Acknowledgement:

The success of this project would not have been possible without the support of hard coffee, assistance of heavy late-night snacks as well as the all-seeing and all-powerful Google. We are immensely blessed to have got this all along the duration of our project. We would like to extend our profound gratitude to each and every one of them.

ABSTRACT:

The advances in AI has made many tasks that only humans were capable of performing to be done by intelligent agent systems. One of such tasks is to learn direct from interacting with the given environment. Using neural networks with deep Q learning a self-learning AI-agent that can learns from interacting with its environment and tackling the obstacles to meet the objectives is developed. Our AI agent that play on superhuman level on Flappy bird game was used to demonstrate its learning capacities.

KEYWORDS:

Machine Learning, Q-learning, feedforward artificial neural network, Game Bot, Reinforcement learning.

Contents

Chapter 1: Introduction

1.1 Background:

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is considered as one of three machine learning paradigms alongside supervised learning and unsupervised learning. It differs from supervised learning in that correct input/output pair need not to be presented, and sub-optimal actions need not be explicitly corrected. Instead focus is on performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge) [1].

Using the power of reinforcement learning many types of problems can be solved. Simple solution for a complex problem can be found. And to demonstrate the power of reinforcement learning I'll be using RL to train an AI agent to do different tasks more efficiently than human.

We try to bring the popular game Flappy Bird. The game is a side-scroller where the player controls a bird, attempting to fly between rows of green pipes without coming into contact them. If the player touches the pipes, it ends the game. The bird briefly flaps upward each time the player taps the screen; if the screen is not tapped, the bird falls due to gravity. Here in this project we use python to develop this game. Some of the core concepts that is required to understand this project are discussed below in short:

1.1.1 Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is considered as one of three machine learning paradigms alongside supervised learning and unsupervised learning. It differs from supervised learning in that correct input/output pair need not to be presented, and sub-optimal actions need not be explicitly corrected. Instead focus is on performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Reinforcement learning algorithms can be categorized into types based on how it selects the next action to take:

- Value Iteration
- Policy Iteration

In policy iteration algorithms, you start with a random policy, then find the value function of that policy (policy evaluation step), then find a new (improved) policy based on the previous value function, and so on. In this process, each policy is guaranteed to be a strict improvement over the previous one (unless it is

already optimal). Given a policy, its value function can be obtained using the Bellman operator. In value iteration, you start with a random value function and then find a new (improved) value function in an iterative process, until reaching the optimal value function. Notice that you can derive easily the optimal policy from the optimal value function. This process is based on the optimality Bellman operator.

The algorithm I'll be using in this project is a type of value iteration reinforcement learning algorithm known as Q-learning.

1.1.2 Q-Learning Algorithm:

Q-learning algorithm (One of many RL algorithm) which I will be using to develop my agent involves an agent, a set of states {S} S, and a set {A} of actions per state. By performing an action {A} in A, the agent transitions from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score). The goal of the agent is to maximize its total (future) reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward. This potential reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state. Q-learning is a model-free reinforcement learning algorithm meaning it does not use transition probability distribution (and reward function) associated with Markov Decision process, which, in RL, represents the problem to be solved. The transitive probability distribution (or transition model) and the reward function are often collectively called "model" of the environment, hence the name "model-free" if the algorithm does not follow the define convention. Instead of calculating a transition probability distribution for the given decision, in Q-learning we using trial-and-error (or interaction) with the given environment (or problem) to approximate a Q function that define how obtainable the reward on the state is.

The Q algorithm has a function that calculates the quality of a state-action combination:

$Q: S \times A \rightarrow R$

Before learning begins, Q is initialized to a possibly arbitrary fixed value. Then, at each time t the agent selects an action A_t, observes a reward R_t, enters a new state S (that may depend on both previous state S_t and the selected action), and Q is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information:

$$\underline{Q^{new}(S_t, A_t) = (1 - \alpha).(S_t, A_t) + \alpha.(R_t + \gamma.\ maxQ(S_{t+1}, A_{t+1}))}$$

Here is the reward received when moving from the state S_t to the state S_{t+1}, and α is the learning rate ($0 < \alpha <= 1$). An episode of the algorithm ends when state S_{t+1} is final or terminal state. However, Q-learning can also learn in non-episode task. If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loop [2].

For all final state S_t, Q (St, A_t) is never updated, but is set to the reward value R observed for state S_t. In most cases, Q (St, A_t) can be taken to equal to zero [2].

