**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

## Marella Sahithi, 121810307028

**Rokkam 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 a branch of machine learning that studies how intelligent agents should operate in a given environment in order to maximise cumulative reward. Along with supervised and unsupervised learning, reinforcement learning is one of three basic machine learning paradigms. 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 in which an agent learns the best action to take for a given task by interacting with a dynamic environment that either rewards or punishes the agent's actions. Reinforcement learning is a semi-supervised learning method in which the model's cost/loss value is delivered indirectly through the environment's incentives. Reinforcement learning is more suited to learning dynamic environmental interactions than static patterns between two sets of input and output values. Many reinforcement learning approaches and architectures have been developed throughout the years, with varied degrees of success. The recent success of deep learning algorithms, on the other hand, has reignited interest in reinforcement learning, which is currently being utilised to solve extremely difficult problems that were previously thought to be unsolvable [1].Artificial agents such as AlphaGo [3] [9] beating world champion Lee Sedol [3] [9] or IBM Watson [5] [14] winning the game of Jeopardy [5] [14] have drew international attention to the emergence of artificial intelligence, which may soon surpass human intelligence [11]. [4]. Reinforcement learning is crucial To create intelligent systems that can learn from their experiences throughout time, a new paradigm is needed. Robotics, healthcare, recommender systems, data centres, smart grids, financial markets, and transportation are all using reinforcement algorithms currently [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.

In 2013, **Volodymyr Mnih et al**. published the first deep learning model that effectively learned control policies from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network trained using a Q-learning variation, using raw pixels as input and a value function forecasting future rewards as output.

[**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**. look at methods for multiple-objective sequential decision-making situations. Despite the fact that there is a growing volume of literature on the issue, little of it specifies when unique methods are required to address multi-objective problems. As a result, we identify three instances in which reducing a multi-objective problem to a single-objective problem is either impossible, infeasible, or undesirable.

When rewards are delayed and sparse, Reinforcement Learning (RL) algorithms might suffer from poor sample efficiency. **Andrew Levy et al**. presented a method for agents to learn temporally extended actions at several levels of abstraction in a sample efficient and automated manner. Our method combines universal value functions and hindsight learning, allowing agents to simultaneously learn policies over many time scales. In a range of discrete and continuous tasks, we show that our strategy dramatically accelerates learning.

**Lucian Buşoniu** **et al.** in this study presents a thorough examination of multiagent reinforcement learning (MARL). The formal description of the multiagent learning goal is a key topic in the area. Different perspectives on this subject have resulted in the formulation of numerous goals, two of which stand out: the stability of the agents' learning dynamics and adaptation to the changing behaviour of other agents.

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:** The Agent's methods for interacting with and changing its environment, and so transferring between states, are called actions. The environment rewards the Agent for every action he or she does. The policy determines which course of action to take.

