**SOLVING A TWO-DIMENSIONAL ENVIRONMENT USING REINFORCEMENT LEARNING TECHNIQUES**

**A Project Report submitted in partial fulfilment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

## Dokula Sanjay, 121810307042

## Sahithi, 121810307028

## Praveen, 121810307019

**Ritesh, 121810307020**

**Under the esteemed guidance of**

**Ms Bharti Gudla**

**Assistant Professor**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GITAM**

**(Deemed to be University)**

**VISAKHAPATNAM**

**MARCH 2022**

**DEPARTMENT OF COMPUTER SCIENCE AND**

**ENGINEERING**

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM**

## (Deemed to be University)



**DECLARATION**

We,hereby declare that the project report entitled “**SOLVING A TWO-DIMENSIONAL ENVIRONMENT USING REIFORCEMENT LEARNING TECHNIQUES**” is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

**121810307042 Dokula Sanjay**

**121810307028 Sahithi**

**121810307019 Pravenn**

**121810307020 Ritesh**

# **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM**

## (Deemed to be University)



# **CERTIFICATE**

This is to certify that the project report entitled “**SOLVING A TWO DIMENSIONAL ENVIRONMENT USING REINFORCEMENT LEARNING TECHNIQUES**” is a bonafide record of work carried out by **Dokula Sanjay (121810307042),Sahithi(121810307028),Ritesh(121810307020)** students submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

|  |  |
| --- | --- |
| **Project Guide** | **Head of the Department** |
|  |  |
| **Ms. Bharti Gudla** | **Dr. Sireesha R** |
| **Assistant Professor** | **Professor** |

**ACKNOWLEDGEMENT**

The success and final outcome of this project required a lot of guidance and assistance from our guide and we are extremely privileged to have got this all along the completion of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank our guide.

We respect and thank Ms Bharti Gudla, for enlightening us for our project on Solving a Two-Dimensional Environment Using Reinforcement Learning and giving us all support and guidance, which made us complete the project duly. We are extremely thankful to sir for providing such a nice support and guidance even while having a busy schedule.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1. | Abstract | |
| 2. | Introduction | |
| 3. | Literature Review | |
| 4. | Problem Identification & Objectives | |
| 5. | System Methodology | |
| 6. | Overview of Technologies  6.1 Q-learning  6.2 Proximal Policy Optimisation(PPO) | |
| 7. | Implementation  7.1 Working | |
|  | 7.2 | Coding |
| 8. | Results & Discussions | |
| 9. | Conclusion & Future Scope | |
| 10. | References | |

1. **ABSTRACT**

Reinforcement learning (RL) is an area of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) concerned with how [intelligent agents](https://en.wikipedia.org/wiki/Intelligent_agent) ought to take [actions](https://en.wikipedia.org/wiki/Action_selection) in an environment to maximise the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside [supervised](https://en.wikipedia.org/wiki/Supervised_learning) and [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning). In this project, we show how to make a Reinforcement Learning agent learn in any environment that it is put in and demonstrate why Q-learning is not advisable to use in a high dimension environment or an environment with continuous observation space with relatively big intervals. In this process, we will come across OpenAI/Gym and stablebaselines3 in implementing the above concept

To ease usage, we have also developed a Graphical Interface using the PyQt Library, a ported version from the c++ Qt library to python.

1. **INTRODUCTION**

Reinforcement Learning (RL) is a machine learning technique where an agent learns the optimal action for a given task through its repeated interaction with a dynamic environment that either rewards or penalises the agent’s action. Reinforcement learning is a semi-supervised learning approach where the cost/loss value required for training the model is made available indirectly in the form of rewards provided by the environment. Reinforcement learning is more suitable for learning dynamic environment interactions rather than learning static patterns between two sets of input and output values. Over the years, a number of reinforcement learning methods and architectures have been proposed with varying success. However, the recent success of deep learning algorithms has revived the field of reinforcement learning finding renewed interest among researchers who are now successfully applying this to solve very complex problems which were considered intractable earlier [1]. Events such as artificial agents like AlphaGo beating world chapmpion Lee Sedol [3] [9] or IBM Watson winning the game of Jeopardy [5] [14] has attracted worldwide attention towards the rise of artificial intelligence which may surpass human intelligence in the near future [11] [4]. Reinforcement learning is a crucial paradigm to build such intelligent systems which can learn from its experience over time. Reinforcement algorithms are now being increasingly applied to Robotics, healthcare, recommender system, data centres, smart grids, stock markets and transportation [13].

