1.INTRODUCTION

1.1 INTRODUCTION

The Reinforcement learning problem involves an agent interacting with an unknown environment, by executing a sequence of actions and learning from the observations and rewards received . Just like how a baby learns to walk, by failing and falling several times The goal of the agent is to maximize cumulative rewards. Unlike a classic planning problem, an RL agent doesn't start with perfect knowledge of the environment, but learns through experience. the behaviour of animals and people can be modulated by providing different rewards and punishments. how deep learning has been applied to the reinforcement learning problem, the challenges faced and how they have been overcome. We then proceed to implement several of the most recent works and analyze their performance on the same test based. We use diffirent learning environment as our test based for this purpose and compare different algorithms based on the cumulative scores obtained by the respective trained agents. We start with an open-source implementation of Deep Q Learning written in Python. In games, machine learning can be used for various purposes. It can be used to obtain maximum scores, win the game at the minimum time possible, obtain most collectables, or improve survivability. As we do best to implement reinforcement learning method as it can learn by itself from its surrounding. Reinforcement learning (RL) is a suitable learning method because its goal is to find the optimum ways or methods to solve problems. After each iteration, its knowledge will improve, thus, produce better problem- solving. RL also has its selection of techniques such as Q- learning. Deep learning techniques, we plan on designing a model that will train the machine to make critical decisions in order to play the game well. This project is aimed at understanding how effective deep learning algorithms are in training a machine to think like a human being. Q-learning and State-Action-Reward method are choosen as both are almost similar except Q-learning is on-policy algorithm. train agents to play trivial games like Flappy Bird, and Reinforcement Learning to train the agents to play more complicated games where the tasks are non-trivial.

1.2 EXISTING SYSTEM

1.2.1 NON- ML TECHNIQUES

In This games of Agent actually has a trivial, unbeatable solution. Create a cyclic path that goes through every square on the board without crossing itself, this is known as a Hamiltonian Cycle. and then just keep following this cyclical path until the Agent is growing up. This will work every time, but it is very boring and also wastes a lot of moves. In an N x N grid, it will take ~N^2 targets to grow fill the board. If the target appear randomly, we would expect that the agent will need to pass through half the currently open squares to reach the target from its current position, or around $N^2/2$ moves at the start of the game. Since this number decreases as the Agent goe down, we expect that on average, the agent will need ~$N^4/4$ moves to beat the game using this strategy. This is about 40,000 moves for a 20x20 board. This involves dynamically cutting and restitching the Hamiltonian cycle to reach the target quickly .The Hamiltonian Cycle approach, this is not guaranteed to win the game.

Cons: Don't have the standard algorithm must be encoded by hand.

It Requires some familiarity with graph theory. Could be slow for larger game boards.

1.2.2 GENETIC ALGORITHMS

Genetic algorithms are another popular approach to this type of problem. This approach is modeled off of biological evolution and natural selection. A machine learning model perceptual inputs to action outputs.The agent moves are selected randomly. Each instance of a model corresponds to an organism in the natural selection analogy, while the model's parameters correspond to the organism's genes**.**To start, a bunch of random models are initialized in an environment and set loose. Once the Agent can loose, a fitness function selects the best individuals from a given generation. In the case of the fitness function would just pick Agent with the highest scores. A new generation is then bred from the best individuals, with the addition of random mutations . Some of these mutations will hurt, some will not have any effect, and some will be beneficial. Over time, the evolutionary pressure will select for better and better models. To play around with and visualize learning via genetic algorithm, see this tool by Keiwan.

Cons: Can be slow to converge because mutations are random. The performance is dependent on the inputs available to the model.

1.3 PROPOSED SYSTEM

1.3.1 REINFORCEMENT LEARNING

Reinforcement learning is made up of two components which are an environment and an agent. The agent acts as the algorithm ,while the environment acts as the object the agent reacts to. Firstly, the environment will send the agent a state.The agent needs to implement its knowledge to take actions in response to that state. Then, the next state and reward willbe sent back to the agent by the environment. Rather than directly telling the agent about which action to take, the reward feedback will just indicate how valuable some sequence of states and actions . This repeating loop helps the agent to evaluate its last action and update its knowledge by the rewards returned by the environment.Finding the best or the optimal policy is the key to solving RL problems .

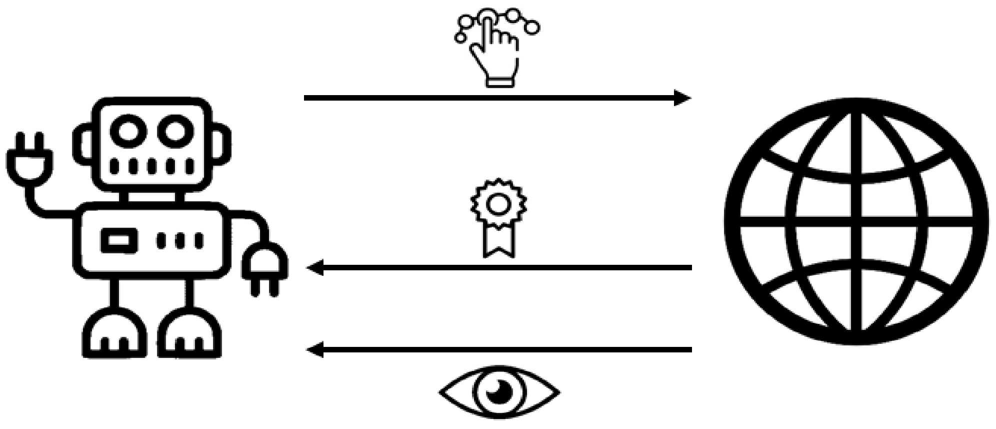

Fig 1:Reinforcement Learning Process

Action (A): moves that the agent can possibly take.