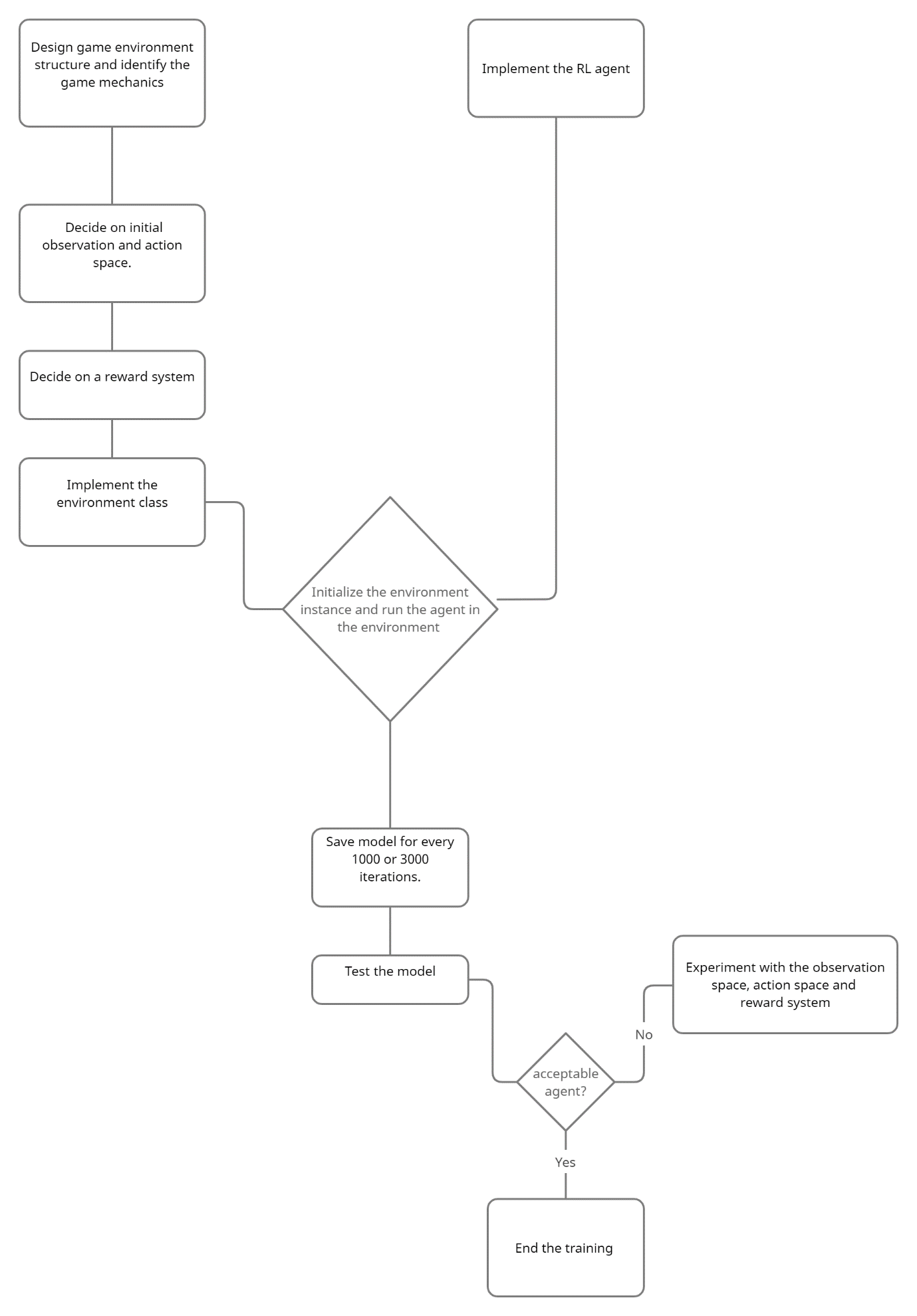
**Advantage Function:** The Advantage function, abbreviated as A(s,a), is a measure of how much a specific action is a good or bad option given a specific state — or, to put it another way, what is the benefit of choosing a specific action from a specific condition. It is mathematically defined as:

**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

The creation of an off-policy TD control method known as Q-learning (Watkins, 1989) was one of the early triumphs in reinforcement learning.

In this scenario, regardless of the policy used, the learned action-value function, Q, directly approximates q, the ideal action-value function. This greatly simplifies the algorithm's analysis and allows for early convergence proofs. In that it affects which state–action pairings are visited and altered, the policy still has an impact. All that is required for proper convergence is that all pairings be updated continuously.

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

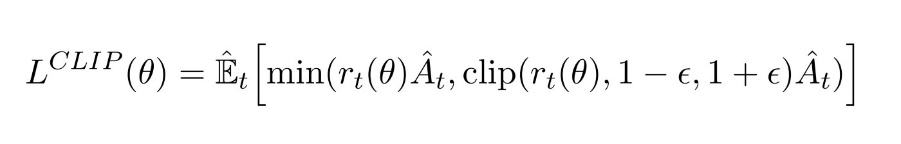
Take action A, observe R, S’

S = S’