There are two sections that work together; the dynamic environment and the RL agent that plays the environment to learn the optimal policy. Policy here is the function that maps the states and actions to maximise the cumulative reward.

1. **LITERATURE REVIEW**

[**Guillaume Lample**](https://arxiv.org/search/cs?searchtype=author&query=Lample%2C+G)**, et al.** presented the first architecture to tackle 3D environments in first-person shooter games, that involve partially observable states. The architecture substantially outperformed the built-in AI agents of the game as well as humans in deathmatch scenarios.

[**Volodymyr Mnih**](https://paperswithcode.com/author/volodymyr-mnih) **et al.** in 2013 presented the first deep learning model to learn control policies directly from high-dimensional sensory input using reinforcement learning successfully. 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.

[**Swagat Kumar**](https://arxiv.org/search/cs?searchtype=author&query=Kumar%2C+S)provided the comparisons between Q-learning and DQN in a 2D environment of continuous observation space called a cart pole system. This paper showed the shortcoming of Q-learning in continuous observation space environments and how the DQN overcame this problem.

**Łukasz Kaiser et al.** has published a paper showing how a model-based reinforcement learning agent can solve the Atari games with fewer interactions than a model-free method. They described Simulated Policy Learning, a complete model-based deep RL algorithm based on video prediction models.

**Jason Rennie,** in his work, presented a novel way of creating web spiders using reinforcement learning and argues that it is the best way to do it.

**D. M. Roijers et al.** surveys algorithms designed for sequential decision-making problems with multiple objectives. Though there is a growing body of literature on this subject, little of it makes explicit under what circumstances special methods are needed to solve multi-objective problems. Therefore, we identify three distinct scenarios in which converting such a problem to a single-objective one is impossible, infeasible, or undesirable.

Reinforcement Learning (RL) algorithms can suffer from poor sample efficiency when rewards are delayed and sparse. [**Andrew Levy**](https://arxiv.org/search/cs?searchtype=author&query=Levy%2C+A) **et al.** introduced a solution that enables agents to learn temporally extended actions at multiple levels of abstraction in a sample efficient and automated fashion. Our approach combines universal value functions and hindsight learning, allowing agents to learn policies belonging to different time scales in parallel. We show that our method significantly accelerates learning in a variety of discrete and continuous tasks.

[**Lucian Buşoniu**](https://link.springer.com/chapter/10.1007/978-3-642-14435-6_7#auth-Lucian-Bu_oniu)

1. **PROBLEM IDENTIFICATION AND OBJECTIVES**

Our primary objective here is to produce an RL agent that has the optimal policy for the given dynamic environment. The agent should quickly and efficiently learn the behavior of the environment that gives the best possible reward. To demonstrate we’ve made a sample arcade style snake game environment which has action space of size 4 and observation space of 36. The observation space depends on the implementor. A traditional Q-learning would not do well in this environment so we opted for a policy gradient method called PPO.

THE DATASET

Unlike supervised learning techniques, RL doesn’t need any dataset to train on as it learns by interacting with a dynamic environment. A model-free does indeed does not require any dataset, but a model-based RL requires priori on environments states

TOOLS USED

* IDEs/Platforms:
  + Visual Studio Code, Tensorboard, anaconda prompt, git
* Programming Language:
  + Python 3.10
* Python Libraries:
  + NumPy, OpenCV, stablebaselines3, gym, matplotlib, Tensorboard, PyQt6

TECHNICAL TERMS

**Actions:**Actions are the [*Agent*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#b6a2)*’s*methods which allow it to interact and change its [*environment*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#4311), and thus transfer between [*states*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#c274). Every action performed by the Agent yields a [*reward*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#6a6f) from the environment. The decision of which action to choose is made by the [*policy*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#a76c).

