and therefore the rate of killing aliens increased.

TABLE IV
NUMBER OF ITERATIONS = 150000 (GAME)

| Reward | Aliens | A-spd | B-spd | P-spd | Avg Score | Highest |
|---|---|---|---|---|---|---|
| R1 | 6 | 5 | 10 | 5 | 20.81 | 106 |
| **R2** | 6 | 5 | 10 | 5 | 62.09 | 289 |
| R3 | 6 | 5 | 10 | 5 | 44.97 | 249 |
| R2 | 6 | 5 | 20 | 5 | 19.69 | 49 |
| R2 | 6 | 10 | 20 | 5 | 15.30 | 64 |
| R2 | 6 | 10 | 20 | 10 | **87.41** | **319** |
| R2 | 10 | 5 | 10 | 5 | 17.72 | 31 |

In the above table, the rewards and number of aliens along with speed of alien, player and bullets are tuned for iterations = 150000 using DDQN. (The scores are averaged over 100 test game plays of all models and the high score is the maximum score obtained out of these 100 test game plays.)

TABLE V
NUMBER OF ITERATIONS = 200000 (GAME)

| Reward | Aliens | A-spd | B-spd | P-spd | Avg Score | Highest |
|---|---|---|---|---|---|---|
| R1 | 6 | 5 | 10 | 5 | 31.24 | 174 |
| **R2** | 6 | 5 | 10 | 5 | 71.96 | 343 |
| R3 | 6 | 5 | 10 | 5 | 14.43 | 61 |
| R2 | 6 | 5 | 20 | 5 | 218.17 | **2149** |
| R2 | 6 | 10 | 20 | 5 | 112.20 | 922 |
| R2 | 6 | 10 | 20 | 10 | **329.88** | 1208 |
| R2 | 10 | 5 | 10 | 5 | 24.10 | 97 |

In the above table, the rewards and number of aliens along with speed of alien, player and bullets are tuned for iterations = 200000 using DDQN.(The scores are averaged over 100 test game plays of all models and the high score is the maximum score obtained out of these 100 test game plays.)

### C. Experimental Results

The Deep Q-Networks implemented in this work take on an average 10 hours to train for 200000 iterations. The training time for models at various parameters ranges in between 9 hours to 11 hours on a device with following configurations, 2.3 GHz Intel core i5 processor and 8 GB RAM on a Macintosh Operating System.

The testing of the models obtained is performed by executing 100 game plays for each model on two set of iterations of 150000 and 200000. The average testing time for each model is one hour on an average. For model trained on 150000 iterations, testing time is 24 minutes on an average and for the model trained on 200000 iterations, testing time is 37 minutes on an average.

The best performance of our Space Shooter bot is achieved for following parameters : $\gamma$ = 0.99, $\epsilon$ = 0.1, batch size = 32, iterations = 200000, C = 100 and reward = sparse rewards based on alien position. During the training of various models, the bot was able to achieve highest score of 1443 when using the current best parameters of model and increasing the speed of bullet as well as the bot.

### D. Figures

The graphs of our model corresponding to Q-values, rewards, scores and episode length with respect to episodes are given in the following figures.

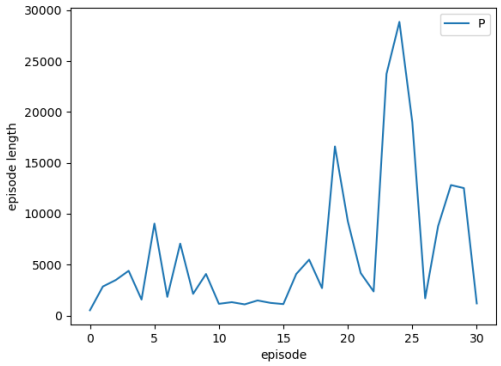


Fig. 2. In the figure, the variation of episode length can be seen with respect to each episode of the game while training our DDQN model. The episode length increases as the bot tries to learn the game and improves it's performance.

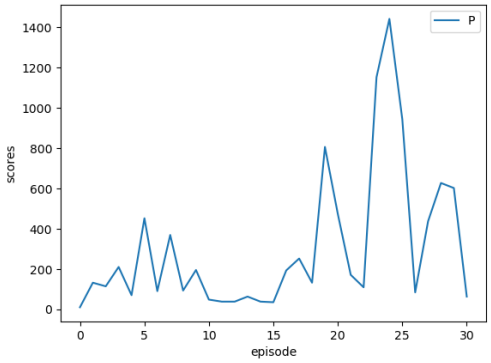


Fig. 3. In the figure, the variation of scores obtained can be seen with respect to each episode of the game while training our DDQN model. More scores mean the bot is able to kill more aliens as it learns about the game.

