Game Playing - A Reinforcement Learning Approach

Submitted in partial fulfillment of requirements

For the degree of

Bachelor in Computer Technology

by

Rohit Surana - 1311126

Jainam Sheth - 1311118

Abhishek Patil - 1311104

Under the Guidance of

Prof. Zaheed Shaikh

DEPARTMENT OF COMPUTER ENGINEERING

K.J.Somaiya College of Engineering, Mumbai-400077

(Autonomous College Affiliated to University of Mumbai)

Batch 2016-2017

K.J.Somaiya College of Engineering, Mumbai-400077

(Autonomous College Affiliated to University of Mumbai)

Certificate of Approval of Examiners

We certify that this project report entitled “GAME PLAYING - A REINFORCEMENT LEARNING APPROACH” is bonafide record of project work done by

1. Rohit Surana
2. Jainam Sheth
3. Abhishek Patil

This project is approved for the award of Bachelors Degree in Computer Engineering of University of Mumbai.

External Examiner   Internal Examiner

Date :

Place : Mumbai 77

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

1. Rohit Surana - 1311126 ________________
2. Jainam Sheth - 1311118 ________________
3. Abhishek Patil - 1311104 ________________

K.J.Somaiya College of Engineering, Mumbai-400077

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that the project entitled GAME PLAYING - A REINFORCEMENT LEARNING APPROACH is bonafide record worked upon by:

1. Rohit Surana
2. Jainam Sheth
3. Abhishek Patil

in the year 2016-17 under the guidance of Prof. Zaheed Shaikh of Department of Computer Engineering in partial fulfillment of requirement for the Bachelors Degree in Computer Engineering of University of Mumbai.

Guide/Co-guide Head of Department

Principal

Date :

Place: Mumbai 77

Abstract

Making computers to play games is a sophisticated learning problem. The need for this problem arises because it enables us to model real world systems and apply techniques/analysis on it before applying it to original real systems and games form a wonderful tool for trying out new approaches. Games have very large state space and there are many decisions that can be made in each state. Thus selecting the optimal solution is a complex decision making problem. In this project, we will be developing a system which will learn to play games without being explicitly programmed to do so using Reinforcement learning algorithm. We will also consider learning techniques such as the use of replay memory with Q-learning. The system will not be provided with any game specific information or hand-designed visual features [2]. It will learn from nothing but the screen images, the reward and terminal signals, and the set of possible actions - just as an amateur human player would learn.

Acknowledgement

We would like to express our sincere gratitude to Professor Zaheed Shaikh for guiding us each step of the way of this project. He also helped us in preparing for a professional environment, we all realised the power of team work and schedule. Secondly we would like to thank the department of Computer Engineering for their resources and training that made the implementation smoother. A special mention to all the lab assistants of the department as well for their efforts and cooperation.

It was a great learning experience and lastly we thank the institute, KJ Somaiya College of Engineering, for giving us this opportunity to explore and expand our horizons through this project and all our academic endeavours.

Contents

List of Figures

Chapter 1

Introduction

This is the chapter which gives an overview of the project by providing information about the existing procedures, motivations for the implementation of the new system and the scope of the project.

1.1 Introduction/Motivation

Our project involves implementing one of the current techniques in reinforcement learning which is Q-learning to let an AI agent learn to play some simple games. This project has far reaching real world consequences because video games often reflect reduced versions of real-world problems. Game playing is a very sophisticated task because it involves building a model that is able to play many games with same hyperparameters and model is a task that incorporates Artificial Intelligence. Reinforcement Learning is a field of machine learning where a software agent learns to interact with its environment by discerning the outputs of these interactions and keeping a goal of obtaining maximum possible reward to complete the given task. Q-learning is one of the former reinforcement algorithms particularly designated to maximize reward in a multistage environment. It's a method of asynchronous dynamic programming which provides agents the potential to act optimally by encountering the results of the actions. We are using Python to program our agent, Keras library to create Artificial Neural Networks and

OpenAi Gym[6] toolkit for extracting the game environment. OpenAI Gym is a toolkit that is used for developing and comparing different reinforcement learning algorithms. Q-Learning is reinforcement learning algorithm which is model independent which will be very useful for our project since it involves learning different games. The input to the agent are the ram contents or the state space of the game environment and current reward. The agent tries to learn to play the game by first taking random actions and evaluating it using q-learning and neural network, thereby improving the policy it follows. We have tested the code for the cartpole game and also for some other games.

1.1.1 Existing Systems

Attempts for making computers play games through machine learning approach has been one of the most trending subject in recent years, mainly because of solving problems which seem very difficult without the use of learning algorithms. The first successful attempt to make computer learn was a TD-Gammon program that used reinforcement learning to make a self-learning backgammon agent successfully learned to play at an expert level[1]. They have used temporal difference learning, a model-free reinforcement learning algorithm along with function approximation. Most recently, the Deep-Q networks (DQN) of Minh et al [3][2] have achieved state of the art performance in many Atari 2600 games implemented in the Arcade Learning Environment (ALE) [4]. They have used a variant of Q-learning algorithm along with deep neural networks and function approximators on the Atari games. Their approach uses sampling from replay memory to avoid instability of nonlinear models. Their agent learned to play as an expert in many games with just the game video, reward and set of actions as input. Other approach uses OpenCV to extract the game features instead of deep neural networks[5].

1.2 Problem Definition

The problem we have considered is to create a general algorithm which will learn to play different games without being explicitly programmed about the specific game. We use reinforcement learning algorithms to train the system like Q-learning. The input provided to the system is screen images represented in pixels, score or the ram of the environment and output will be an action (left, right or fire) which in turn will result in a reward or a punishment. This input will be processed by Convolutional Neural Network with two hidden layers and then fed to the learning algorithm. With every right move to maximize the score, the system gets a reward. The objective of learning algorithm is to select available actions for making an action so that the reward will be maximized and the agent learns the correct way of playing the game. With some more techniques such as using replay memory, exploration exploitation the result is made considerably better. Finally after some proper training the system will learn the game and play like an expert.

1.3 Scope of the thesis

To make a computer learn how to play games without having any domain specific knowledge about the game is similar to how we humans think - by experience. Some Reinforcement learning algorithms will be used which are an important model of how we learn. Praise from our parents, grades in school, salary at work – these are all examples of rewards. Credit assignment problems (selecting the right action to do in some particular case) and exploration-exploitation (exploring new ways to come up with the right solution), optimization problems come up every day both in business and in relationships. That's why it is important to study this problem and games form a wonderful tool for trying out new approaches. So our objective is to develop a system which can learn the game by itself and play the game better than a human and also to have a general algorithm which can learn various games such as cart pole or Atari 2600 games such as breakout, pong with similar goals.

Chapter 2

Literature Survey

Literature Survey specifies all the knowledge about the techniques that can be used for object recognition and feature extraction required for the implementation of the project.

1. **Reference Papers:**

   (a) **Human-level control through deep reinforcement learning:** Teaching computers to play video games is a complex learning problem that has recently seen increased attention. In this paper, we develop a system that, using constant model and hyperparameter settings, learns to play a variety of Atari games. In order to accomplish this task, we extract object features from the game screen, and provide these features as input into reinforcement learning algorithms. We detail our testing of different hyperparameter settings for two reinforcement learning algorithms, using both linear and neural network function approximators, in order to compare the performance of these approaches. We also consider learning techniques such as the use of replay memory with Q-learning, finding that even on simple MDPs this method significantly improves performance. Finally, we evaluate our model, selected through validation on the game Breakout, on a test set of different video games.

(b) **Playing Atari with Deep Reinforcement Learning:**

We present the first deep learning model to successfully learn control policies di- rectly from high-dimensional sensory input using reinforcement learning. The

model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future

rewards. We apply our method to seven Atari 2600 games from the Arcade Learn- ing Environment, with no adjustment of the architecture or learning algorithm. We

find that it outperforms all previous approaches on six of the games and surpasses

a human expert on three of them.

2. **Learning to play atari :** This approach uses Arcade Learning Environment (ALE) which is a simple object-oriented framework that allows programmers to develop AI agents for Atari 2600 games[3]. The game screen is extracted using the ALE. OpenCV is then used to detect the edges and draw contours around distinct closed edges in the extracted game screen. This information is given to a clustering algorithm such as DBSCAN to classify the features on a given screen. After that object tracking is done. For this a simple greedy algorithm is used that matches each entity on the previous screen with the closest entity sharing the same label on the current screen. All this will give a state-action pair of data which is given to the Markov Decision Process (MDP) which will make the decision about

which should be the next action based on the present state and action with the goal of getting maximum reward. A reward in our case will be in form of the score of the game. This decision can be improved by using Q-learning or SARSA which are explained as follows:

3. **Q-learning:** Q-learning algorithm gives the optimal solution for the next action to be taken by considering the future rewards. Q-learning is an off policy approach where it updates its Q-values using the Q-value of the next state s and the greedy action a. In other words, it estimates the reward for state-action pairs assuming a greedy policy. In Q-learning we define a function Q(s, a representing the maximum discounted future reward when we perform action a in state s, and continue optimally from that point on.

4. **SARSA:** State-Action- Reward-State- Action (SARSA) is an algorithm for learning a Markov decision process policy, used in the reinforcement learning area of machine learning. SARSA is an on policy approach where it updates its Q-values using the Q-value of the next state s and the current policies action a. It estimates the return for state-action pairs assuming the current policy continues to be followed. Another approach[1][2] uses a convolutional neural network which takes raw pixels as input data and whose output is a value function estimating the future rewards. This network is then trained with the Q-learning algorithm to get the optimal solution.