Q-learning algorithm to decide which action is to be taken in the given state space. So basic steps for the development of the agent are as follows:

Step 1: Represent the problem as an environment in which the agent can interact and learn about the problem.

Step 2: Initialize Q-table, initial policy, initial reward matrix, exploration decay value, learning rate, initial state and the environment along with the connection with agent.

Step 3: Learn the available set of actions and take a random sample for next action.

Step 4: Update the q-value of the current state using the Bellman equation.

Step 5: Repeat step 3 and 4 until the q-table is populated properly.

Step 6: Solve the problem (reach the goal) in the environment using the calculated Q-table.

1.1.3 Artificial Neural Network:

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems process information. One main example about such a system is the human brain [9]. Composed of a number of neurons, this paradigm is the novel structure of the information processing system. Those neurons work in together to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Like other machine learning methods, neural networks have been used to solve a wide. Variety of tasks that are hard to solve using ordinary rule-based programming, including computer vision and speech recognition. With their remarkable ability to derive meaning from complicated or imprecise data, neural networks can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an" expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest. Commonly, a class of statistical models may be called" neural" if it has two properties. First, it has to consist of sets of adaptive

weights. In other words, the model needs to have numerical parameters that are tuned by a learning algorithm. Second, the class should be capable of approximating nonlinear functions of their inputs. The adaptive weights are conceptually connection strengths between neurons, which are activated during training and prediction. Neural networks are also similar to biological neural networks in performing functions collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned.

1.1.3 Adam Optimizer:

Instead of using a traditional gradient descent algorithm to optimize or converge our neural network we are using Adam optimizer. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients.

The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which Adam was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best-known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods.

1.2 Problem Statement:

Although AI is finally getting the hype it deserves, mostly it is for supervised learning and training one still needs a lot of computation power and large labeled data collection. In this project, using Q-learning concept we want to demonstrate the idea of machine learning where a machine learns to solve a problem (in this project play a game like flappy bird) not by copying a success-data feed (as in supervised and unsupervised learning) or by selective breeding of successful agent that found it success through random action.

1.3 Objectives:

The main objective of this project is to develop an intelligent agent using Q learning so that it can learn to play game 'Flappy Bird'. The intended goals of this project are as follows:

- To develop a self-learning agent that imitates the natural learning process of human.
- To develop a side-scrolling game called "Flappy Bird" as an environment in which the designed agent can be trained and tested.
- To train and test the developed self-learning agent in the developed environment and find if the agent can play as good as if not better than a human.

1.4 Feasibility Analysis:

Feasibility analysis is the practical extent to which project can be performed successfully. To evaluate feasibility of our system we performed feasibility study, which helps us to determines whether the solution considered to accomplish the requirements is practical and workable in our system.

1.4.1 Economic Feasibility:

For determining the efficiency of the new project Economic feasibility analysis is one of the most commonly used method. It is also known as cost analysis. It helps in identifying profit against investment expected from the output. To solve a relatively small problem (play a 2D game), Auto can be trained in a relatively low-end hardware, which does not cost much and can be found easily in the market.

Hence this project is feasible economically for the user and also for developer too.

1.4.2 Technical Feasibility:

This project can be trained on a on any type of decent hardware but for efficiency a processor with processing power equal to or greater than i5-5300 paired with a GPU GTX 950. Both of the major hardware can be easily found in the market. Similarly, many researches have been gone in the subject of reinforcement learning that proves the principle of the core algorithm works. To develop the 2D game for implementing our project, we have used python with pygames.

Hence, this project seems feasible in technical terms.

1.4.3 Operational Feasibility:

Operational feasibility refers to the measure of solving problems with the help of new proposed system. It helps in taking advantages of opportunities and fulfill the requirements as identified during the development of the project.

Once the agent is trained, the agent can run on personal computer with a minimal hardware.

During the period of feasibility study and research we studied the operational environment of our project and calculated the operational feasibility. In this case we studied about the operational environment and necessary resource available. Further there is no any restrictions from government regulations for implementing the system.

1.7 System Requirement:

Software Requirement:

- A working copy of Ubuntu or any Debian based Linux or any version of Windows greater than 7 will do
- Python 3
- Other libraries of python that will be used in this project

Hardware Requirement:

- A working computer
- Processor greater than i5-5300 (preferred)
- A GPU greater than GTX 950 (preferred for training)

Chapter 2: Literature Review

The concept of testing the capacity of machine learning through games is not new and the paper on Q-learning algorithm with rough set theory was published in 1981 by Zdzislaw Pawlak. Till now countless research paper and projects have been done to improve the learning. Our project can be divided in two parts, the environment and the agent. So, we have taken many in inspirations from many sources for the development of the agent as well as the environment. Both are discussed below:

2.1 The Environment – Flappy Bird:

For training and testing our self-learning agent, we have chosen a simple but wildly popular mobile game named flappy bird. Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen (Vietnamese: Nguyễn Hà Đông), under his game development company dotGears.[8] The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them. [8].