State (S): current situation that is returned by theenvironment.

Reward (R): immediate feedback returned by theenvironment in order to evaluate the last action.

Policy (π): agents' employed strategies in determining theaction that will be taken based on the current state.

Value (V): expected discounted long-term reward.

Vπ (s): expected long-term rewards for the current stateunder current policy π.

Q-value: expected discounted long-term rewards underaction a.

Qπ (s,a): expected long-term rewards for the current state is under policy π by taking action. There are two types of algorithm model which are model-based and model-free.

2.LITERATURE SURVEY

Reinforcement Learning is of great importance in the field of AI. It allows representing mathematical methods for agents to act flawlessly in complex environments. A great deal of work has been done for developing smart self-learning agents for games. However, DQN and DSN are just a couple of available methods which allows us to embed in these agents. Therefore, it is necessary to perform a comprehensive study of the existing works to re-evaluate the issues and the room for advancements, Enhancing Game Development. emphasizes the creation of smart self-learning agents to corroborate the process of game development. These agents inhibiting human-like behaviour can help with the game balancing and assessment, almost identical to the performance of game developers. They bestowed four case studies, each demonstrating a parity between skill and style. Also, they have evaluated multi-faceted theories in this paper with each having their pragmatic implication. In the first two case studies, they have created playtesting agents for mobile games that showed how the observation space could substantially influence the efficacy of the solutions trained with RL. The other two case studies are engrossed towards designing game playing agents that shows how the existing RL models need to be tweaked to perform well on the various environments, even including Atari. All these case studies helped in recognizing the challenges. game designers have to encounter in order to accomplish their desired outcome. Many such intelligent agents are developed for improving games which plays a crucial role in enhancing the gaming experience. It aims to provide a detailed model of a convolutional neural network that rests upon the methods of Q learning. The method is applied to a range of Atari 2600 games where the input is given in the form of raw pixels and the output received is a value function giving an idea of the future rewards. All the Atari 2600 games are in The Arcade Learning Environment . A set of tasks was given to the agents introducing a challenging level for human players. These agents partially observed the tasks as it was impossible to understand the current situation by just analyzing the current screen. For six out of seven games, in which the method is applied, the network has outstripped all the earlier RL algorithms and even outperformed an expert human player on the three of them. Finally, the author was successful in providing a new Deep Learning Model for Reinforcement Learning for agents in the Atari 2600 games without altering the architecture and hyper parameters.

2.1 BLOCK DIAGRAM

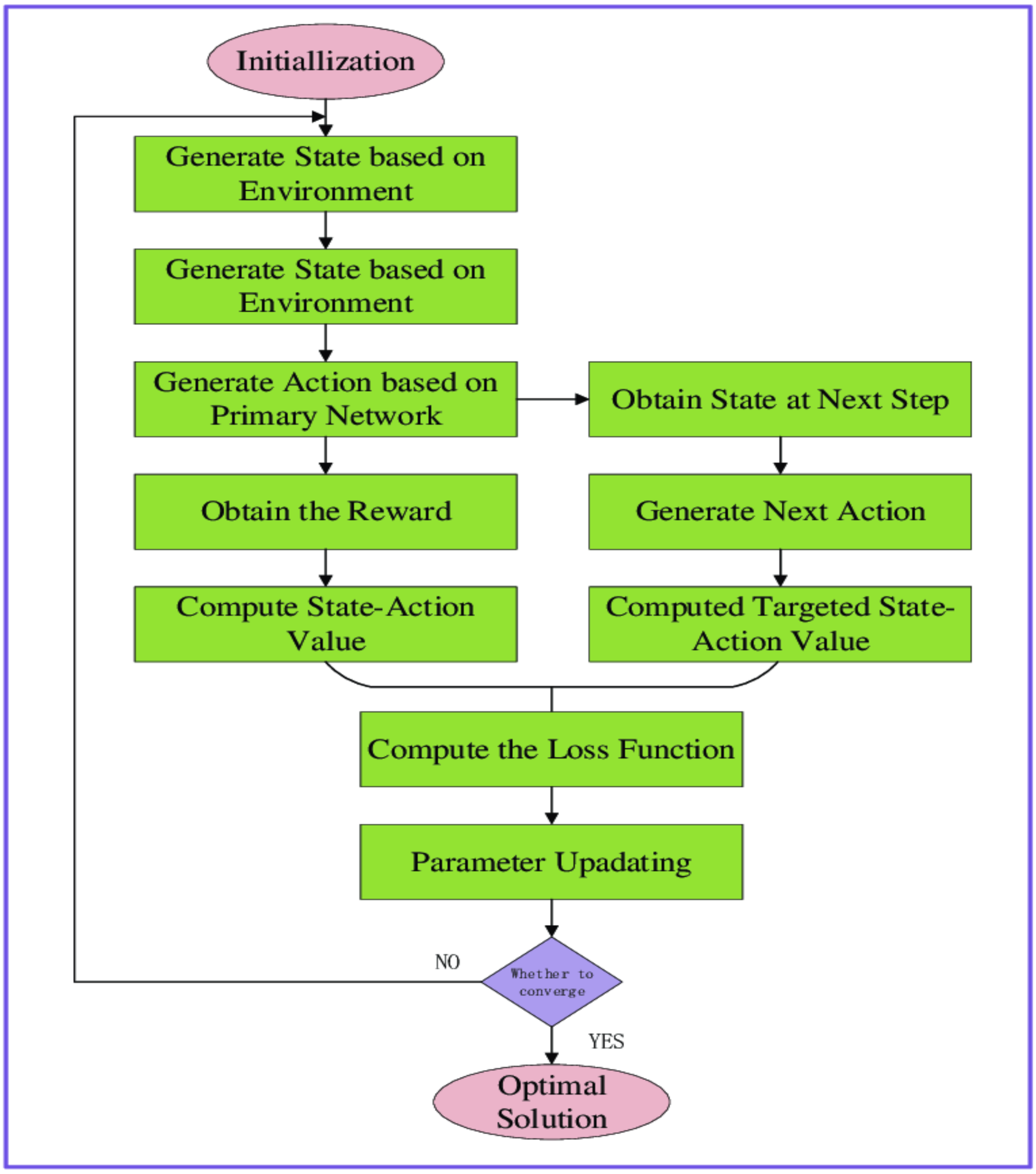
Initiallization
Generate State based on Environment
Generate State based on Environment
Generate Action based on Primary Network
Obtain State at Next Step
Obtain the Reward
Generate Next Action
Compute State-Action Value
Computed Targeted State-Action Value
Compute the Loss Function
Parameter Upadating
NO
Whether to converge
YES
Optimal Solution