Until S is terminal

6.2 Proximal Policy Optimisation(PPO):

PPO (Proximal Policy Optimisation) is a new innovation in Reinforcement Learning that improves Trust Region Policy Optimization (TRPO). This approach was proposed in 2017 and performed admirably when it was implemented by OpenAI. To comprehend and appreciate the algorithm, we must first comprehend the concept of a policy. A policy is a mapping from action space to state space in Reinforcement Learning. It can be thought of as instructions for the RL agent as to what activities it should do based on the present state of the environment. When we talk about assessing an agent, we're usually talking about evaluating the policy function to see how well the agent follows the policy. Policy Gradient approaches are extremely useful in this situation. When an agent is "learning," it calculates policy gradients since it doesn't know which actions produce the best results in the corresponding states. It functions similarly to a neural network design, in that the gradient of the output, i.e., the log of probabilities of actions in that particular state, is calculated with respect to environmental parameters, and the change is represented in the policy. The Actor-Critic Model, which uses two Deep Neural Networks, one taking action (actor) and the other handling rewards, is the most frequent implementation of PPO (critic). PPO's mathematical equation is as follows:



The modified policy will be -clipped to a small zone in order to prevent large revisions that could be irreversibly detrimental. It also assures that the old and new policies are at least close in proximity (as indicated by ), and that excessively big revisions are not permitted.

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. To train, it requires a large amount of data. 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 SnakeEnv import SnakeEnv

import time

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

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)

    def select\_environment(self):

        action = self.sender()

        self.envName = action.text()

        print("action: ", self.envName)

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

    def compile(self):

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

app = QApplication(sys.argv)

GUI = Window()

# GUI.cmd

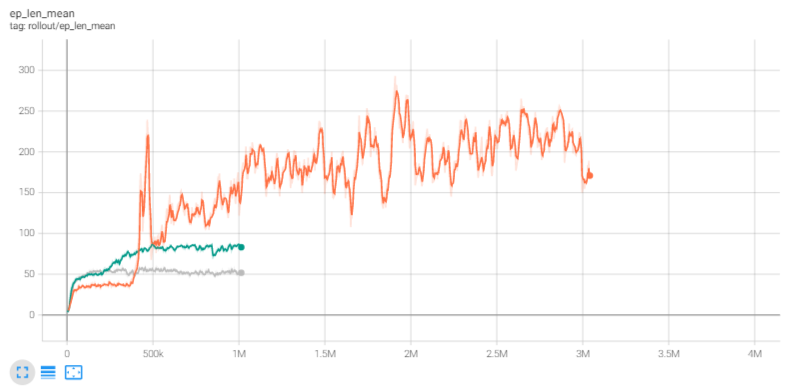
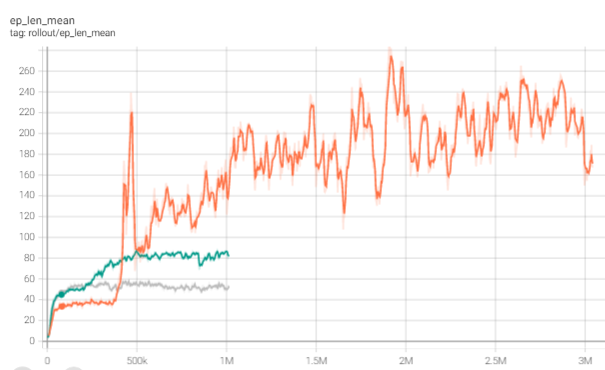
sys.exit(app.exec())