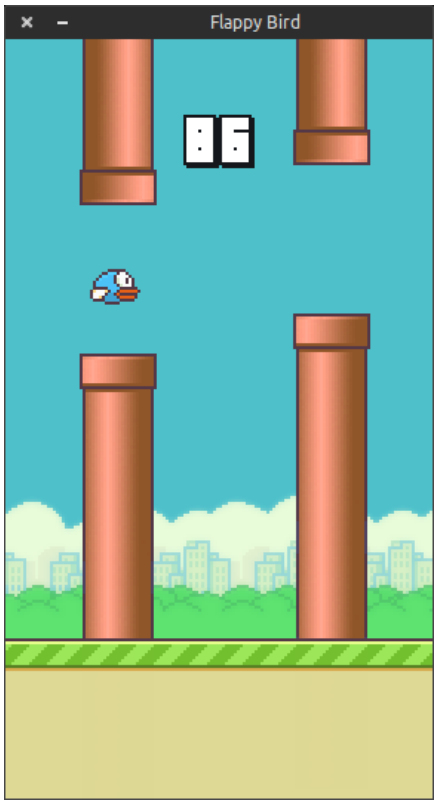

Figure 1Flappy Bird The game

1.2 The Agent:

The develop our agent we have to look into many techniques and machine learning concepts. One of the most popular implementation of reinforcement learning is the DeepMind's Atari player. This is almost real end-to-end deep learning, because the AI bot gets inputs in the same way as a human player—it directly sees the image on the screen and the reward/change in points after each move, and that is all the information it needs to decide. Please refer to the reference [4] for more in depth detail about the project. The older version of the DeepMind Atari Project uses Q-learning but with pixel input. The pixels input is pre-processed using a deep convolution neural net that scales down the input pixel resolution to a relatively small resolution intensifying the features of the given input. Using the given set of images as input, the algorithm defines states of the environment and that obtained states is used for the Q-algorithm. Google DeepMind's efforts to use Deep learning techniques to play games have paved way to looking at artificial Intelligence problems with a completely different lens. Their recent success, AlphaGo [4], the Go agent that has been giving a stiff competition to the experts in the game show clearly the potential of what Deep learning is capable of. DeepMind's previous venture was to learn and play the Atari 2600 games just from the raw pixel data. Mnih et al. are able to successfully train agents to play these games using reinforcement learning, surpassing human expert-level on multiple games [4], [7]. Here, they have developed a novel agent, a deep Q-network (DQN) combining reinforcement learning with deep neural networks. The deep Neural Networks acts as the approximate function to represent the Q-value (action-value) in Q learning. They also discuss a few techniques to improve the efficiency of training and better the stability. They use an "experience replay" of previous experiences from which mini-batches are randomly sampled to update the network so as to de-correlate experiences and delayed updates for the cloned model from which target values are obtained (explained in detail later) to better the stability. Another advantage of this pipeline is the complete absence of labeled data. The model learns by playing with the game emulator and learns to make good decisions over time. It is this simple learning framework and their stupendous results in playing the Atari games, inspired us to implement a similar algorithm for this project.

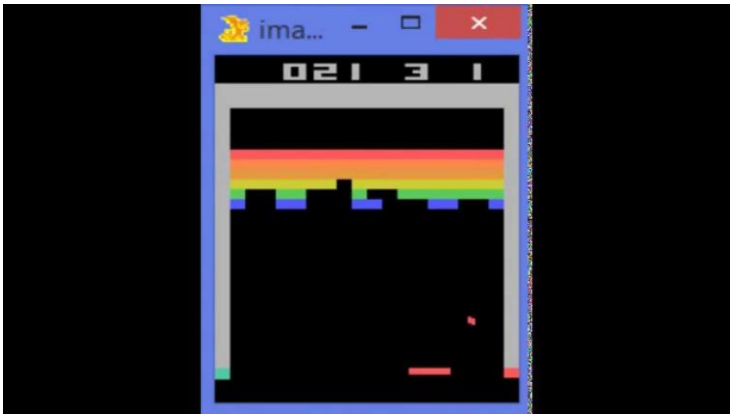

Figure 2AI playing Breakout Through Pixel Input

Although they used direct pixel input we would be using in game(environment) sensor to game data of the environment to the agent so that we don't have to take all the pixels as input to our neural network. This may require a little tweaking in the environment to make it 'AI friendly' but this reduces the required computational power to train the AI drastically.