Fig 2: Block Diagram of Playing game using Reinforcement Learning

This architecture of the proposed system has the following components:

- Start intilize evnironment,agent and model.
- Run episods or timesteps, save states,actions,reward
- Evaluate observed reward.
- Learn Fit saved data to improve model.

2.2 TURTEL:

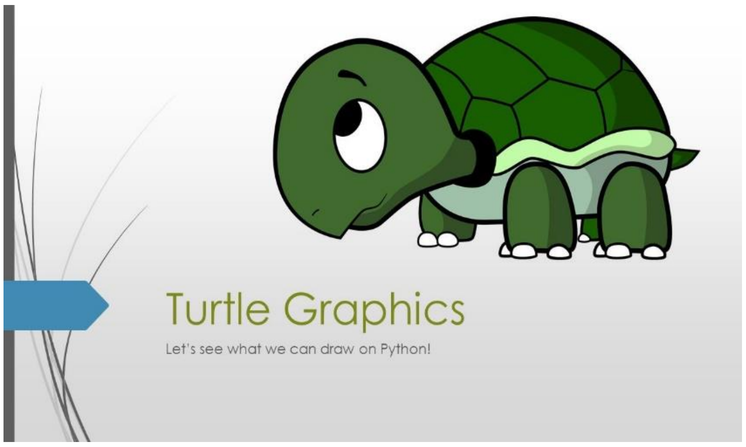

Fig: 3 Turtle Graphics library logo

Turtle is a Python library which used to create graphics, pictures, and games. It was developed by Wally Feurzeig, Seymour Parpet and Cynthina Slolomon in 1967. It was a part of the original Logo programming language. The Logo programming language was popular among the kids because it enables us to draw attractive graphs to the screen in the simple way. It is like a little object on the screen, which can move according to the desired position. Similarly, turtle library comes with the interactive feature that gives the flexibility to work with Python. In this tutorial, we will learn the basic concepts of the turtle library, how to set the turtle up on a computer, programming with the Python turtle library, few important turtle commands, and develop a short but attractive design using the Python turtle library. Turtle is a pre-installed library in Python that is similar to the virtual canvas that we can draw pictures and attractive shapes. It provides the onscreen pen that we can use for drawing. The turtle Library is primarily designed to introduce children to the world of programming. With the help of Turtle's library, new programmers can get an idea of how we can do programming with Python in a fun and interactive way. It is beneficial to the children and for the experienced programmer because it allows designing unique shapes, attractive pictures, and various games. We can also design the mini games and animation. In the upcoming section, we will learn to various functionality of turtle library. turtle is built in library so we don't need to install separately. We just need to import the library into our Python environment.The Python turtle library consists of all important methods and functions that we will need to create our designs and images. Import the turtle library using the following command. Now, we can access all methods and functions. First, we need to create a dedicated window where we carry out each drawing command. We can do it by initializing a variable for it.