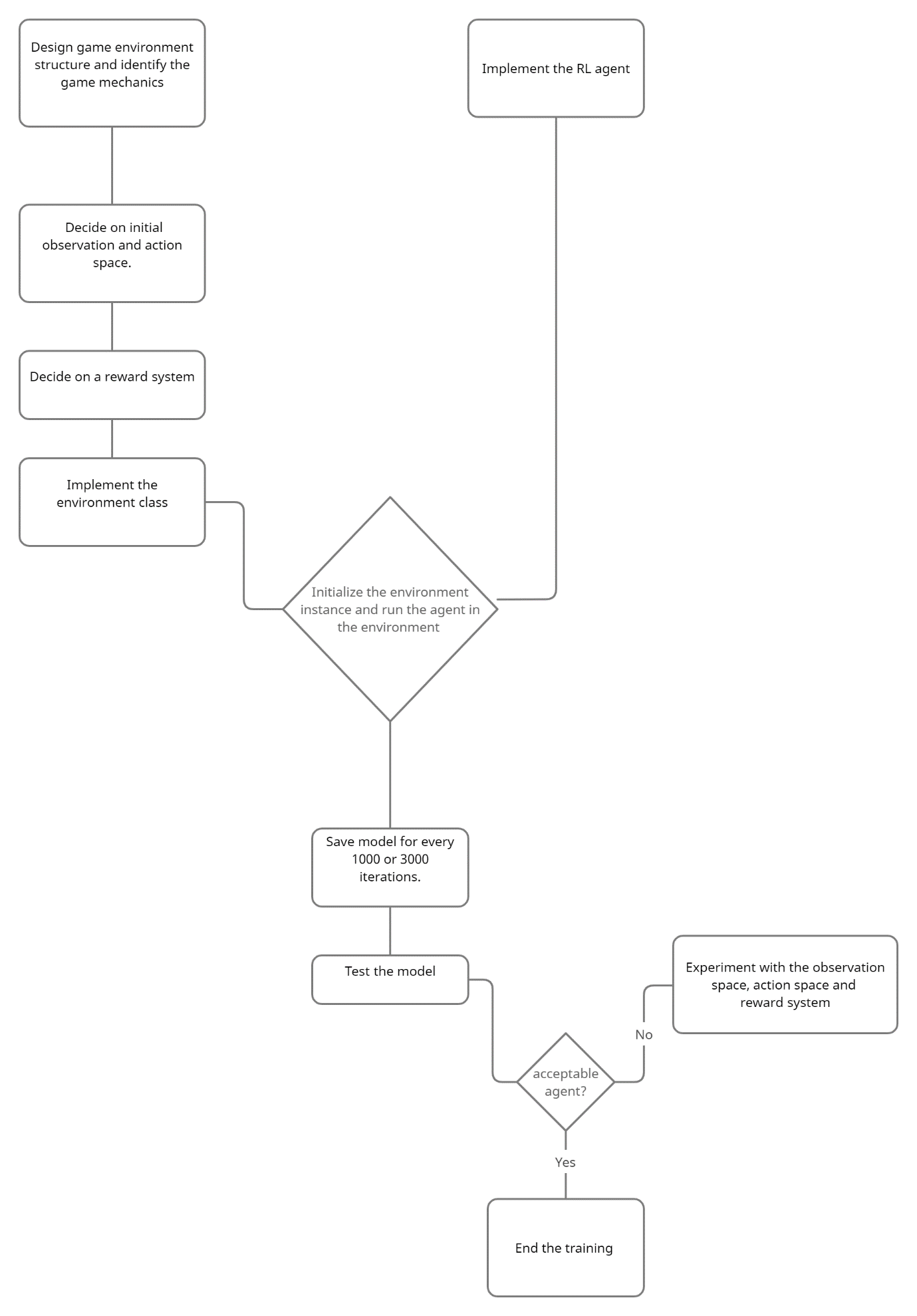
**Advantage Function:**Usually denoted as *A(s,a)*, the Advantage function is a measure of how much is a certain [*action*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#8751)a good or bad decision given a certain [*state*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#c274) — or more simply, what is the advantage of selecting a certain action from a certain state. It is defined mathematically as:

**Agent:** The learning and [*acting*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#8751) part of a [*Reinforcement Learning*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#fc9f) problem, which tries to maximize the [*rewards*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#6a6f) it is given by the [*Environment*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#4311). Putting it simply, the Agent is the model which you try to design.

**Discount Factor (γ)**: The discount factor, usually denoted as γ, is a factor multiplying the future expected [*reward*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#6a6f), and varies on the range of [0,1]. It controls the importance of the future rewards versus the immediate ones.

**Episode:**All [*states*](https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e#c274) that come in between an initial-state and a terminal-state It is important to remember that different episodes are completely independent of one another.

1. **SYSTEM METHODOLOGY**

****

1. **OVERVIEW OF TECHNOLOGIES**

6.1 Q-learning

One of the early breakthroughs in reinforcement learning was the development of an o↵-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by

In this case, the learned action-value function, Q, directly approximates q⇤, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state–action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.