Chapter 3

Project Management Plan

The Software Project Management Plan for this project defines the project management goals for the project and includes a description of the deliverables and deadlines.

3.1 Feasibility Analysis

Feasibility analysis is the analysis and evaluation of a proposed system to determine whether the system is feasible technically, within the estimated costs and that the system is profitable. For a newly proposed system, a feasibility study must be approved for the development. The following aspects are considered while determining the feasibility analysis.

3.1.1 Operational feasibility

The system is flexible as it can add more than one game for learning without changing the code. Also, it has a user friendly interface. therefore, the system is Operationally feasible.

3.1.2 Cultural feasibility

The system could be accepted at organizational as well as personal level very well.

3.1.3 Technical feasibility

Basic knowledge of python language along with Artificial Neural Network library such as Keras, UBUNTU operating system and good RAM is required for the developers. It is a technically flexible system which can work on different platforms given the requirements are fulfilled. Hence the system is technically feasible.

3.1.4 Schedule feasibility

The project could be completed within the given deadline. The major task is to develop a general algorithm that can play more than one games which will take more time. Since the algorithm is developed, the system is time feasible.

3.1.5 Economic feasibility

The software developed to implement the project uses python, libraries like Keras and OpenAi Gym Environment. All of these are freely available and accessible and therefore, the system is economically feasible.

3.1.6 Reliability

The project provides its functionality as long as it is installed on the device. It requires no or minimum maintenance.

3.1.7 Portability

The code can run on any system having Ubuntu operating system given that the keras library is already installed.

3.2 Lifecycle Model

The most suitable process model for the development of our project will be Agile Methodology which gives the liberty to move quickly and easily between various phases of development without any restrictions. Moreover, the workflow is bidirectional, i.e. any changes can be roll backed and modifications to previous versions can be applied as and when required. This is very crucial while creating algorithms and coding as updates and corrections are constantly ongoing activities for any form of application development. The following is the diagram for our model.

3.3 Project Cost and time estimation

The project is based on pure coding in python programming language using machine learning algorithms like Q-learning. The project uses Artificial Neural Network libraries i.e. Keras library which is freely available. Therefore there is no such major project cost included. Small charges are spent on internet usage for gathering information and documentation purpose. The project was completed on 5 April,2017.

3.4 Resource plan

As we are using a reinforcement learning approach, the resources required for implementing the model include keras library which is used to train the neural network, OpenAi gym which provides the game environment and python 2.7.2 for programming the model. There are certain activities involved that include writing and submitting the proposal, Conduct Literature Survey, Hardware Setup, Software Requirement Specification document, Software Project Management Plan, Software Design Description, Training the network, Testing the network, Modification, final testing.

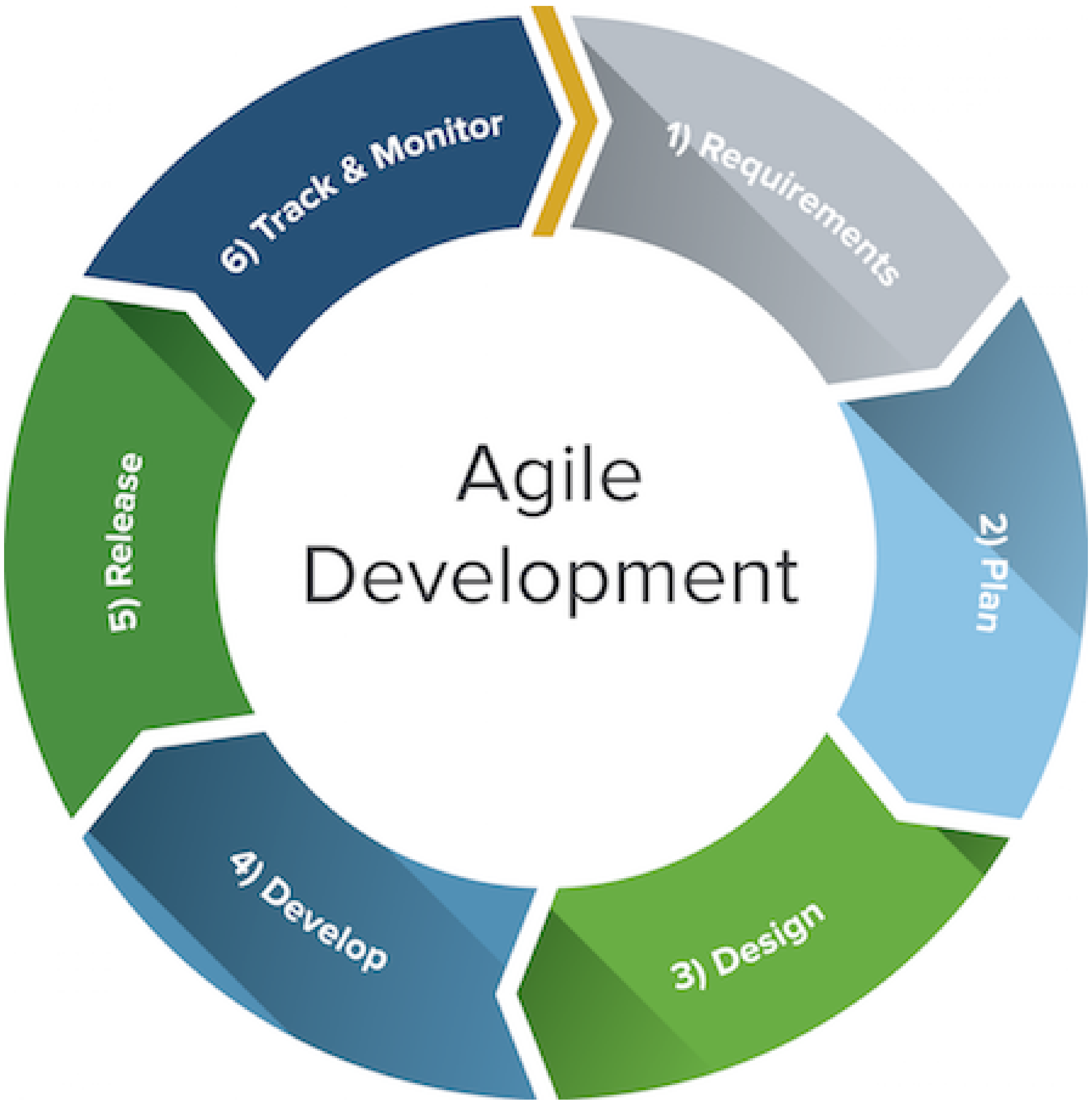
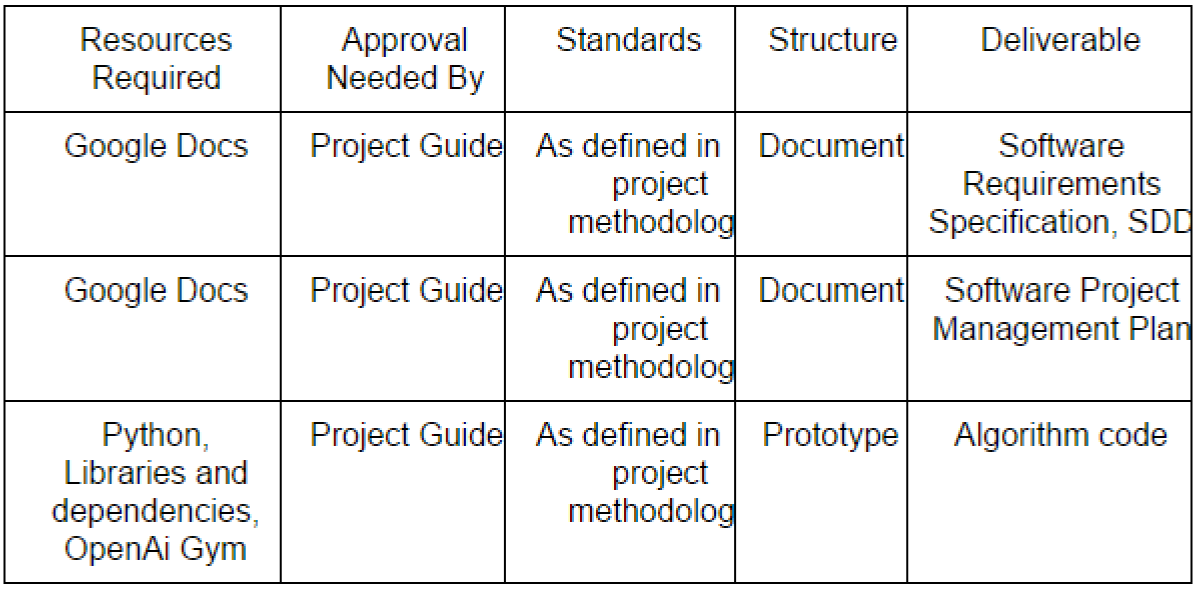