2.3 OPEN AI GYM:

Fig 4:Logo of Open AI Gym Library

If you're looking to get started with Reinforcement Learning, the OpenAI gym is undeniably the most popular choice for implementing environments to train your agents. A wide range of environments that are used as benchmarks for proving the efficacy of any new research methodology are implemented in OpenAI Gym, out-of-the-box. Furthermore, OpenAI gym provides an easy API to implement your own environments. The first thing we do is to make sure we have the latest version of gym installed. One can either use conda or pip to install gym. In our case, we'll use pip. he fundamental building block of OpenAI Gym is the Env class. It is a Python class that basically implements a simulator that runs the environment you want to train your agent in. Open AI Gym comes packed with a lot of environments, such as one where you can move a car up a hill, balance a swinging pendulum, score well on Atari games, etc. Gym also provides you with the ability to create custom environments as well. We start with an environment called MountainCar, where the objective is to drive a car up a mountain. The car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single go. Therefore, the only way to succeed is to drive back and forth to build up momentum. he basic structure of the environment is described by the observation_space , action_space attributes of the Gym Env class. The observation_space defines the structure as well as the legitimate values for the observation of the state of the environment. The observation can be different things for different environments. The most common form is a screenshot of the game. There can be other forms of observations as well, such as certain characteristics of the environment described in vector form. Similarly, the Env class also defines an attribute called the action_space, which describes the numerical structure of the legitimate actions that can be applied to the environment.

3.SYSTEM ANALYSIS

3.1 SOFTWARE REQUIREMENTS:

Platform – Windows 11

Software – Pycharm

3.2 HARDWARE REQUIREMENTS:

Processor – Intel® core™ i3 CPU M 380 @2.53GHz

RAM – 4.00GB

Keyboard - 101 keys.

4.DOMAIN

4.1 PYTHON

Fig 5: Logo of Python

Python is a general purpose, dynamic, high level, and interpreted programming languageIt supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures. Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development. Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development. Python supports multiple programming pattern, including object-oriented, imperative, and functional or procedural programming styles. Python is not intended to work in a particular area, such as web programming.

4.2 PYTHON'S FEATURE SET:

The factors that played an important role in molding the final form of the language and are given by

1. Easy to code: Python is very easy to code. Compared to other popular languages like Java and C++, it is easier to code in Python.
2. Easy to read: Being a high-level language, Python code is quite like English. Looking at it, you can tell what the code is supposed to do. Also, since it is dynamically-typed, it mandates indentation. This aids readability.
3. Expressive: Suppose we have two languages A and B, and all programs that can be made in A can be made in B using local transformations. However, there are some programs that can be made in B, but not in A, using local transformations. Then, B is said to be more expressive than A.

4. Free and Open-Source: Firstly, Python is freely available. You can download it from the link. Secondly, it is open source. This means that its source code is available to the public. You can download it, change it, use it, and distribute it. This is called FLOSS Free/Libre and Open Source Software.
5. High- Level: This means that as programmers, we don’t need to remember the system architecture. Nor do we need to manage the memory. This makes it more programmer- friendly and is one of the key python features.
6. Portable: We can take one code and run it on any machine, there is no need to write different code for different machines. This makes Python a portable language.
7. Interpreted: If you are any familiar with languages like C++ or Java, you must first compile it, and then run it. But in Python, there is no need to compile it. Internally, its source code is converted into an immediate form called bytecode. So, all you need to do is to run your Python code without worrying about linking to libraries, and a few other things. By interpreted, we mean the source code is executed line by line, and not all at once. Because of this, it is easier to debug your code. Also, interpreting makes it just slightly slower than Java, but that does not matter compared to the benefits it has to offer.
8. Object-Oriented: A programming language that can model the real world is said to be object-oriented. It focuses on objects and combines data and functions. Contrarily, a procedure-oriented language revolves around functions, which are code that can be reused. Python supports both procedure-oriented and object-oriented programming whichis one of the key python features. It also supports multiple inheritance, unlike Java. A class is a blueprint for such an object. It is an abstract data type and holds no values.
9. Extensible: If needed, you can write some of your Python code in other languages like C++. This makes Python an extensible language, meaning that it can be extended to otherlanguages.
10. Embeddable: We just saw that we can put code in other languages in our Python source code. However, it is also possible to put our Python code in a source code in a different language like C++s.This allows us to integrate scripting capabilities into our program of the other language.
11. Large Standard Library: Python downloads with a large library that you can use so you don’t have to write your own code for every single thing. There are libraries for regular expressions, documentation-generation, unit-testing, web browsers, threading.

4.3 PYCHARM

PyCharm is one of the most popular Python IDEs. There is a multitude of reasons for this, including the fact that it is developed by JetBrains, the developer behind the popular IntelliJ IDEA IDE that is one of the big 3 of Java IDEs and the "smartest JavaScript IDE" WebStorm. Having the support for web development by leveraging Django is yet another credible reason.

Available as a cross-platform application, PyCharm is compatible with Linux, macOS, and Windows platforms. Sitting gracefully among the best Python IDEs, PyCharm provides support for both Python 2 (2.7) and Python 3 (3.5 and above) versions.