Algorithm:

Initialise Q(s,a) for all s S

Loop for each episode:

Initialise S

Loop for each step of episode:

Choose A from S using policy derived from Q

Take action A, observe R, S’

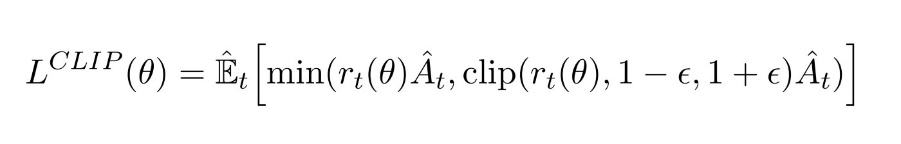
S = S’

Until S is terminal

6.2 Proximal Policy Optimisation(PPO):

Proximal Policy Optimisation (PPO) is a recent advancement in Reinforcement Learning, which provides an improvement on Trust Region Policy Optimization (TRPO). This algorithm was proposed in 2017 and showed remarkable performance when OpenAI implemented it. To understand and appreciate the algorithm, we first need to understand what a policy is. A policy, in Reinforcement Learning terminology, is a mapping from action space to state space. It can be imagined to be instructions for the RL agent in terms of what actions it should take based upon which state of the environment it is currently in. When we talk about evaluating an agent, we generally mean evaluating the policy function to find out how well the agent is performing, following the given policy. This is where Policy Gradient methods play a vital role. When an agent is “learning” and doesn’t really know which actions yield the best result in the corresponding states, it does so by calculating the policy gradients. It works like a neural network architecture, whereby the gradient of the output, i.e, the log of probabilities of actions in that particular state, is taken with respect to parameters of the environment and the change is reflected in the policy, based upon the gradients.

 The most common implementation of PPO is via the Actor-Critic Model which uses 2 Deep Neural Networks, one taking the action(actor) and the other handling the rewards(critic). The mathematical equation of PPO is shown below:



The updated policy will be ε-clipped to a small region so as to not allow huge updates which might potentially be irrecoverably harmful. However, it also ensures that the old policy and new policy are at least at a certain proximity (denoted by ε), and very large updates are not allowed.

1. **IMPLEMENTATION**

**7.1 Working**

First researched on search algorithms and found out that they do not perform like a human.So looked at cnn approach for human like behaviour. It needs a lot of data to train on, we need an approach that needs to work without any previous data and interact with a dynamic environment.

RL techniques are the best suited for this kind of problem. After testing algorithms like Q-learning, DQN, and some policy gradient methods found out that Q-learning is not suitable for high dimension observation space environment, DQN does not converge always and needs add ons, PPO a proximal policy optimization although takes a bit time but always converges to the optimal policy.

The project workflow is:

1. First the environment is selected and then load a model if already trained or else train the agent and save the model.
2. Next run the agent in the environment
3. All this is made esay to do by createing a gui interface with PyQt

**7.2 Coding:**

**SnakeEnv.py**

from pickletools import uint8

import gym

from gym import spaces

import cv2

import numpy as np

import random

import time

from collections import deque

import sys

SNAKE\_LEN\_GOAL = 50

def collision\_with\_apple(apple\_position, score):

    apple\_position = [random.randrange(1,50)\*10,random.randrange(1,50)\*10]

    score += 1

    return apple\_position, score

def collision\_with\_boundaries(snake\_head):

    if snake\_head[0]>=500 or snake\_head[0]<0 or snake\_head[1]>=500 or snake\_head[1]<0:

        # print("collided")

        return 1

    else:

        return 0

def collision\_with\_self(snake\_position):

    snake\_head = snake\_position[0]

    if snake\_head in snake\_position[1:]:

        # print('collided with body')

        return 1

    else:

        return 0

N\_DISCRETE\_ACTIONS = 4

class SnakeEnv(gym.Env):

    """Custom Environment that follows gym interface"""

    metadata = {'render.modes': ['human']}

    def \_\_init\_\_(self):

        super(SnakeEnv, self).\_\_init\_\_()

        # Define action and observation space

        # They must be gym.spaces objects

        # Example when using discrete actions:

        self.action\_space = spaces.Discrete(N\_DISCRETE\_ACTIONS)

        # Example for using image as input (channel-first; channel-last also works):

        self.observation\_space = spaces.Box(low=-500, high=500,

                                            shape=(5+SNAKE\_LEN\_GOAL,), dtype=np.float32)

    def step(self, action):

        self.prev\_actions.append(action)