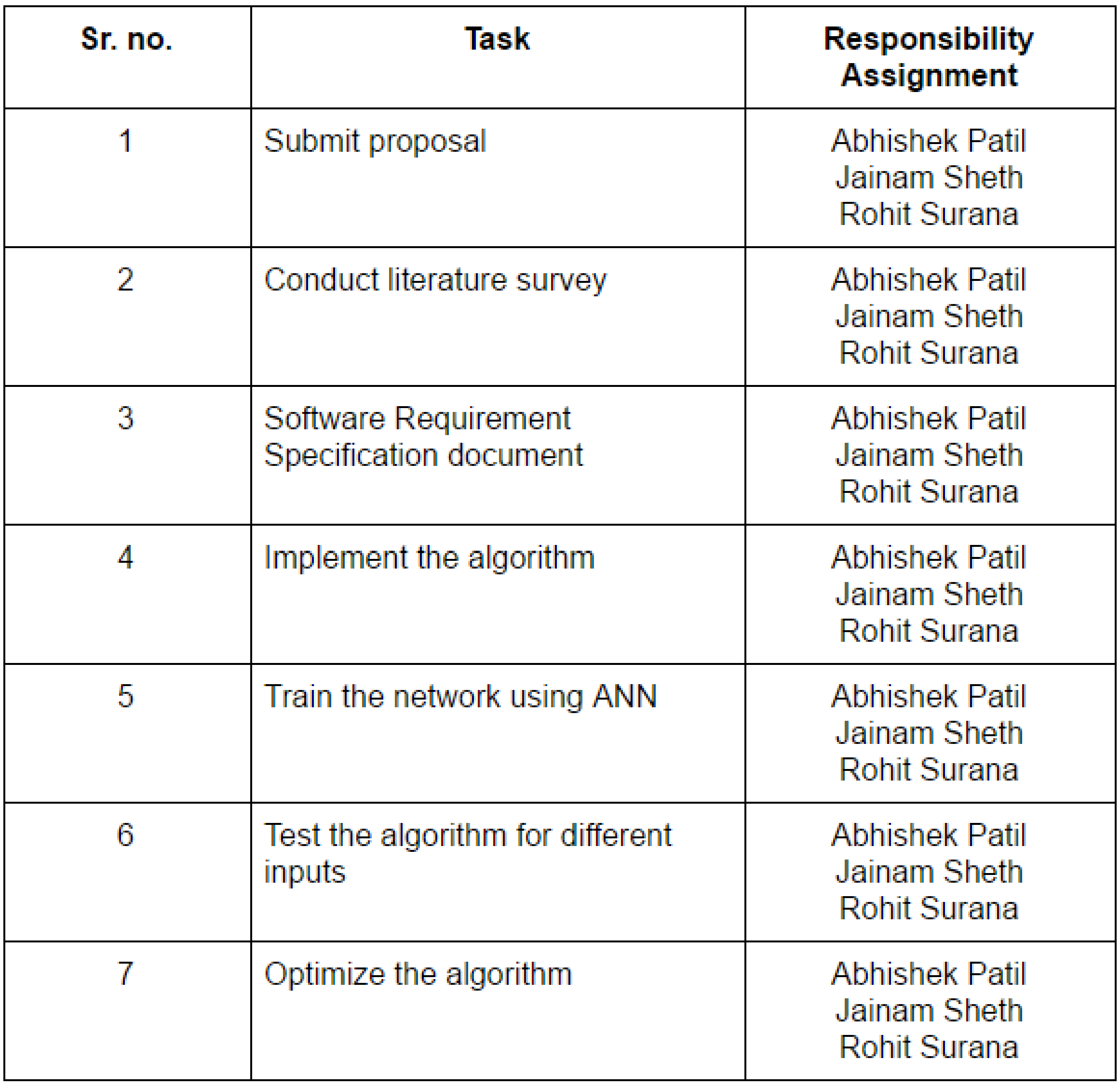
Figure 3.1: Agile methodology



Resources Required	Approval Needed By	Standards	Structure	Deliverable
Google Docs	Project Guide	As defined in project methodolog	Document	Software Requirements Specification, SDD
Google Docs	Project Guide	As defined in project methodolog	Document	Software Project Management Plan
Python, Libraries and dependencies, OpenAi Gym	Project Guide	As defined in project methodolog	Prototype	Algorithm code

Figure 3.2: Task and Responsibility

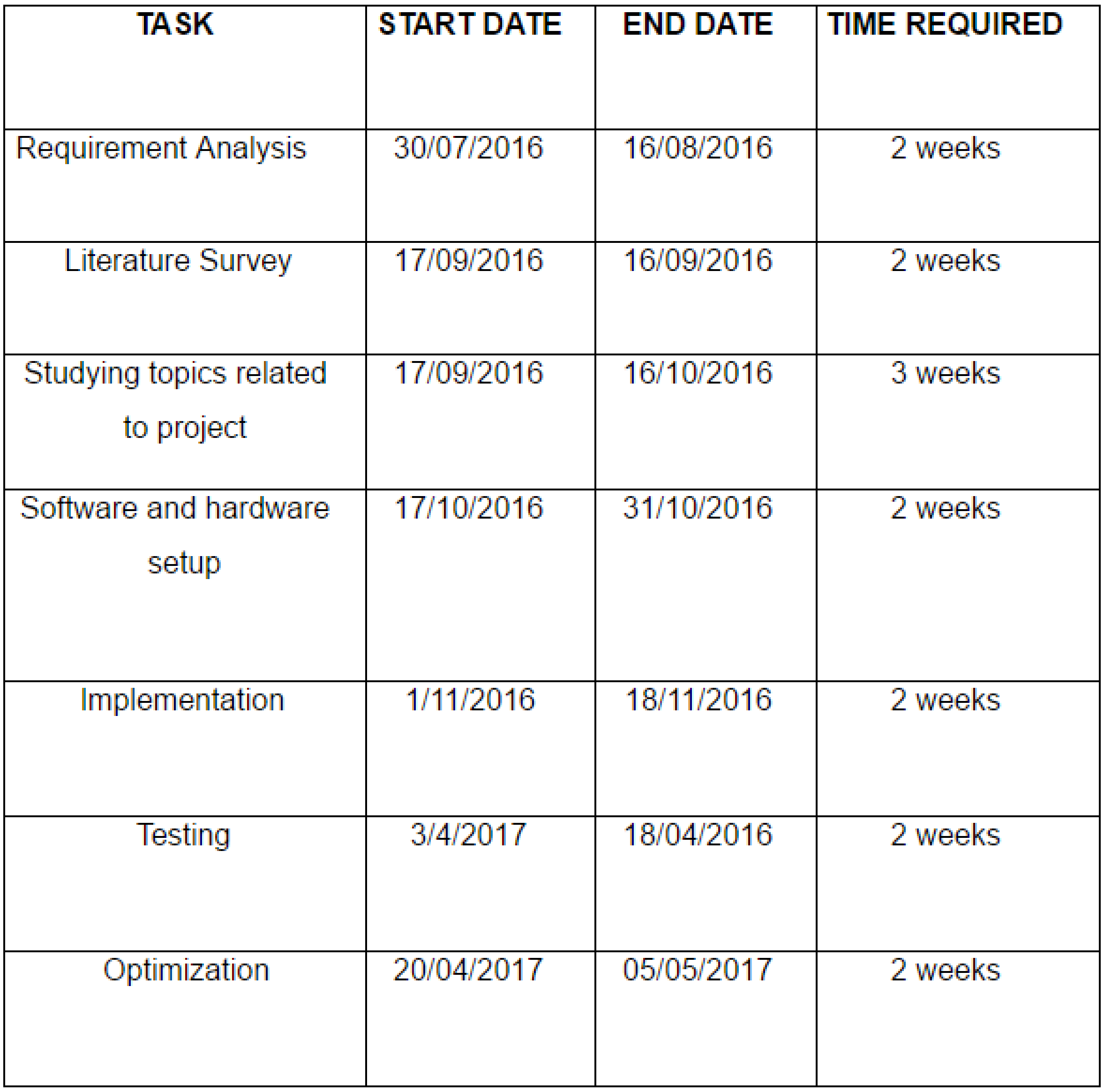
3.5 Task Responsibility Assignment Matrix



Sr. no.	Task	Responsibility Assignment
1	Submit proposal	Abhishek Patil Jainam Sheth Rohit Surana
2	Conduct literature survey	Abhishek Patil Jainam Sheth Rohit Surana
3	Software Requirement Specification document	Abhishek Patil Jainam Sheth Rohit Surana
4	Implement the algorithm	Abhishek Patil Jainam Sheth Rohit Surana
5	Train the network using ANN	Abhishek Patil Jainam Sheth Rohit Surana
6	Test the algorithm for different inputs	Abhishek Patil Jainam Sheth Rohit Surana
7	Optimize the algorithm	Abhishek Patil Jainam Sheth Rohit Surana

Figure 3.3: Task and Responsibility Assignment Matrix

3.6 Project Timeline Chart



TASK	START DATE	END DATE	TIME REQUIRED
Requirement Analysis	30/07/2016	16/08/2016	2 weeks
Literature Survey	17/09/2016	16/09/2016	2 weeks
Studying topics related to project	17/09/2016	16/10/2016	3 weeks
Software and hardware setup	17/10/2016	31/10/2016	2 weeks
Implementation	1/11/2016	18/11/2016	2 weeks
Testing	3/4/2017	18/04/2016	2 weeks
Optimization	20/04/2017	05/05/2017	2 weeks

Figure 3.4: Project Timeline Chart

Chapter 4

Project Analysis and Design

This chapter gives details about the system design by presenting the architecture diagram, the use-case diagram, class diagram as well as the activity diagram of the proposed system. Brief explanation for each class has also been provided. It also contains information about the dictionaries used in the project.

4.1 Software Architecture diagram

Following is the architecture which best depicts the software system.