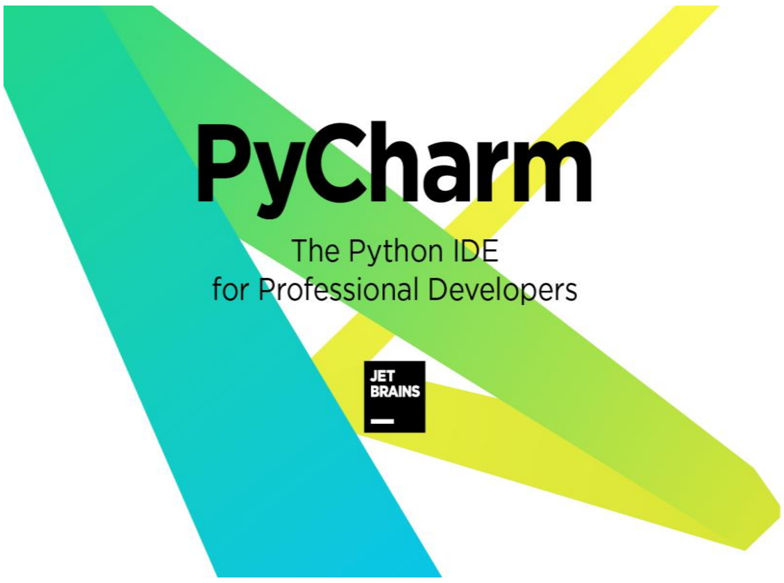

Fig 6: Logo of Pycharm

PyCharm comes with a plethora of modules, packages, and tools to hasten Python development while cutting-down the effort required to do the same to a great extent, simultaneously. Further, PyCharm can be customized as per the development requirements, and personal preferences call for. It was released to the public for the very first time back in February of 2010. In addition to offering code analysis, PyCharm features:

- A graphical debugger
- An integrated unit tester
- Integration support for version control systems (VCSs)
- Support for data science with Anaconda

5.MODULE DESIGN

5.1 CREATING CUSTOM ENVIRONMENT:

OpenAI Gym comes packed with a lot of awesome environments, ranging from environments featuring classic control tasks to ones that let you train your agents to play Atari games like Breakout, Pacman, and Seaquest. However, you may still have a task at hand that necessitates the creation of a custom environment that is not a part of the Gym package. Thankfully, Gym is flexible enough to allow you to do. The environment that we are creating is basically a game that is heavily inspired by the Dino Run game, the one which you play in Google Chrome if you are disconnected from the Internet. There is a dinosaur, and you have to jump over cacti and avoid hitting birds. The distance you cover is representative of the reward you end up getting.The very first consideration while designing an environment is to decide what sort of observation space and action space we will be using. The observation space can be either continuous or discrete.

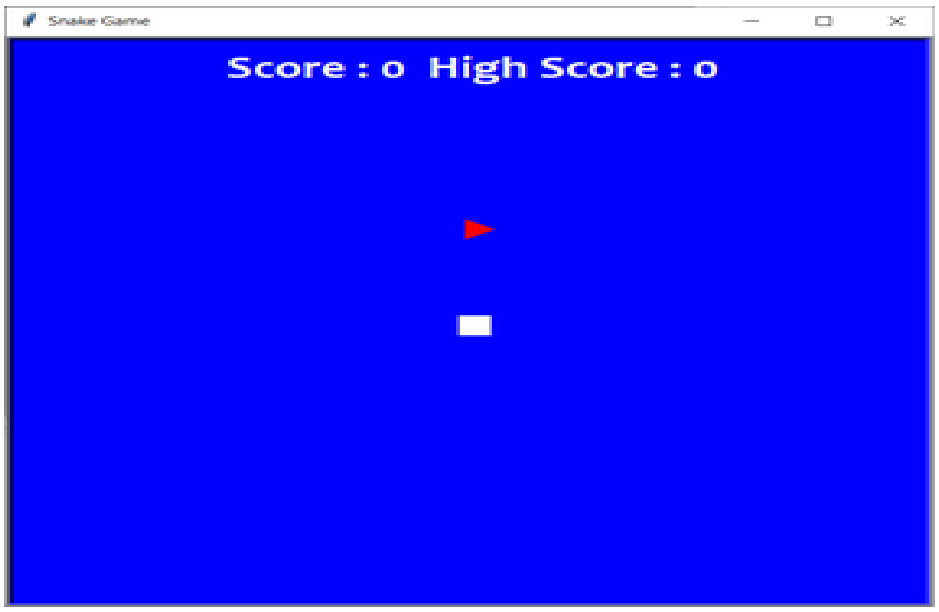

Fig 7: Creating Custom Environment

An example of a discrete action space is that of a grid-world where the observation space is defined by cells, and the agent could be inside one of those cells. An example of a continuous action space is one where the position of the agent is described by real-valued coordinates.The action space can be either continuous or discrete as well. An example of a discrete space is one where each action corresponds to the particular behavior of the agent, but that behavior cannot be quantified. An example of this is Mario Bros, where each action would lead to moving left, right, jumping, etc. Your actions can't quantify the behavior being produced, you can jump but not jump high, higher, or lower. However, in a game like Angry Birds, you decide how much to stretch the slingshot. When we reset our environment, we need to reset all the state-based variables in our environment. These include things like fuel consumed, episodic return, and the elements present inside the environment.

Now that we have the reset function out of the way, we begin work on implementing the step function, which will contain the code to transition our environment from one state to the next given an action. In many ways, this section is the proverbial meat of our environment, and this is where most of the planning goes.

We first need to enlist things that need to happen in one transition step of the environment. This can be basically broken down into two parts:

1. Applying actions to our agent.
2. Everything else that happens in the environments, such as behaviour of the non-RL actors.