        # cv2.imshow("Snake",self.image)

        # cv2.waitKey(50)

        # self.image = np.zeros((500,500,3),dtype='uint8')

        # # test the tuple(self.apple\_position) in rectangle

        # cv2.rectangle(self.image,(self.apple\_position[0],self.apple\_position[1]),(self.apple\_position[0]+10,self.apple\_position[1]+10),(0,0,255),3)

        # for pos in self.snake\_position:

            # print("snek")

            # cv2.rectangle(self.image,(pos[0],pos[1]),(pos[0]+10,pos[1]+10),(0,255,0),3)

        #     # print(self.snake\_position)

        #     # cv2.waitKey(1000)

        # self.render()

        time\_to\_take\_step = time.time() + 0.05

        k= -1

        while time.time() > time\_to\_take\_step:

            if k == -1:

                k = cv2.waitKey(1)

            else:

                continue

        # try changing +=10 to +=5 if somthing is not working properly

        if action == 1:

            self.snake\_head[0]+=10

        elif action == 0:

            self.snake\_head[0]-=10

        elif action == 2:

            self.snake\_head[1]+=10

        elif action == 3:

            self.snake\_head[1]-=10

        # if action == 1 and self.previous\_button\_direction !=0:

        #     self.snake\_head[0]+=10

        # elif action == 0 and self.previous\_button\_direction !=1:

        #     self.snake\_head[0]-=10

        # elif action == 2 and self.previous\_button\_direction !=3:

        #     self.snake\_head[1]+=10

        # elif action == 3 and self.previous\_button\_direction !=2:

        #     self.snake\_head[1]-=10

        # print("snake head ",self.snake\_head)

        apple\_reward = 0

        if self.snake\_head == self.apple\_position:

            # print("at ap",self.snake\_position,self.snake\_head)

            self.apple\_position, self.score = collision\_with\_apple(self.apple\_position,self.score)

            self.snake\_position.insert(0,list(self.snake\_head))

            apple\_reward=10000

            # print("at apple",self.snake\_position)

        else:

            # print("at norm",self.snake\_position,self.snake\_head)

            self.snake\_position.insert(0,list(self.snake\_head))

            self.snake\_position.pop()

            # print("normal",self.snake\_position)

        if collision\_with\_boundaries(self.snake\_head) == 1 or collision\_with\_self(self.snake\_position) == 1:

            # print("dead")

            font = cv2.FONT\_HERSHEY\_SIMPLEX

            self.image = np.zeros((500,500,3),dtype='uint8')

            cv2.putText(self.image,'Your Score is {}'.format(self.score),(140,250), font, 1,(255,255,255),2,cv2.LINE\_AA)

            cv2.imshow('Snake',self.image)

            # cv2.waitKey(100)

            self.done = True

            # self.total\_reward = len(self.snake\_position) - 3

            # self.reward = self.total\_reward - self.prev\_reward

            # self.prev\_reward = self.total\_reward

        euclidean\_dist\_apple = np.linalg.norm(np.array(self.snake\_head) - np.array(self.apple\_position))

        self.total\_reward = ((250 - euclidean\_dist\_apple) + apple\_reward)/100

            # self.reward = self.total\_reward - self.prev\_reward

            # self.prev\_reward = self.total\_reward

        if self.done:

            self.total\_reward = -10

        info = {}

        head\_x = self.snake\_head[0]

        head\_y = self.snake\_head[1]

        snake\_length = len(self.snake\_position)

        apple\_delta\_x = self.apple\_position[0] - head\_x

        apple\_delta\_y = self.apple\_position[1] - head\_y

        # create observation:

        self.observation = [head\_x, head\_y, apple\_delta\_x, apple\_delta\_y, snake\_length] + list(self.prev\_actions)

        self.observation = np.array(self.observation)

        return self.observation, self.total\_reward, self.done, info

    def reset(self):

        self.image = np.zeros((500,500,3),dtype='uint8')

        self.snake\_position = [[250,250],[240,250],[230,250]]

        self.apple\_position = [np.random.randint(low=1,high=50)\*10,np.random.randint(low=1,high=50)\*10]

        self.score = 0

        self.reward = 0

        self.previous\_button\_direction = 1

        self.button\_direction = 1

        self.snake\_head = [250,250]

        self.prev\_reward = 0

        self.done = False

        head\_x = self.snake\_head[0]

        head\_y = self.snake\_head[1]

        snake\_length = len(self.snake\_position)