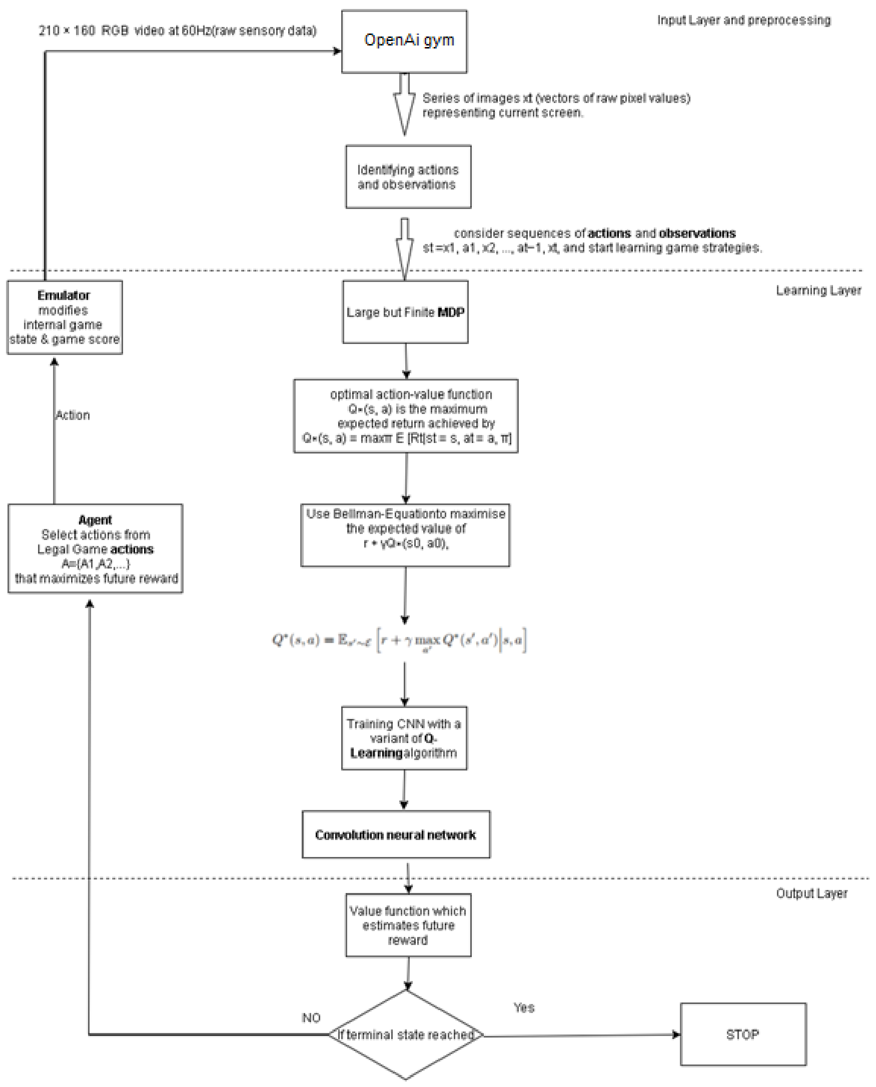

210 × 160 RGB video at 60Hz(raw sensory data)
OpenAi gym
Input Layer and preprocessing
Series of images xt (vectors of raw pixel values) representing current screen.
Identifying actions and observations
consider sequences of actions and observations
st =x1, a1, x2, ..., at−1, xt, and start learning game strategies.
Learning Layer
Emulator modifies internal game state & game score
Large but Finite MDP
optimal action-value function Q∗(s, a) is the maximum expected return achieved by Q∗(s, a) = maxπ E [Rt|st = s, at = a, π]
Action
Agent Select actions from Legal Game actions A={A1,A2,...} that maximizes future reward
Use Bellman-Equationto maximise the expected value of r + γQ∗(s0, a0),
Q*(s,a) = E_{s'∼ε} [r + γ max_{a'} Q*(s',a') | s,a]
Training CNN with a variant of Q-Learningalgorithm
Convolution neural network
Output Layer
Value function which estimates future reward
NO
If terminal state reached
Yes
STOP

Figure 4.1: Architecture

The layers are described as:

1. **Input layer** The input layer creates the game environment using openai gym as well as identifies the the current state of the system. Initially, it performs random actions and observes the result of these actions.

2. **Learning layer** This layer uses a reinforcement learning algorithm i.e. Q-learning algorithm and passes its output to the neural network. In such manner the network is trained for various inputs and the agent is set to take actions that will maximize

the reward.

4.2 Architectural style and justification

Architectural style LAYERED ARCHITECTURE**Justification** The architecture is a hierarchical system organization which acts as a Multilevel client-server where each layer acts as a Server- service provider to layers "above it" or a Client-service consumer of layer "below it". Each layer exposes an interface (API) to be used by above layers and the connectors are protocols of layer interaction. We can see that this layered architecture comprises of four different layer, each using the functionality of an upper layer. The data transfers takes place through these calls and communications between inter as well as intra layer communications. Thus, the layered architecture is the most justifiable architecture for this application.

4.3 Software Requirements Specification Document

4.3.1 Product Overview

The input provided to the system is screen images represented in pixels or the ram of the environment and output will be an action (left, right or fire) which in turn will result in a reward or a punishment. With every right move to maximize the score, the system gets a reward. The objective of learning algorithm is to select available actions for making an action so that the reward will be maximized and learn the correct way of playing the game. In all, after the agent gets completely wise, it can compete a human by achieving a greater score than human.

4.3.2 Specific Requirement

External Interface Requirements

1. **User Interfaces** Terminal window used to install the softwares and run the codes. A Text editor or any editor for implementing the code and modifying it. A window which displays the running game environment.

2. **Hardware Interfaces** No hardware interface is required.

3. **Software Interfaces** The operating system on which the product would work are:

   (a) OpenAI Gym Beta - A toolkit for developing and comparing reinforcem ent learning algorithms.

   (b) Python 2.7 for coding.

   (c) Arcade Learning Environment (ALE) - a simple object oriented framework that allows researchers and hobbyists to develop AI agents for Atari 2600 games.

   (d) Editor: Text editor or notepad.

   (e) Documentation: Microsoft word 2010.

   (f) Ubuntu

The product is a standalone product and does not require any other networking software

Functional Requirements

1. Input raw pixels and score - Extracting the score from ALE and giving the raw pixels as input to the Convolutional Neural Network (CNN).

2. Conversion to grayscale - The input raw pixels must be resized to 84*84 and then should be converted into grayscale. Hence the state space must be reduced.

3. Best move prediction - The learning algorithm will try to maximize the score by finding the next action which will be the one with the maximum reward and also exploring random states sometimes so that it does not end up at local minima.

Non-functional Requirements

1. Reliability - The computer may run for n number of days to get trained, so the processor must be sturdy enough to work in such an environment.

2. Performance - Since the agent's state space is going to be large, the learning algorithm should not take infinite time to reach the goal state.

3. Availability - The computer is recommended to have a NVIDIA graphics card and a fast processor to compute the large state space.

4.3.3 Software System Attributes

1. **Reliability** The agent developed is highly reliable, once trained over a sufficient period the agent becomes expert at game playing. After this point agent continues to play optimally and never fails and hence its highly reliable.

2. **Portability** Portability with respect to our project will be that the algorithm implemented should be able to learn different games without changing the code specific to that game. If using OpenAi Gym for the implementation then the project can only be implemented on Linux/Ubuntu or Mac.

3. **Performance** Static requirements - All the softwares or dependencies or libraries used must clear the minimum requirement for version. Dynamic requirements - The algorithm or the agent implemented should work on different game environment of similar type

4.4 Software Design Document

4.4.1 Data Flow Diagrams

1. **Context Level**

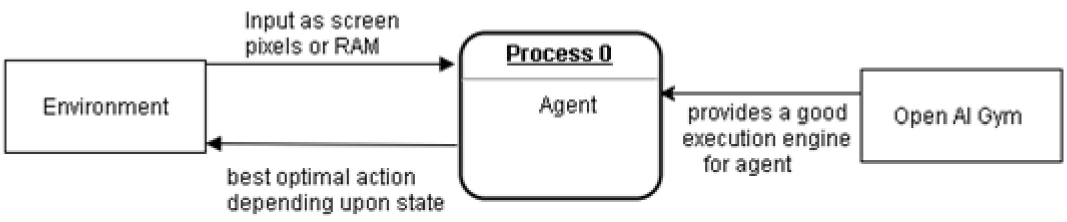

Input as screen pixels or RAM
Process 0
Agent
Environment
provides a good execution engine for agent
Open AI Gym
best optimal action depending upon state