5.2 PLAYING WITH STATE SPACE:

The state space S is a set of all the states that the agent can transition to and action space A is a set of all actions the agent can act out in a certain environment. There are also Partial Observable cases, where the agent is unable to observe the complete state information of the environment. The agent learns to play agent (with experience replay), but maybe it’s possible to change the state space and achieve similar or better performance. State space ‘no direction’: don’t give the agent the direction is going. State space ‘coordinates’: replace the location of the target. The coordinates are scaled between 0 and 1. State space ‘direction 0 or 1’: the original state space. State space : don’t tell the agent when the body is up, right, down or left, only tell it if there’s a dangerous.

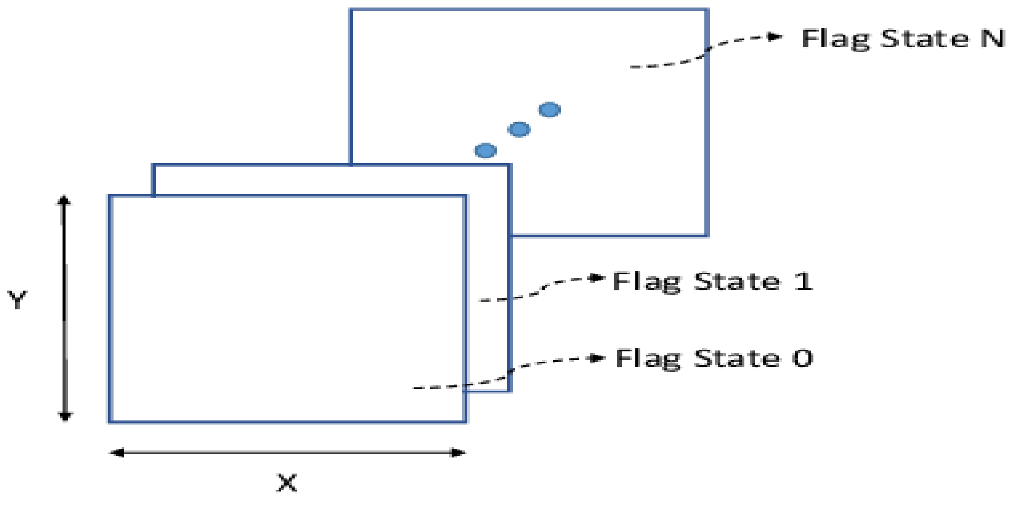

Fig 8: State Space of Environment

5.3 EXPERIENCE REPLAY:

Starting with the firs time step, the Experience Replay starts the training data generation phase and uses the Q Network to select an ε-greedy action. The Q Network acts as the agent while interacting with the environment to generate a training sample. No DQN training happens during this phase. The Q Network predicts the Q-values of all actions that can be taken from the current state. We use those Q-values to select an ε-greedy action.

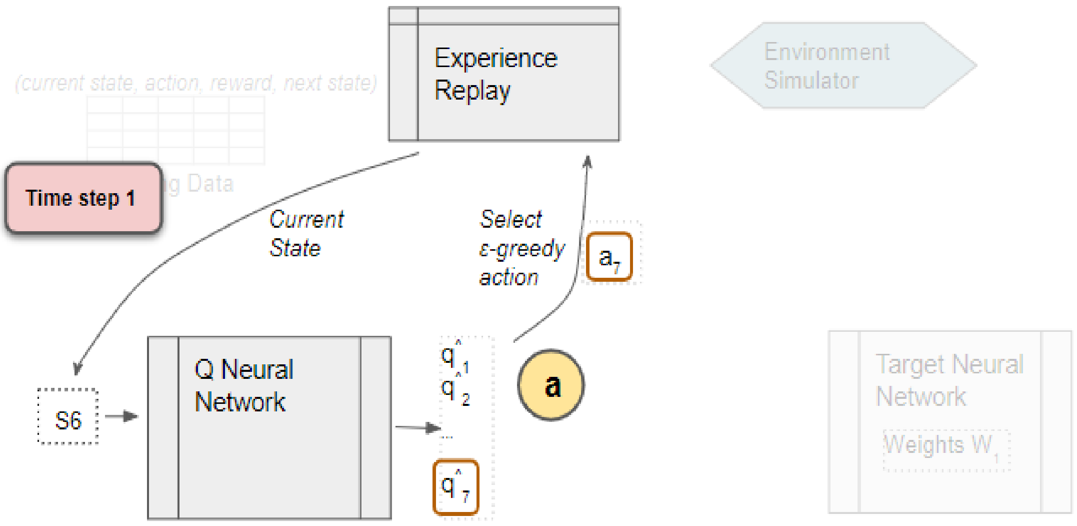

Fig 9: Experience Replay executes the ε-greedy action and receives the next state and reward.