        apple\_delta\_x = self.apple\_position[0] - head\_x

        apple\_delta\_y = self.apple\_position[1] - head\_y

        self.prev\_actions = deque(maxlen=SNAKE\_LEN\_GOAL)

        for i in range(SNAKE\_LEN\_GOAL):

            self.prev\_actions.append(-1)

        self.observation = [head\_x, head\_y, apple\_delta\_x, apple\_delta\_y, snake\_length] + list(self.prev\_actions)

        self.observation = np.array(self.observation)

        return self.observation  # reward, done, info can't be included

    def render(self):

        cv2.imshow("Snake",self.image)

        cv2.waitKey(50)

        self.image = np.zeros((500,500,3),dtype='uint8')

        cv2.rectangle(self.image,(self.apple\_position[0],self.apple\_position[1]),(self.apple\_position[0]+10,self.apple\_position[1]+10),(0,0,255),3)

        for pos in self.snake\_position:

            # print("snek")

            cv2.rectangle(self.image,(pos[0],pos[1]),(pos[0]+10,pos[1]+10),(0,255,0),3)

    def close(self):

        sys.exit()

**ModelLoad.py**

import gym

from stable\_baselines3 import A2C,PPO

import os

from SnakeEnv import SnakeEnv

import cv2

import sys

class Model():

    def \_\_init\_\_(self,\*args):

        # self.models\_dir = "models/PPO-1647365718"

        # self.env = SnakeEnv()

        # self.model\_path = f"{self.models\_dir}/{self.model\_name}"

        if len(args)>1 and isinstance(args[1],str):

            print("2 arg and env arg")

            if args[1] == "Snake Environment":

                print("selecting snake env")

                self.env = SnakeEnv()

            elif args[1] ==  "MountainCar Environment":

                print("selecting mountain car env")

                self.env = gym.make("MountainCar-v0")

        else:

            print("selecting default env")

            self.env = SnakeEnv()

            self.env.reset()

        if len(args)>0 and isinstance(args[0],str):

            print("1 arg and model arg")

            print(f"Selecting {args[0]} model")

            self.model\_name = args[0]

        else:

            print("selecting default model")

            self.model\_name = "models/PPO-1647365718/1920000.zip"

        self.model\_path = self.model\_name

        self.model = PPO.load(self.model\_path, env=self.env)

        self.episodes = 50

    def run(self):

        for ep in range(self.episodes):

            obs = self.env.reset()

            done = False

            while not done:

                self.env.render()

                action, \_ = self.model.predict(obs)

                obs, reward, done, info = self.env.step(action)

                key = cv2.waitKey(50)

                if key == ord('q'):

                    # sys.exit()

                    self.env.close()

        # self.env.close()

**qlearning.py**

import gym

import numpy as np

import math as m

from numpy import average

import matplotlib.pyplot as plt

import pickle

# from numpy import block

from SnakeEnv import SnakeEnv

import time

env = gym.make("MountainCar-v0")

# env = SnakeEnv()

# env = gym.make("CartPole-v1")

# env = gym.make("LunarLander-v2")

alpha = 0.1

discount = .95

episodes = 30000

show\_every = 500

epsilon = .6

start\_epsilon\_decay = 1

end\_epsilon\_decay = episodes//2

q\_table = None

epsilon\_decay\_value = epsilon/(end\_epsilon\_decay - start\_epsilon\_decay)

discrete\_obs\_size = [10] \* len(env.observation\_space.low)

print(env.observation\_space.high)

print(len(env.observation\_space.low))

print(env.action\_space.n)

discrete\_obs\_wind\_size = (env.observation\_space.high - env.observation\_space.low) / discrete\_obs\_size

# discrete\_obs\_wind\_size = (np.array([m.pow(2,31)]\*8) - (np.array([-m.pow(2,31)]\*8))) / discrete\_obs\_size

print(discrete\_obs\_size)

print(discrete\_obs\_wind\_size)

print(len(discrete\_obs\_size+[env.action\_space.n]))

if q\_table is None:

    # initialize the q-table#

    q\_table = np.random.uniform(low=-2, high=0, size=(discrete\_obs\_size + [env.action\_space.n]))

else:

    with open(q\_table, "rb") as f:

        q\_table = pickle.load(f)

def get\_discrete\_state(state):

    discrete\_state = (state-env.observation\_space.low) / discrete\_obs\_wind\_size

    return tuple(discrete\_state.astype(int))

discrete\_state = get\_discrete\_state(env.reset())

metrics = {'ep':[], 'avg':[], 'min':[], 'max':[]}

ep\_rewards = []

for episode in range(episodes):

    episode\_reward = 0

    discrete\_state = get\_discrete\_state(env.reset())