Figure 4.2: Context Level

2. **DFD Level 1**

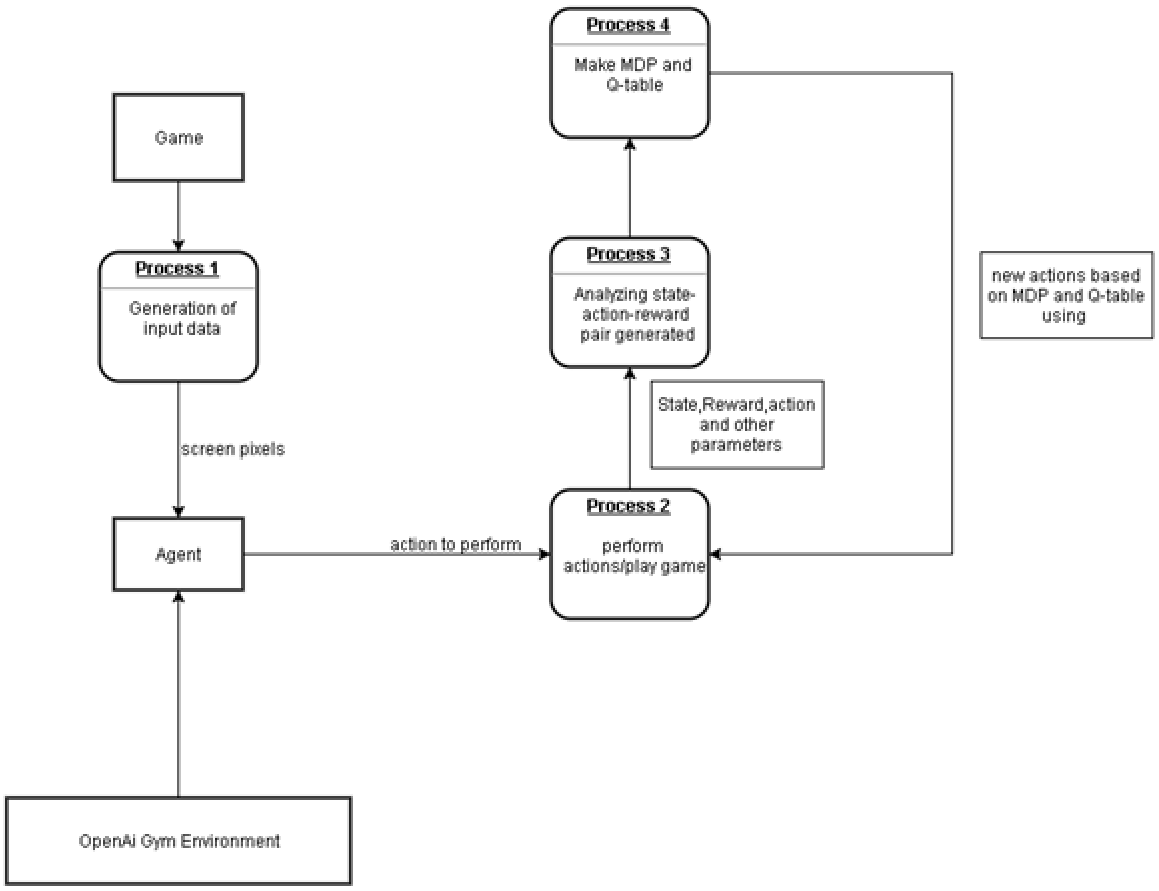

Process 4
Make MDP and Q-table
Game
Process 1
Generation of input data
Process 3
Analyzing state-action-reward pair generated
new actions based on MDP and Q-table using
State,Reward,action and other parameters
screen pixels
Agent
action to perform
Process 2
perform actions/play game
OpenAi Gym Environment