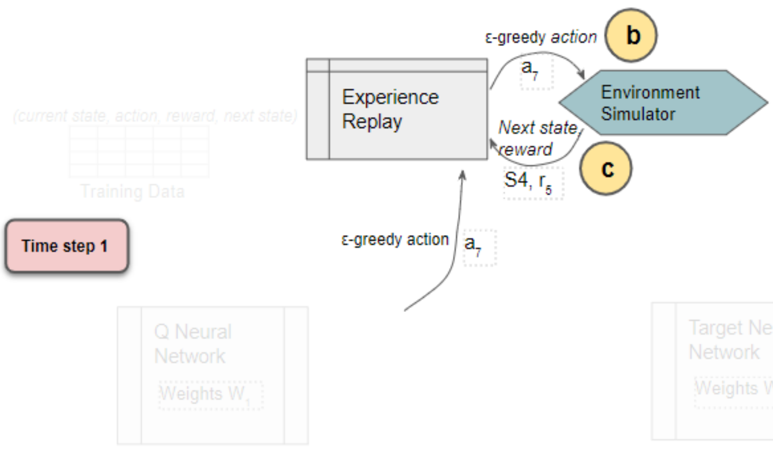

Fig 10: Storing in Replay Memory

It stores the results in the replay data. Each such result is a sample observation which will later be used as training data. The Replay Buffer is a finite-sized array, in which experiences are over-written with newer ones after a certain amount of time. We do not need to store the entire history of experiences across the agent's lifetime. If the buffer is too large, it would "over-replay" early experiences that had poor performance.

5.4 DEEP Q NETWORK

5.4.1 TARGET NETWORK:

The strategy for learning in DQN is to transform the problem into that of supervised learning. “But we do not have any labels”, you might wonder. Yes, you are right. But this is where the recursive nature of the Bellman equation for Q-values helps us. As per the Bellman equation,

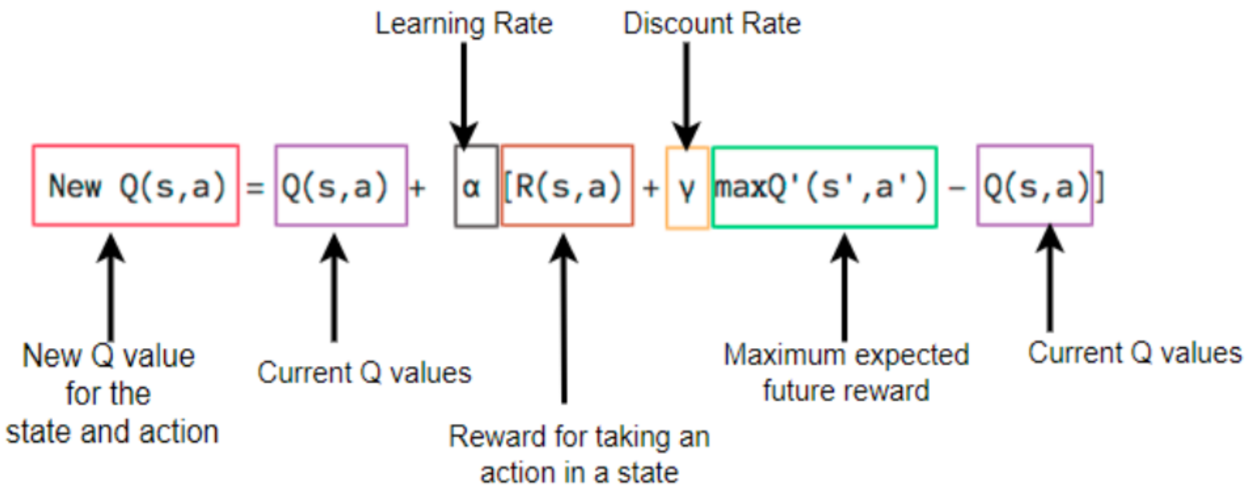
Learning Rate
Discount Rate
New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
New Q value for the state and action
Current Q values
Reward for taking an action in a state
Maximum expected future reward
Current Q values

Fig 11: BellMan equation

Thus, the idea is to use the right side of the equation above as our ‘true label’ and the left side of the equation as the ‘predicted label’. Similar to any supervised learning approach then, the aim is to bring ‘predicted label’ as close as possible to ‘true label’.

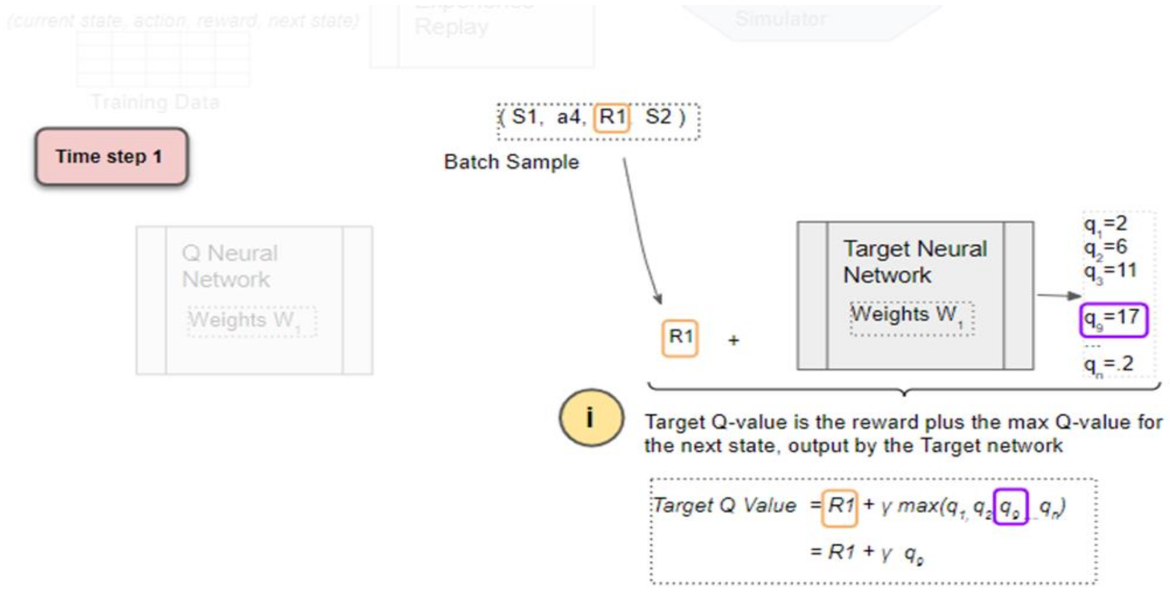
Time step 1
(S1, a4, R1, S2)
Batch Sample
Q Neural Network
Weights W1
Target Neural Network
Weights W1
R1 +
q1=2
q2=6
q3=11
q9=17
qn=2
Target Q-value is the reward plus the max Q-value for the next state, output by the Target network
Target Q Value = R1 + γ max(q1, q2, q9, qn)
= R1 + γ q9