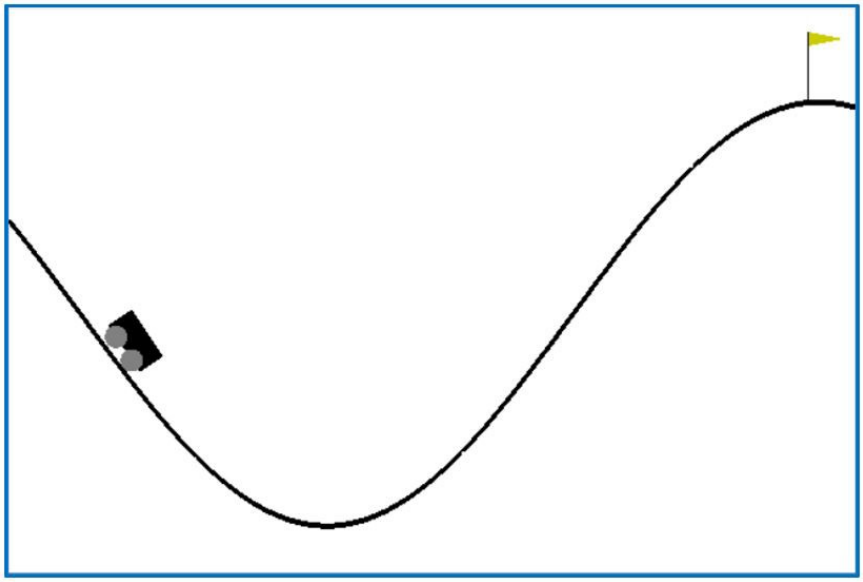

Figure 3 AI Friendly Mountain Car Env. (Open AI)

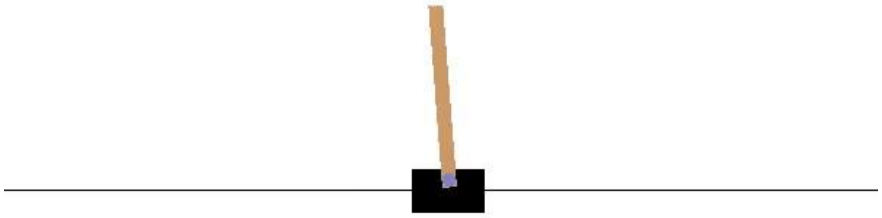

Figure 4 Open AI's Cartpole Balance Env.

Chapter 3: Methodology

Our project is divided into three parts, part one: environment development and part two: agent development and part three: Training and Testing. All of the phases are developed under iterative model of software development. Specifies of the development process of all the phases are discussed below in detail:

3.1 The Environment:

3.1.1 The base – Flappy Bird:

Our environment is the direct copy of the mobile game flappy bird. Since run a android simulation is a bit computationally heavy we decided to rewrite the whole game in python using pygames. Luckily for us the game is not mechanically complex. If "space" is pressed the bird will flap and gain some height. And the goal of the game is to cover as much as distance you can avoiding the pipes. The pipes are generated randomly with a fixed game size so that the player cannot follow a certain pattern to win or to get a high score.

3.1.2 Making the environment AI friendly:

Although the game is playable by a human, we need add some functions so that our agent can 'play' the game. To make the environment accessible to the agent, the environment must have some way to let an agent control its player as well as some way to let the agent experience of see the environment. To solve these two issues, we have implemented control function and in-game vision.

Basically, control function is a way to give the agent control of the player. Here we have defined a function *frame_step(action)* which contains all the core logic of the environment. The agent can call that function with the desired action to influence the environment. Each possible action is assigned a numerical value:

0 – Do Nothing

1 – Flap the wings (hit space bar)

The function *frame_step(action)* also returns the reward obtained from the action taken, the next state (in this case it's the sensory readings) and a done variable that denotes whether the running episode is finished or not. This gives the agent a way to execute action in the environment.

Using control function our agent can influence the environment but still has no way to see the environment. Developers of DeepMind used the pixel generated in the game as input for their agent but we used the sonar readings from the origin point of the player as the input for our agent. As shown in the screenshot below, we generated some lines from the origin point of the player, which can collide with the pipes giving it the distance between the pipe and the player. The idea is based of bat using sonar to guide it through the night. The distances are then calculated and is returned to the agent as readings through function frame_step(action).