Figure 4.3: DFD Level 1

4.4.2 Use Case Diagram

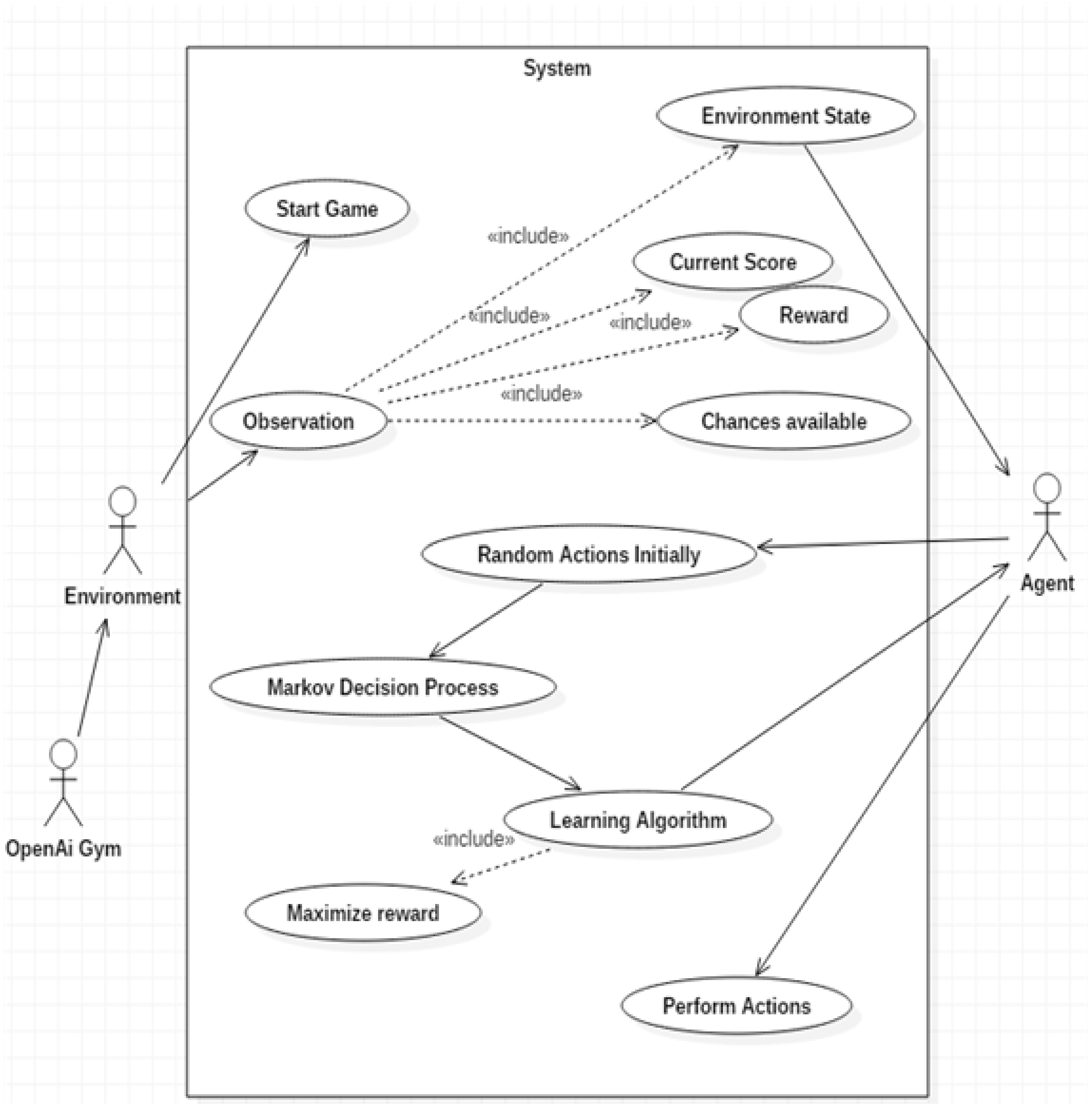


Figure 4.4: Use Case

4.4.3 Class Diagram

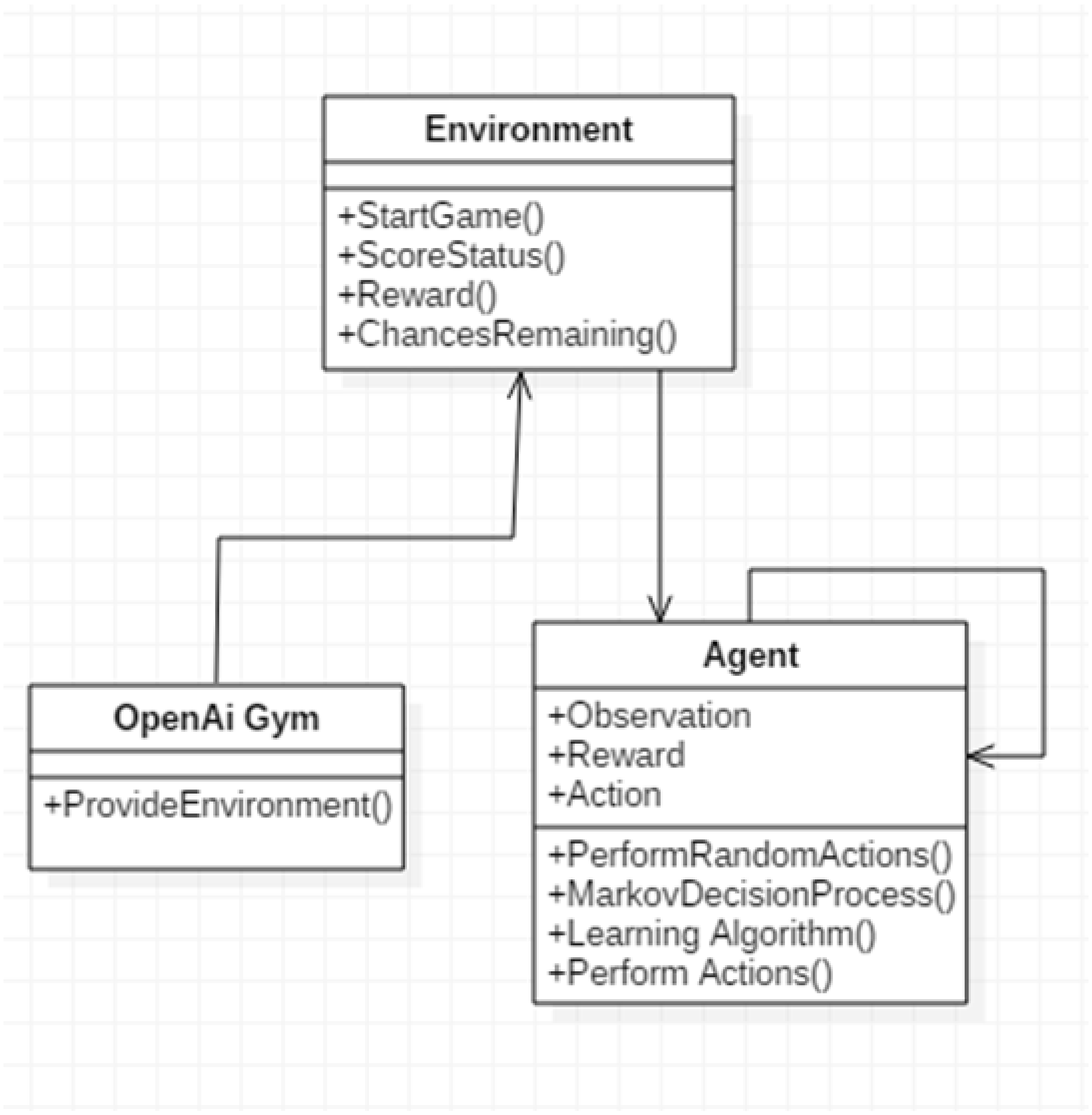


Figure 4.5: Class Diagram

4.4.4 Entity-Relationship Diagram

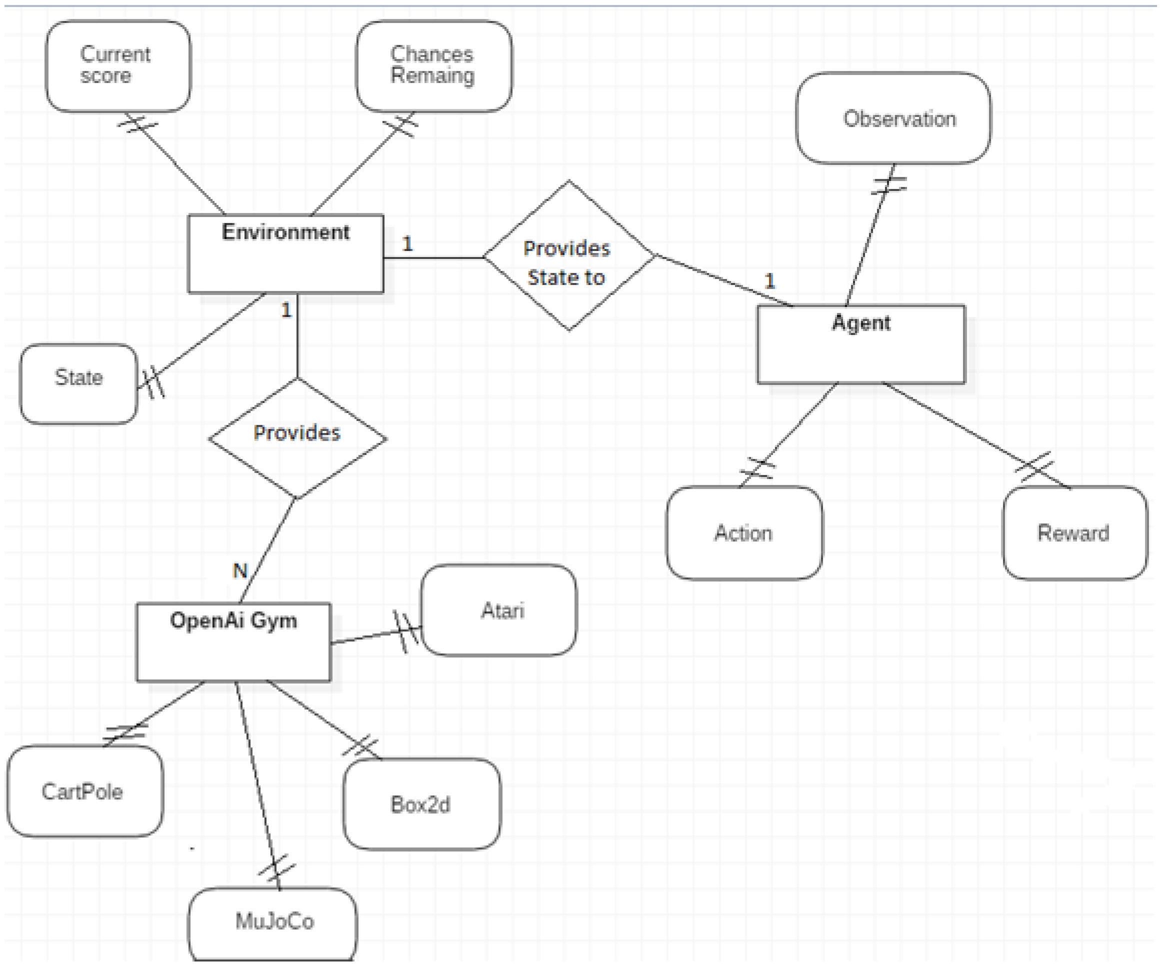


Figure 4.6: ER Diagram

Chapter 5

Project Implementation

This document defines three different implementation algorithms that has been used for the system. All these techniques produce an interpreted output.

5.1 Approach

We have implemented an agent in python which learns to play simple video games on its own. Traditional game playing algorithms were nothing but hard-coded programs i.e. they were explicitly programmed. Our agent first starts playing randomly and gradually builds its experience and stores the state, action, reward pair in the replay memory. It has the following functions action, observe and replay .We then train a sequential linear neural network in keras using replay memory and use Q-learning algorithm to predict the optimal action to be taken in a particular state.

5.2 Programming Languages Used For Implementation

We used python since it has excellent libraries for neural networks, agent and environment building. Python is a general purpose and interpreted programming language.

It's design philosophy emphasizes code readability. The syntax of python is simplest among all the programming languages.

5.3 Tools Used

We have used eclipse as the editor for writing programs. Python compiler for compiling python scripts. Google docs for creating documents and reports. IBM rational rose for gantt chart creation. Testing tools for performing unit testing on the algorithm. Open AI Gym provided us with an excellent environment for building the game agent. Various game environments like cartpole, mountain car etc were available thus we didn't had to worry about model generation.

5.4 Deployment Diagram

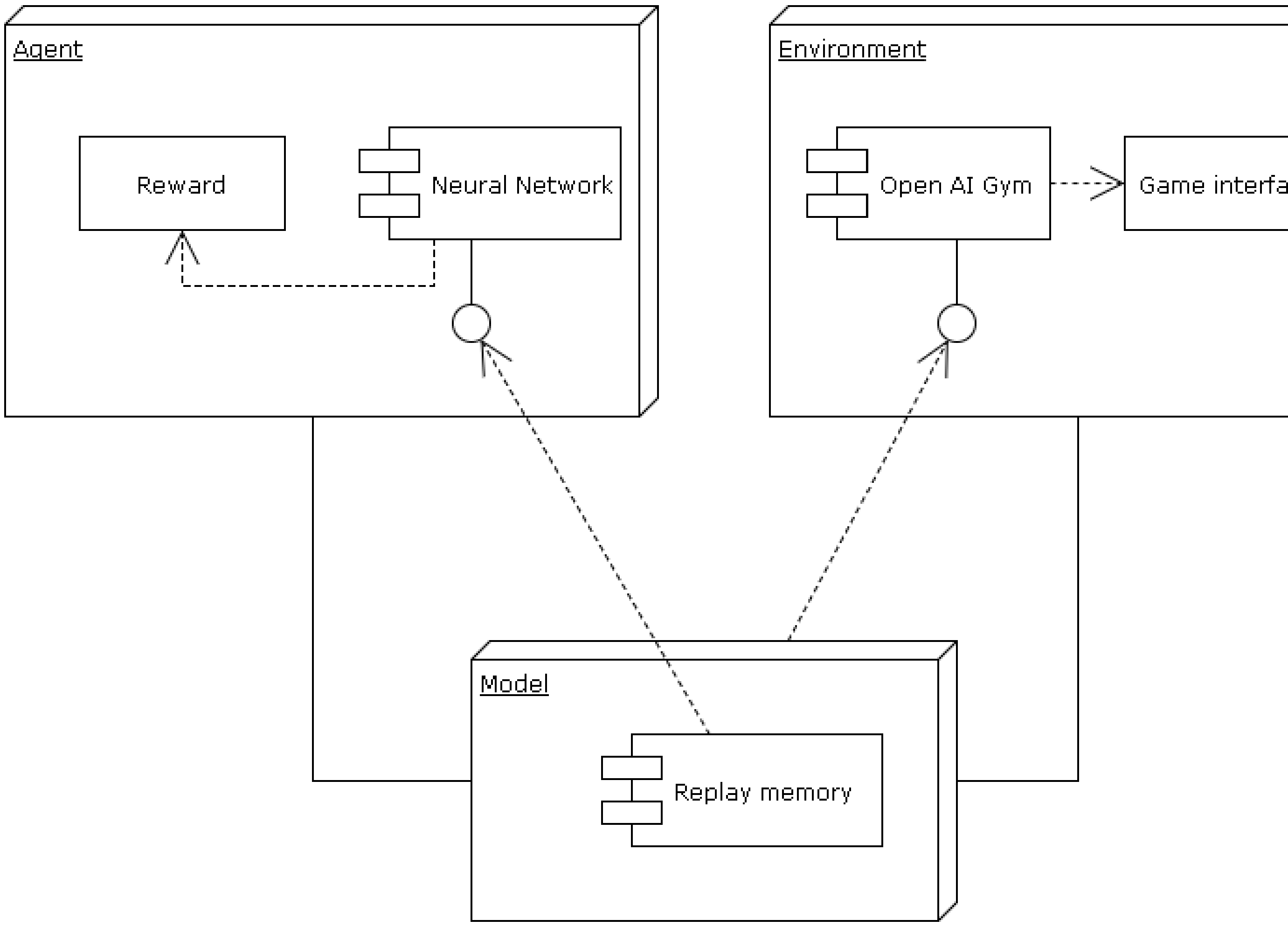


Figure 5.1: Deployment Diagram

Chapter 6

System Test Document

This document defines the testing strategy that will be used for the proposal system. Objective of this document is mainly to communicate project–wide quality standards and procedures. It involves the documentation of modules that should be developed and are developed before or during the testing of Software and the outcome of those tests.