    # env.reset()

    done = False

    if episode%show\_every == 0:

        print(episode)

        render = True

    else:

        render = False

    while not done:

        if np.random.random()>epsilon:

            action = np.argmax(q\_table[discrete\_state])

        else:

            action = np.random.randint(0,env.action\_space.n)

        new\_state, reward, done, info = env.step(action)

        new\_discrete\_state = get\_discrete\_state(new\_state)

        episode\_reward+=reward

        # print(new\_state)

        if render:

            env.render()

        if not done:

            max\_future\_q = np.max(q\_table[new\_discrete\_state])

            current\_q = q\_table[discrete\_state + (action,)]

            new\_q = (1-alpha) \* current\_q + alpha \* ( reward + discount \* max\_future\_q)

            q\_table[discrete\_state + (action,)] = new\_q

        # elif new\_state[0]>=env.goal\_position:

            # print(f"made it on episode {episode}")

            # q\_table[discrete\_state + (action,)]=0

        discrete\_state = new\_discrete\_state

    ep\_rewards.append(episode\_reward)

    if end\_epsilon\_decay >= episode >= start\_epsilon\_decay:

        epsilon-=epsilon\_decay\_value

    if not episode%show\_every:

        average\_reward = sum(ep\_rewards[-show\_every:])/show\_every

        metrics['ep'].append(episode)

        metrics['max'].append(max(ep\_rewards[-show\_every:]))

        metrics['min'].append(min(ep\_rewards[-show\_every:]))

        metrics['avg'].append(average\_reward)

env.close()

plt.plot(metrics['ep'],metrics['avg'],label="average")

plt.plot(metrics['ep'],metrics['min'],label="min")

plt.plot(metrics['ep'],metrics['max'],label="max")

plt.legend(loc=1)

plt.show()

with open(f"qtable-{int(time.time())}.pickle", "wb") as f:

    pickle.dump(q\_table, f)

**interface.py**

from PyQt6.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton,QFileDialog

from PyQt6.QtGui import QIcon, QAction

from PyQt6 import QtGui

from PyQt6 import QtCore

import sys

from PyQt6.QtCore import QProcess,QFileInfo

from ModelLoad import Model

import os

# from qlearning import load

class Window(QMainWindow):

    def \_\_init\_\_(self):

        super(Window,self).\_\_init\_\_()

        self.setGeometry(50,50,500,300)

        self.setWindowTitle("Reinforcement Learning")

        self.setWindowIcon(QIcon('forza\_ferrari.png'))

        self.model = None

        self.filename = None

        self.envName = None

        self.parameters ={'modelpath':"",'envname':""}

        self.home()

    def home(self):

        self.quit\_btn = QAction("quit",self)

        self.quit\_btn.setCheckable(True)

        self.quit\_btn.triggered.connect(QtCore.QCoreApplication.instance().quit)

        self.snake = QAction("load snake env",self)

        self.snake.setCheckable(True)

        self.snake.triggered.connect(self.load\_snake\_model)

        self.run\_btn = QPushButton("play",self)

        # self.run\_btn.setCheckable(True)

        self.run\_btn.clicked.connect(self.run)

        self.run\_btn.move(200,100)

        self.compile\_btn = QPushButton("compile agent",self)

        # self.run\_btn.setCheckable(True)

        self.compile\_btn.clicked.connect(self.compile)

        self.compile\_btn.move(200,150)

        self.select\_model = QAction("Select model",self)

        self.select\_model.triggered.connect(self.load\_model)

        self.Snake\_env = QAction("Snake Environment",self)

        self.Snake\_env.triggered.connect(self.select\_environment)

        self.Mount\_env = QAction("MountainCar Environment",self)

        self.Mount\_env.triggered.connect(self.select\_environment)

        self.statusBar()

        mainMenu = self.menuBar()

        fileMenu = mainMenu.addMenu('&File')

        fileMenu.addAction(self.quit\_btn)

        envMenu = mainMenu.addMenu('&Env')

        envMenu.addAction(self.Snake\_env)

        envMenu.addAction(self.Mount\_env)

        modelMenu = mainMenu.addMenu('&Model')

        modelMenu.addAction(self.snake)

        modelMenu.addAction(self.select\_model)

        self.show()

    def load\_snake\_model(self):

        self.model = Model()