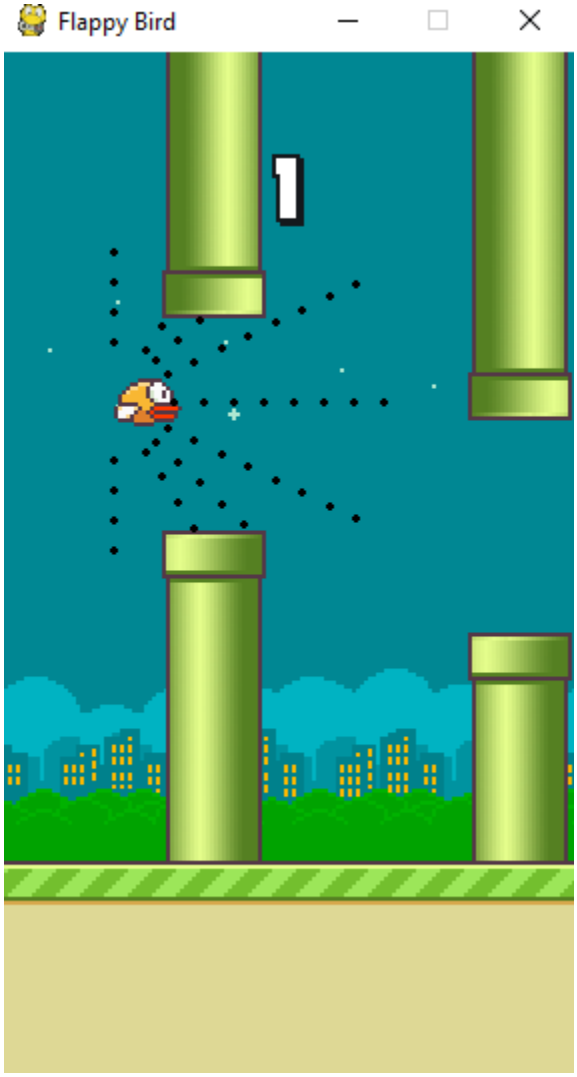

Figure 5 Agent Vision in Flappy Bird

3.2 The Agent:

Our self-learning artificial intelligent agent is designed to take in inputs from the environment and give the appropriate action which has the highest chance of getting to the goal. The agent receives the reading from the environment as a array of numerical values returned by the function *frame_step (action)* along with reward. The obtained readings and rewards are then used to calculated the next appropriate action using *get_action(readings)*. Then the cycle continues.

3.2.1 Overcoming the limitation of Q-learning:

Normal Base Q learning using lookup table to store the Q value of the state action pair but our environment is much complex and has more than 11^{12} possible state action pairs. So, we have implemented a neural network to predict a Q value of the possible actions given the current state as input. We use a feedforward neural network to predict the q-values of the possible action/s with the given state. The action with the higher Q values get executed. And the cycle continues. The technique of pairing Q-learning algorithm with a neural network is also called Deep Q learning.

The implemented feedforward deep neural network is composed up of 4 layers. The first, input layer take 11 numerical values as inputs and process it for the next layer, the second layer and third layer consists of 24 neurons each implementing rectifier as its activation function. And for the final layer, the output layer consists up of 2 neurons with a linear activation function. The whole model uses mean square error to calculate its loss and uses Adam optimizer with initial learning rate of 0.001 to decrease the loss and converge to the given target value. The defined neural network will predict the Q values from the given reading but it must have a target value to converge to during training for it to improve its prediction. So, using the **Bellman equation** with the obtained reward, we can calculate the target Q value of the required state-action pair.

q_update = reward + discount_rate np.amax (self.model.predict(next_state))*

Now we have the prediction as well as the target for our neural network to train.

Now for exploration and exploitation, we define a code segment:

```
if np.random.rand() < self.exploration_rate:
    return random.randrange(self.action_space)
```

Which will return a random action if the exploration rate is smaller than the obtained random value. The initial exploration rate is set to 1 with decay of 0.995 and minimum exploration as 0.001.

3.2.2 Learning from past experience:

To make learning more stable we used a biologically inspired mechanism termed experience replay that randomizes over the data (state-action pairs predicated q value and target q value). Thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. [4]. Basically, the agent saves the past experiences in its memory and uses these to improve its current self. The improvement of the network and the agent is not limited to current episode but also the episodes it played before.