6.1 Testing Approach

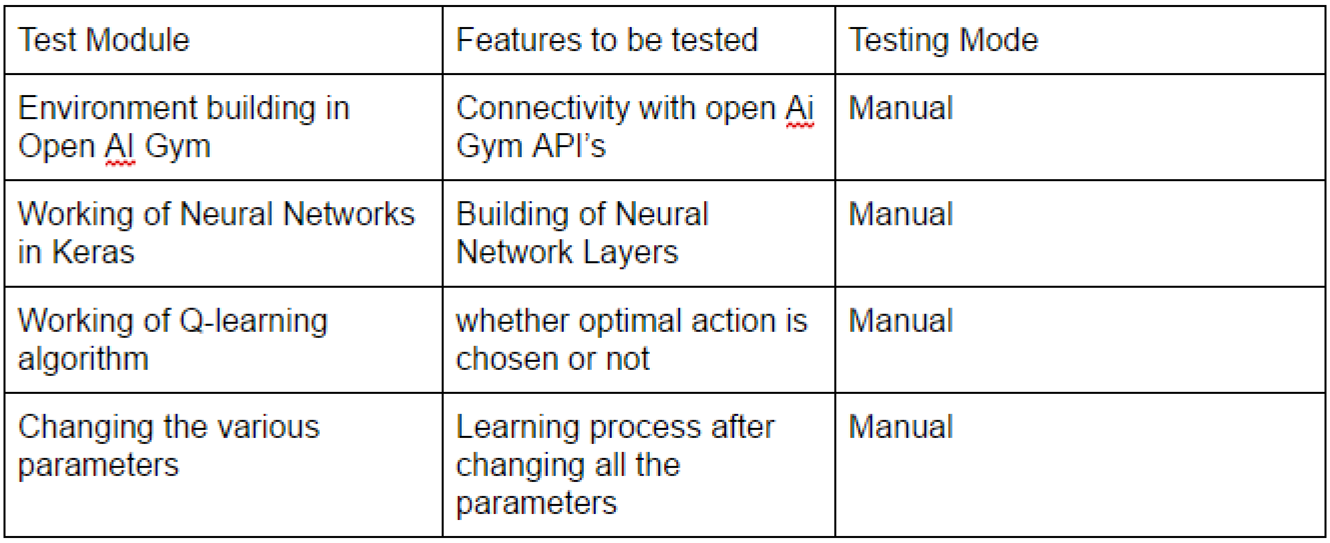
The testing approach will follow agile testing methodology. The testing practice follows the principles of the agile, emphasizing testing from the perspective of customers who will utilize the system. The testing does not emphasize on rigidly defined testing procedures, but will focus on testing iteratively against newly developed code. The activities of testing are as follows:

1. A Test Plan is developed mentioning the modules and features to be tested.
2. Unit Tests are performed as mentioned in the Test Plan. Unit Test cases consists of the module to be tested, the test case, given input, expected output and actual output.

3. Integrated System Testing will be performed to test the system as a whole. It will help in checking whether the system works properly by integrating one module at a time. Integrated System Test cases will consists of the property of system to be tested, the test case, given input, expected output and actual output.

6.2 Test Plan

The project team will perform the testing according to the test plan specified as follows:

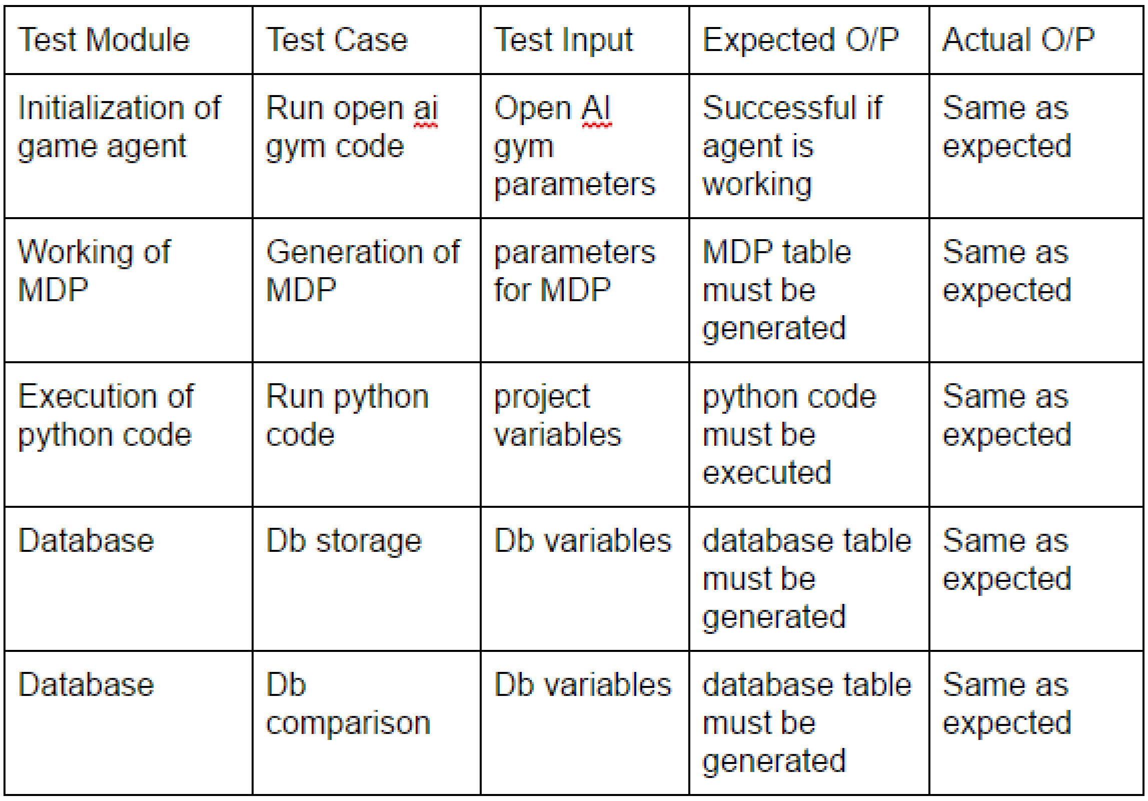


Test Module	Features to be tested	Testing Mode
Environment building in Open AI Gym	Connectivity with open Ai Gym API's	Manual
Working of Neural Networks in Keras	Building of Neural Network Layers	Manual
Working of Q-learning algorithm	whether optimal action is chosen or not	Manual
Changing the various parameters	Learning process after changing all the parameters	Manual

Figure 6.1: Test Plan

6.3 Unit test cases

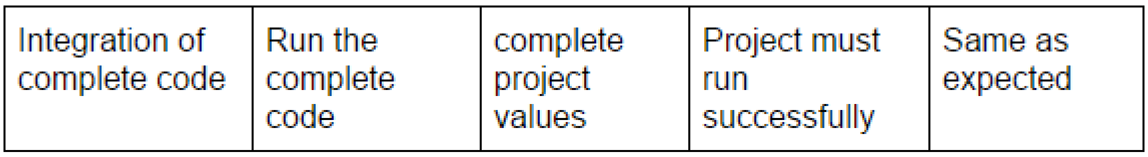
In computer programming, unit testing is a method by which individual units of source code are tested, to determine if they are fit for the use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual or procedure. In object oriented programming a unit is usually a method. Unit test are created by programmers or occasionally by white box testers during the development process.



Test Module	Test Case	Test Input	Expected O/P	Actual O/P
Initialization of game agent	Run open ai gym code	Open AI gym parameters	Successful if agent is working	Same as expected
Working of MDP	Generation of MDP	parameters for MDP	MDP table must be generated	Same as expected
Execution of python code	Run python code	project variables	python code must be executed	Same as expected
Database	Db storage	Db variables	database table must be generated	Same as expected
Database	Db comparison	Db variables	database table must be generated	Same as expected

Figure 6.2: Unit test cases

6.4 Integrated System Test Cases



Integration of complete code	Run the complete code	complete project values	Project must run successfully	Same as expected

Figure 6.3: Integrated system test cases

Chapter 7

Conclusion and Future work.

This document defines the over all progress of the project that has been implemented in this semester successfully.

In this paper, we have implemented an algorithm to learn video games and tested on different games. Our algorithm performs excellently well on Cartpole game by achieving a maximum score as specified in OpenAi Gym. We have also tested the impact on learning by changing the hyperparameters in our algorithm. However, apart from cartpole, our algorithm is not able to generalize and solve other game environments. This is due to the increased complexity and large state space of the games like breakout, Pac-man, etc.