1. **RESULTS AND DISCUSSIONS**

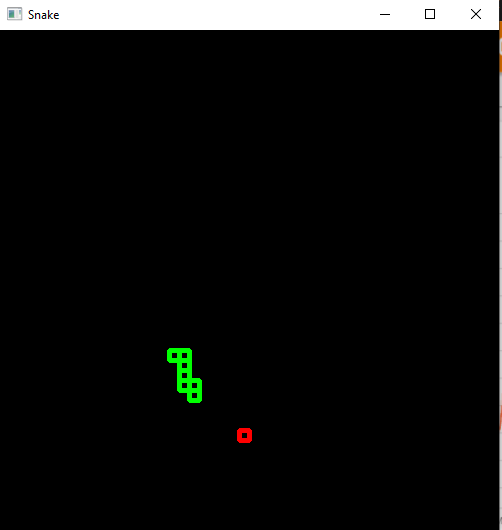
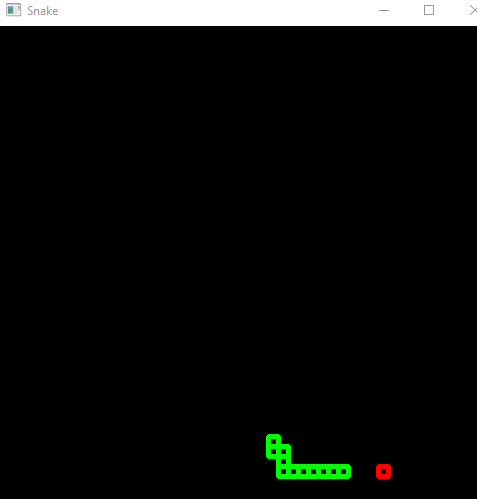
After testing algorithms like Q-learning, DQN, and some policy gradient methods found out that Q-learning is not suitable for high dimension observation space environment, DQN does not converge always and needs add ons, PPO a proximal policy optimization although takes a bit time but always converges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

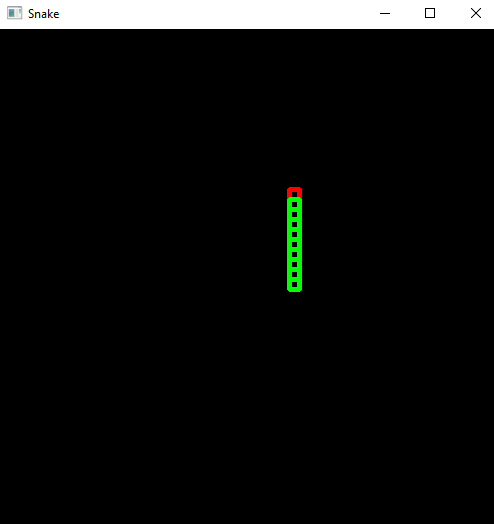
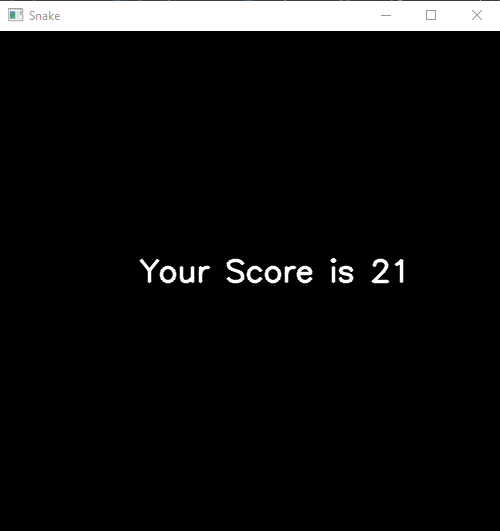
The reward system was revised multiple times to adjust the agents behaviour for the environment. First it was same the game score which did not give any viable result, then tried out period rewards for getting a score, this had a bit progress in terms of moving toward the goal but still not acceptable. 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:



Renderings of the environment while the agent is interacting.





1. **CONCLUSION AND FUTURE SCOPE**

Reinforcement Learning is the newest and most promising for of machine learning method to train the behaviour of control systems like network routing, game AI, human brain replicating, etc. This project can further be pushed to 3D environments. There are a lot of domains where RL can do a much better and efficient job than existing approaches, for example we can simulate a football game to see the many ways the agent tries as it is very fast in computations and we can evaluate a few interesting startegies to use. Another can be in fluid mechanics, we can make a simulation to create a car chasis that the agent can modify. This can lead to very interesting and innovative also naïve strategies and models to increase the aerodynamics or drags or whatever the critera need.

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/>.
5. Reinforcement Learning Tutorials : <https://pythonprogramming.net>