    # def mountain(self):

    #     load()

    def run(self):

        self.model.run()

    def load\_model(self):

        filepath,wot = QFileDialog.getOpenFileName(self,"open model","~","model files(\*.zip)")

        path = os.path.normpath(filepath)

        path = path.split(os.path.sep)

        mod = f"{path[-3]}/{path[-2]}/{path[-1]}"

        self.filename = mod

        self.parameters['modelpath'] = self.filename

        print(mod)

        # self.model = Model(self.filename,self.envName)

    def select\_environment(self):

        action = self.sender()

        self.envName = action.text()

        print("action: ", self.envName)

        self.parameters['envname'] = self.envName

        # self.model = Model(self.filename,self.envName)

    def compile(self):

        self.model = Model(self.parameters['modelpath'],self.parameters['envname'])

app = QApplication(sys.argv)

GUI = Window()

# GUI.cmd

sys.exit(app.exec())

1. **RESULTS AND DISCUSSIONS**

After testing algorithms like Q-learning, DQN, and some policy gradient methods found out that Q-learning is not suitable for high dimension observation space environment, DQN does not converge always and needs add ons, PPO a proximal policy optimization although takes a bit time but always converges to the optimal policy.

The reward system was revised multiple times to adjust the agents behaviour for the environment. First it was same the game score which did not give any viable result, then tried out period rewards for getting a score, this had a bit progress in terms of moving toward the goal but still not acceptable. Finally tested another reward system based on Euclidean distance between agent and the goal and got a very solid result.

The following outputs were obtained:

1. **CONCLUSION AND FUTURE SCOPE**

Developing a system that can detect this fatal disease would be beneficial in remote areas where radiotherapists are unavailable. This paper aims to be fruitful in early diagnosis of pneumonia to prevent bad consequences. Not much specific work particularly pneumonia has been done so far in the dataset. Hence, the development of more algorithms in the field of bioinformatics and medicine can be worthy. In this model we applied a convolutional neural network for detection and extraction purposes. We observed in the model summary how the layered structure works on images and employed the help of adam optimiser and performed epochs. The simple interface is quick and works on this model efficiently for detection of pneumonia. The information provided in literature review like using of advanced neural networks can be helpful to gain greater accuracy as well as precise detection by utilising their deep layered architectures.

1. **REFERENCES**
2. **Journals/Articles**

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38, 2017.

[2] E. Bisong. Google colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform, pages 59–64. Springer, 2019.

[3] S. Borowiec. Alphago seals 4-1 victory over go grandmaster lee sedol. The Guardian, 15, 2016.

[4] J. Fang, H. Su, and Y. Xiao. Will artificial intelligence surpass human intelligence? Available at SSRN 3173876, 2018.

[5] D. A. Ferrucci. Introduction to this is watson. IBM Journal of Research and Development, 56(3.4):1–1, 2012.

[6] Google Colaboratory. Online gpu cloud by google. https:// colab.research.google.com/.

[7] A. Gulli and S. Pal. Deep learning with Keras. Packt Publishing Ltd, 2017.

[8] H. V. Hasselt. Double q-learning. In Advances in neural information processing systems, pages 2613–2621, 2010.

[9] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang. Overview on deepmind and its alphago zero ai. In Proceedings of the 2018 international conference on big data and education, pages 67–71, 2018.

[10] Kaggle. Online gpu cloud with datasets. https://www.kaggle. com/.

[11] P. Kraikivski. Seeding the singularity for ai. arXiv preprint arXiv:1908.01766, 2019.

[12] S. Kumar. Reinforcement learning code for cartpole system. https: //github.com/swagatk/RL-Projects-SK.git, 2020.

[13] Y. Li. Reinforcement learning applications. arXiv preprint arXiv:1908.06973, 2019.

[14] J. Markoff. Computer wins on jeopardy!: trivial, its not. New York Times, 16, 2011.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[16] OpenAI Gym. Toolkit for developing and comparing reinforcement learning algorithms. <https://gym.openai.com/>.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[18] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence, 2016.

[19] C. J. Watkins and P. Dayan. Q-learning. Machine learning, 8(3- 4):279–292, 1992.

1. **Websites**
2. PyQt basic tutorial: <https://pythonprogramming.net/basic-gui-pyqt-tutorial/>
3. PyQt6 Documentation : <https://www.riverbankcomputing.com/static/Docs/PyQt6/>
4. OpenAI/Gym : <https://gym.openai.com/>.