## V. LIMITATIONS AND IMPROVEMENTS

Our work has certain limitations, which can be overcome in future works. Since game development can be quite complex and is mostly open ended, the main focus of this work was learning about the Reinforcement Learning Algorithms and their affect on the game environment. The game environment can be further improved and bot can be trained on the improved environment. Several studies are being conducted on Deep Reinforcement Learning methods. Deep Learning has 2 major drawbacks, first being that the model may forget the previous information provided and second being the powerlessness of the models to verify the correctness of the input provided to it. New algorithms like 'Imagination-Augmented Agents (I2A)' can be used wherein the policy function is a combination of both model-free and

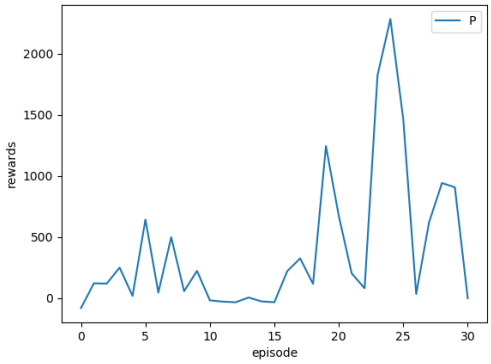


Fig. 4. In the figure, the variation of rewards given according to alien positions can be seen with respect to each episode of the game while training our DDQN model. As the game progresses, the bot is able to obtain increasing rewards.

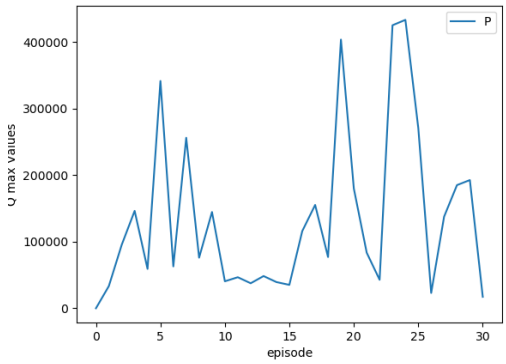


Fig. 5. In the figure, the variation of Q-values can be seen with respect to each episode of the game while training our DDQN model.

model based components. The deep neural networks used in our work use convolutional neural networks which can be replaced with various other types of networks and their results can be evaluated to check for improvements.

For improvement of the Space Shooter bot, different Deep Reinforcement Learning models can be used to study their effect on the learning of the bot. A sophisticated game environment can be developed and the performance of the bot can be tested on these environment. Given proper resources, the bot can be trained on numerous parameters and further tuning can be performed.

## VI. CONCLUSIONS AND FUTURE WORK

In this study, two Deep Reinforcement Learning Algorithms were used for training a Space Shooter Bot to kill the aliens. The bot was able to perform comparable to a human player. Tuning was done on the Deep Q-Networks as well as the game environment to study behavior of the bot in different situations. Our bot was successful in killing a decent number of aliens

given proper training and hyperparameter tuning. The results obtained met our expectations.

The bot can further be improved by using newer and more efficient models, so that the training time can be reduced and the performance can be better. Our future work consists of experimenting with better models and building a better environment for our Space Shooter bot and test it's performance in these new scenarios. Once these scenarios are studied and evaluated, this work can be expanded to various eclectic game environments, expecting positive results. The use of Deep Reinforcement Learning models for simulating the game environments has a wide scope since a lot is yet to be uncovered in this domain and the studies being done have demonstrated positive results so far.

## VII. ACKNOWLEDGEMENT

We would like to thank Dr. Guni Sharon for providing us with the opportunity to do research on such an interesting topic as part of our project. Also, we appreciate the guidance provided by James Ault in successful completion of the project.

## REFERENCES

[1] Guni Sharon. Csce 689 [reinforcement learning] - deep q-learning.

[2] Guni Sharon. Csce 689 [reinforcement learning] - deep neural nets as function approximators.

[3] Ivan Idris. *Instant Pygame for Python Game Development How-to*. Packt Publishing Ltd, 2013.

[4] Free Code Camp. Space invaders pygame. https://github.com/attreyabhatt/Space-Invaders-Pygame, 2019.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[7] Michelle McPartland and Marcus Gallagher. Learning to be a bot: Reinforcement learning in shooter games. In *AIIDE*, 2008.

[8] Michelle McPartland and Marcus Gallagher. Reinforcement learning in first person shooter games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):43–56, 2010.

[9] Yitao Liang, Marlos C Machado, Erik Talvitie, and Michael Bowling. State of the art control of atari games using shallow reinforcement learning. *arXiv preprint arXiv:1512.01563*, 2015.

[10] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.