Function *save_to_memory(self, state, action, reward, next_state, done)* saves the given data in a double-ended queue which is later extracted in the function *experience_reply()*. Data of a certain batch size (in this case 20) is randomly selected and is used for optimizing the neural network model.

3.4 Required Tools:

Tools and libraries used in the development of this project are listed below:

- **Tools:**
  - Visual Studio Code : For basic text editing
  - Python 3 : For compiling and running the written codes
  - Google Docs : For documentation
  - Git/GitHub : For source code management

- **Libraries:**
  - Pygame : To create, manage, and interact with the created environment
  - NumPy : For matrix calculation
  - Math : For calculation
  - Random : To get some randomness
  - Json : To manipulate json variables
  - TensorFlow : Creating, using and manipulating models
  - Matplotlib : Plotting data and finding in a visually pleasing way
  - Pandas : For data analysis

3.5 Use Case Diagram:

A use case is a software and system engineering term that describes how a user uses a system to accomplish a task. A use case diagram of our project is attached below:

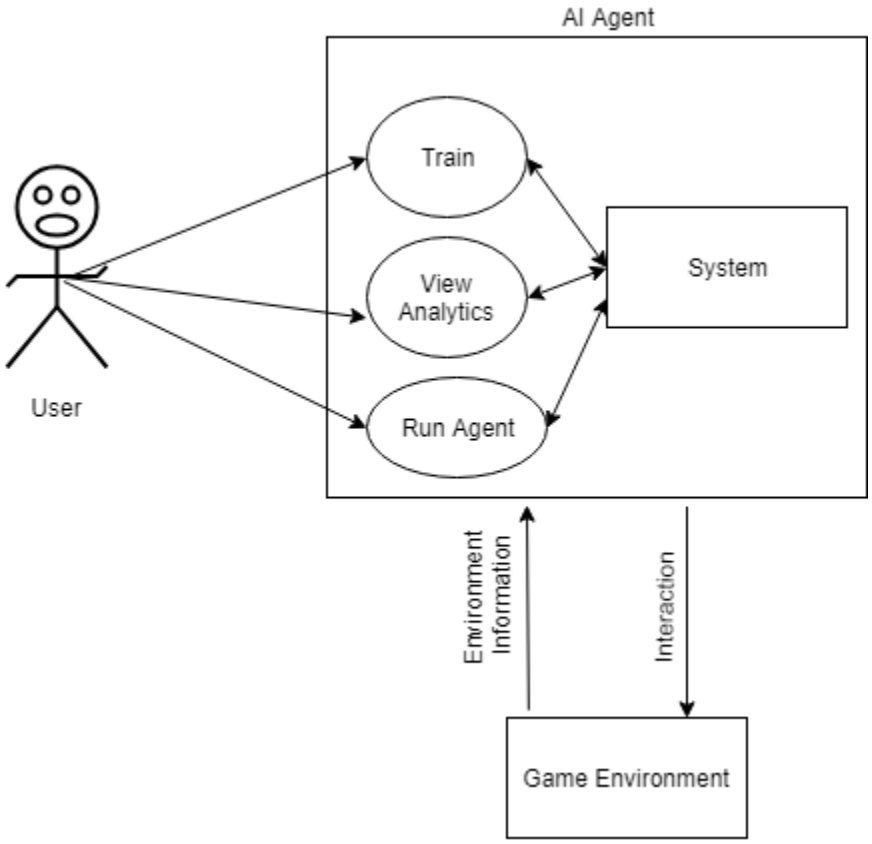

Figure 6 use case diagram

Chapter 4: Epilogue

4.1 Work Progress:

4.1.1 Progress Report:

Current progress is show as a reward graph and loss graph. For live testing our agent with trained model can be found at: *git@github.com:lurayy/AI_playing_flappy_bird.git*.

4.1.2 Reward Graph:

The graph below shows the reward distribution across the first 300 game played during the training session:

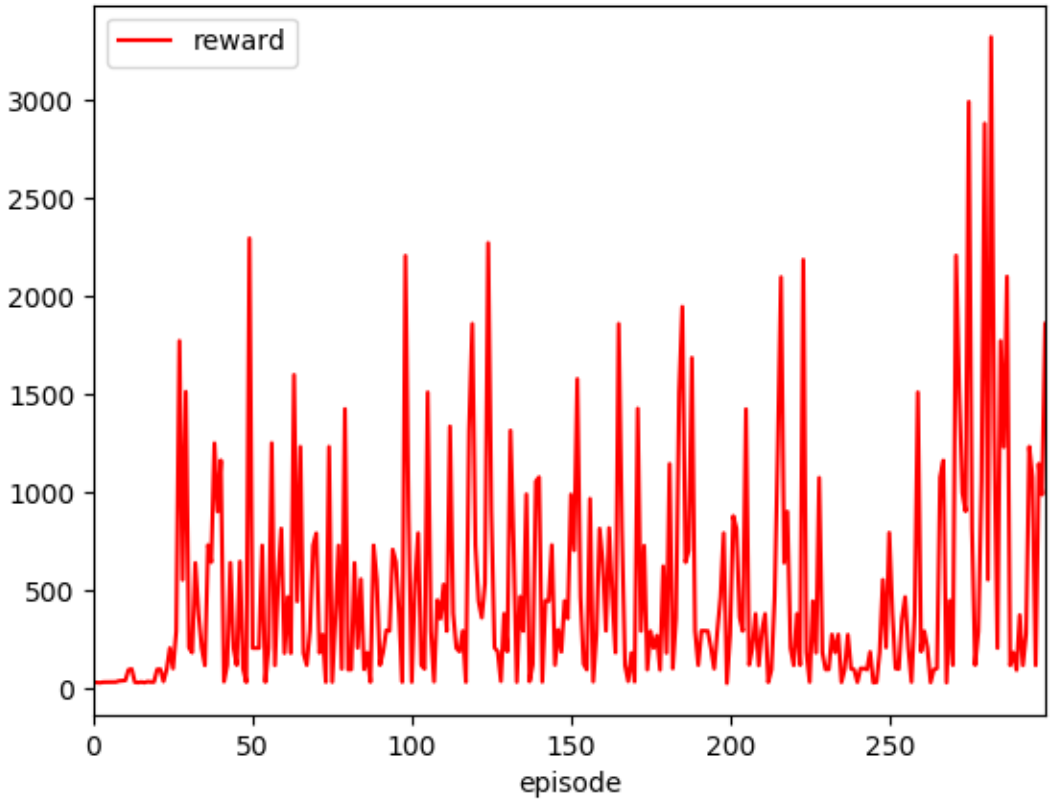

Figure 7 Reward Graph Of first 300 games played

4.1.3 Loss Graph:

It shows the accuracy of the implemented deep neural network to predict the optimal q value of the given states. Loss is calculated using mean squared error between predicted Q value and the optimal Q value. Since future reward have more significance on the current calculated optimal q value of the state action pair (as per the Bellman's Equation) the change between the model prediction of Q value and the optimal calculated Q value (which includes the future reward) is high much higher later in the training session. After being trained up to certain episodes the model can predicts the future rewards more accurately hence affecting the optimal calculated Q much more.