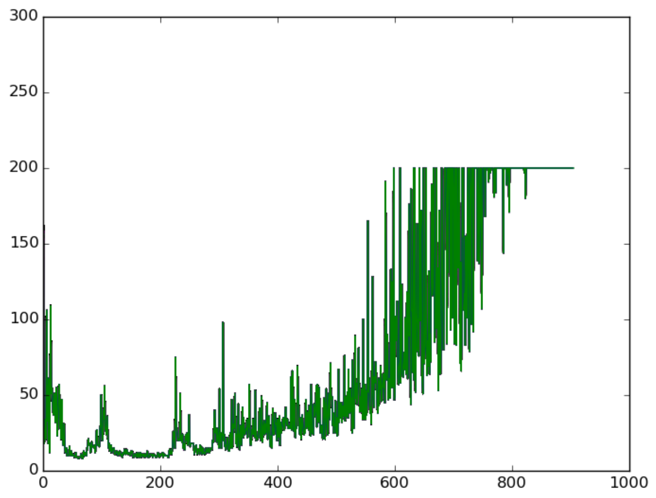


Figure 7.1: Training graph when Gamma is 0.99

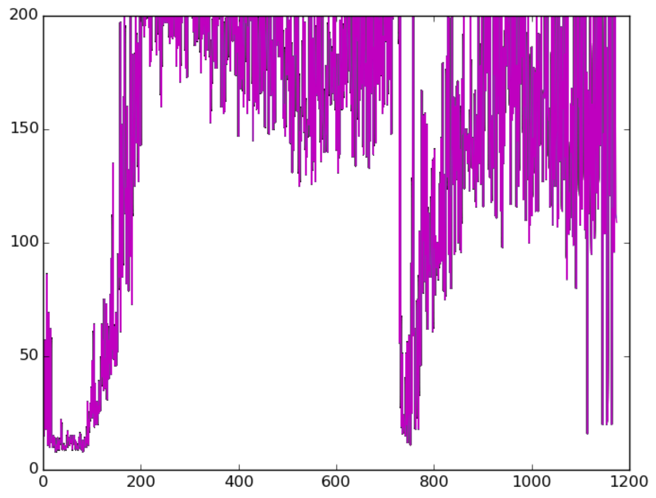


Figure 7.2: Training graph when Gamma is 0.75

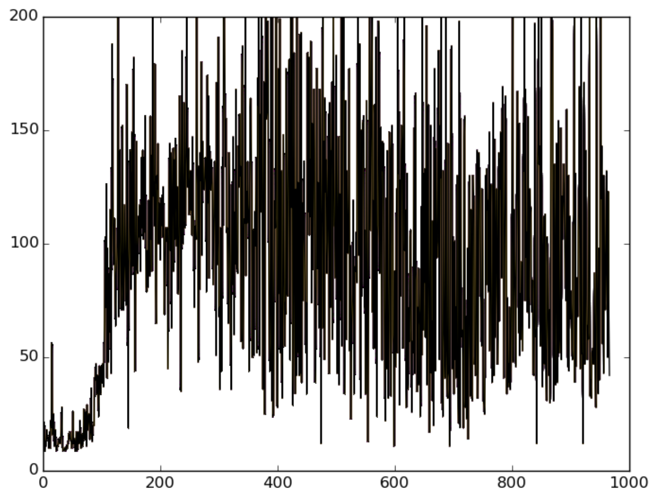


Figure 7.3: Training graph when Gamma is 0.50

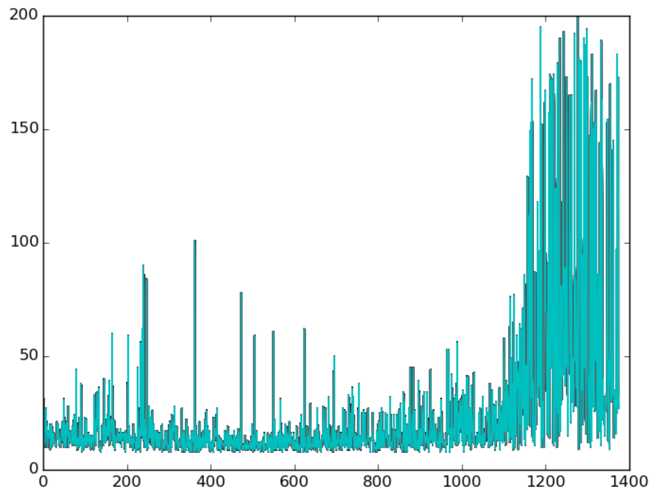


Figure 7.4: Training graph when Gamma is 0.25

Further, we can consider using convolution neural networks instead of simple neural networks to increase the learning rate and tackle the problem of increased complexity. Now our focus is to make a general algorithm which is capable of playing any game with only minor or no changes at all.

REFERENCES

[**1**] Gerald Tesauro. Temporal difference learning and td-gammon. Communications of the ACM, 38(3):58–68, 1995.

[**2**] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: Nature 518.7540 (2015), pp. 529–533.

[**3**] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: arXiv preprint arXiv:1312.5602 (2013).”

[**4**] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.”

[**5**] Hershey, David, and Blake Wulfe. ”Learning to Play Atari Games.”,D. (2015). Stanford Machine Learning class, 2-3.”

[**6**] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. arXiv preprint.”

Chapter 8

Appendix

Minimum User Requirements:

The minimum hardware requirement is 256 MB RAM and a very good internet connection and a good graphics card

User ManualL:

Following figure shows the screenshot of Cartpole game. One of the simplest games. We have trained an algorithm which achieves the highest possible score of 200 in a matter of minutes.

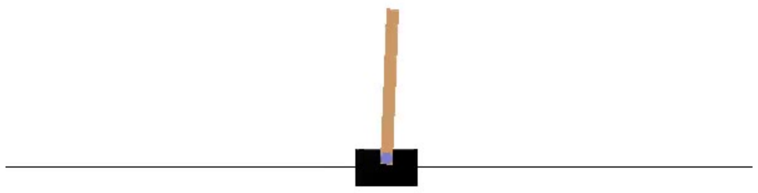
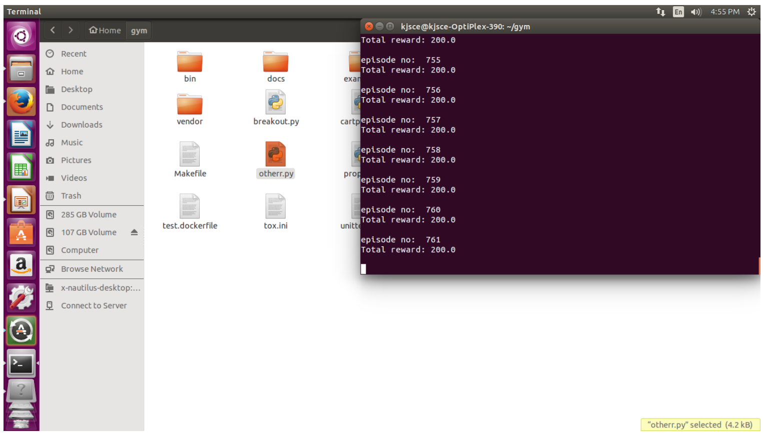


Figure 8.1: Cartpole game

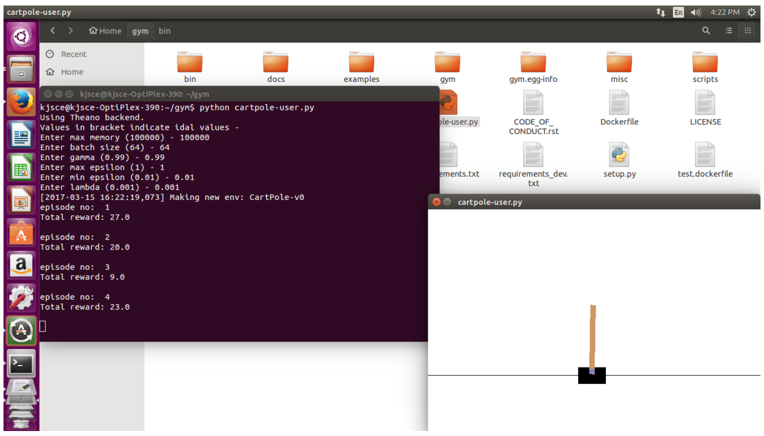
Following figure shows the instant when the game is just started.



```
kjsce@kjsce-OptiPlex-390: ~/gym
Total reward: 200.0

episode no:  755
Total reward: 200.0

episode no:  756
Total reward: 200.0

episode no:  757
Total reward: 200.0

episode no:  758
Total reward: 200.0

episode no:  759
Total reward: 200.0

episode no:  760
Total reward: 200.0

episode no:  761
Total reward: 200.0
```

Figure 8.2: Game satrted

Game achieves the highest possible score.



```
kjsce@kjsce-OptiPlex-390: ~/gym
kjsce@kjsce-OptiPlex-390:~/gym$ python cartpole-user.py
Using Theano backend.
Values in bracket indicate idal values -
Enter max memory (100000) - 100000
Enter batch size (64) - 64
Enter gamma (0.99) - 0.99
Enter max epsilon (1) - 1
Enter min epsilon (0.01) - 0.01
Enter lambda (0.001) - 0.001
[2017-03-15 16:22:19,073] Making new env: CartPole-v0
episode no:  1
Total reward: 27.0

episode no:  2
Total reward: 20.0

episode no:  3
Total reward: 9.0

episode no:  4
Total reward: 23.0
```

Figure 8.3: High score achieved