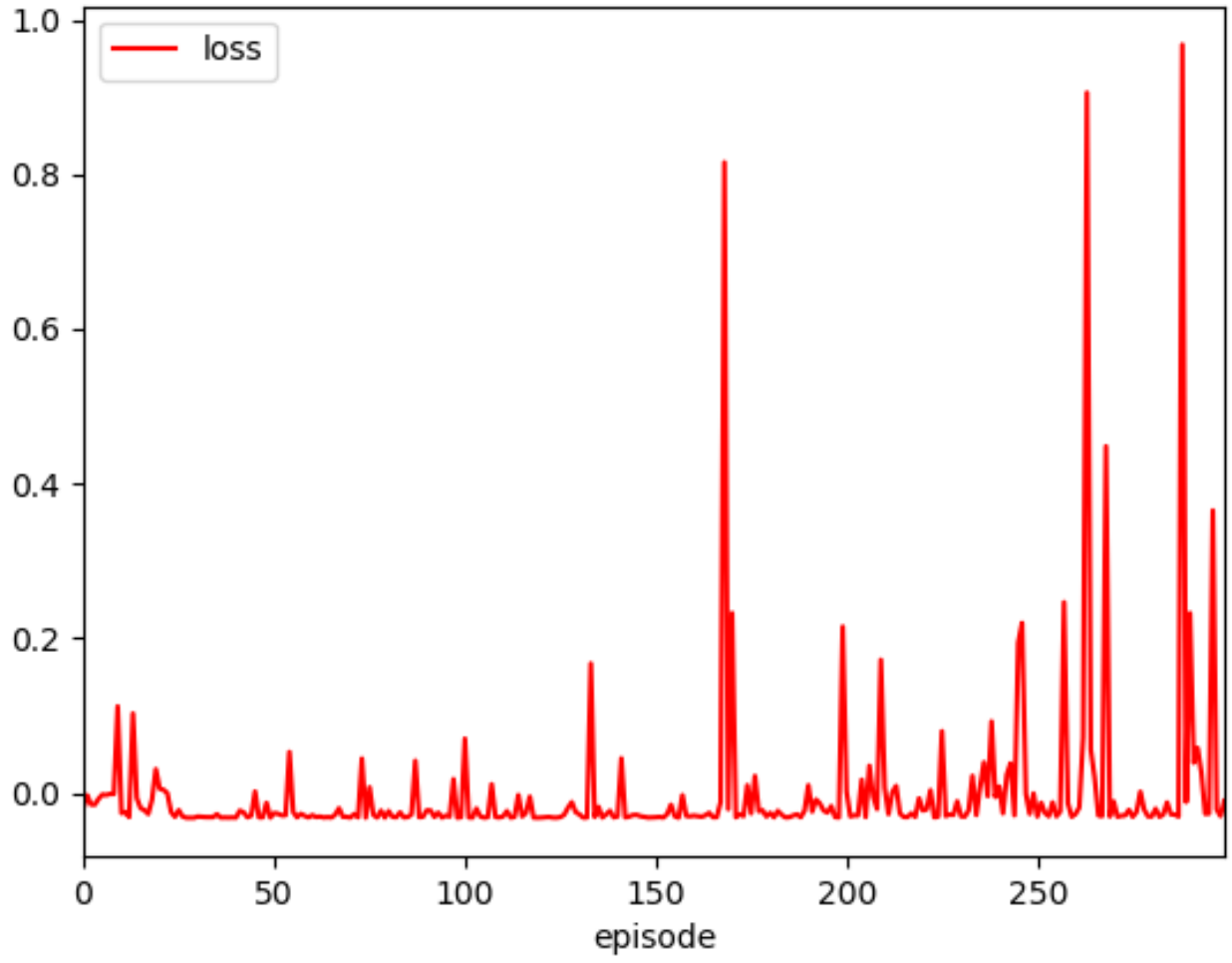

Figure 8 Loss Graph for first 300 games played

4.2 Task Completed:

The following tasks has been completed:

- ✓ Research.
- ✓ Design the environment and the agent's core algorithm.
- ✓ Designing and coding the deep neural network.
- ✓ Coding the game (environment) Flappy Bird in python using Pygame.
- ✓ Training, testing and adjusting the hyperparameters of our algorithm
- ✓ Data collection for statistical analysis and data visualization.

4.3 Task Remaining:

The following tasks remains to be completed:

- ➢ Code clean up and optimization
- ➢ Training the agent for longer episodes
- ➢ Final Documentation of the project
- ➢ Agent progress during training visualization
- ➢ Data visualization

References

[1] Sutton, Richard S.; Barto, Andrew G. (November 13, 2018). Reinforcement Learning: An Introduction (PDF) (Second ed.). A Bradford Book. [Accessed 9th Mar.2019]

[2]"Q-learning", *En.wikipedia.org*, 2019. [Online].

Available: https://en.wikipedia.org/wiki/Q-learning. [Accessed 9th Mar.2019]

[3]"Reinforcement learning", *En.wikipedia.org*, 2019. [Online].

Available: https://en.wikipedia.org/wiki/Reinforcement_learning.[Accessed 10th Mar.2019]

[4]"DeepMind Atari AI", *Arxiv.org*, 2019. [Online].

Available: https://arxiv.org/pdf/1312.5602v1.pdf. [Accessed: 10- Mar- 2019].

[5] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv.org*, 2019. [Online].

Available: https://arxiv.org/abs/1412.6980v8. [Accessed: 11- Jun- 2019].

[6]"sourabhv/FlapPyBird", *GitHub*, 2019. [Online].

Available: https://github.com/sourabhv/FlapPyBird. [Accessed: 11- Jun- 2019].

[7]"Playing Atari with Deep Reinforcement Learning | DeepMind", *DeepMind*, 2019. [Online].

Available: https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/. [Accessed: 10- Mar- 2019].

[8] R. Williams, "What is Flappy Bird? The game taking the App Store by storm", *Telegraph.co.uk*, 2019. [Online]. Available: https://www.telegraph.co.uk/technology/news/10604366/What-is-Flappy-Bird-The-game-taking-the-App-Store-by-storm.html. [Accessed: 11- Jun- 2019].

[9]"Everything you need to know about Adam Optimizer", *Medium*, 2019. [Online].

Available: https://medium.com/@nishantnikhil/adam-optimizer-notes-ddac4fd7218. [Accessed: 11- Jun- 2019].