Fig 12: Finding Maximum Reward

However, this naive implementation poses a problem. Since the ‘true label’ (hereafter referred to as the TD-Target) and the ‘predicted label’ are estimated from the same network, as the network improves the TD-Target also shifts. The effect of this is similar to a cat crazily chasing after a laser pointer. To combat this, the authors introduce Target Networks . The idea is to have two copies of the Q-network, one is kept current by updating it at every time step, while the other is kept frozen for a set number of steps. This ensures that the target is not continuously moving. After every ‘n’ gradient update steps, the target network is updated by copying the parameters of the current network into the target. This modification was found to help improve the stability during training by reducing oscillations and divergence of the policy.

5.4.2 Q NETWORK:

Q-Learning replaces the regular Q-table with a neural network. Rather than mapping a state-action pair to a q-value, a neural network maps input states to action, Q-value pairs. Q-value, mathematically denoted as Q(s, a) is a measure of the “quality” or “goodness” of a particular state-action pair. More concretely, it represents the expected return of performing action aa in state ss and then following the policy ππ. Computing the Q-value requires both state and action information . However, DQN only takes the state vector as input. The output layer of the network then has the same shape as the action space of the environment, thus outputting the Q-values of each of the actions ,this setup is the reason why DQN cannot be applied to a continuous action space. The agent then acts by picking the action with the highest Q-value.

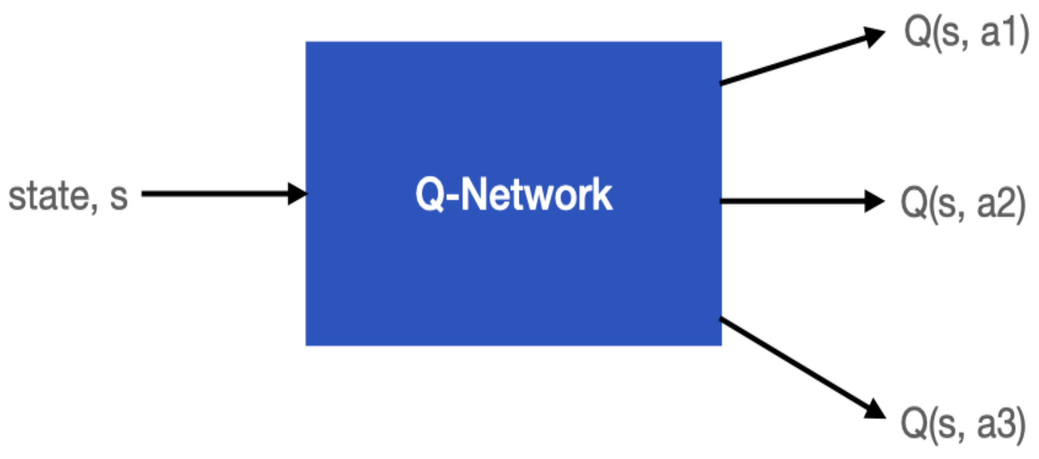

Fig 13: Architecture of Q Network

As for the loss computation, mean-squared-error is applied to the TD-Target and the ‘predicted y’ on a mini-batch of past experiences sampled from the Replay Buffer. Each of the experiences in the batch contains information about <state, action, next state, reward, done>. The ‘state’ and the 'action' from the sampled batch of experiences are used to compute the 'Predicted Q' using the Q-network. The 'next state' and the 'reward' from the sampled batch are involved in computing the 'Target Q' via the Target Network.

COMPUTE LOSS:

Back-propagate the loss and update the weights of the Q Network using gradient descent. The Target network is not trained and remains fixed, so no Loss is computed, and back-propagation is not done. This completes the processing for this time-step. The processing repeats for the next time-step. The Q network weights have been updated but not the Target network's. This allows the Q network to learn to predict more accurate Q values, while the target Q values remain fixed for a while, so we are not chasing a moving target. After T time-steps, copy the Q network weights to the Target network. This lets the Target network get the improved weights so that it can also predict more accurate Q values. Processing continues as before. that Q-Learning used the Target Q Value, the Current Q Value, and observed reward to update the Current Q Value using its update equation. The DQN works in a similar way. Since it is a neural network, it uses a Loss function rather than an equation. It also uses the Predicted and Current Q Value, Target Q Value, and observed reward to compute the Loss to train the network, and thus improve its predictions.

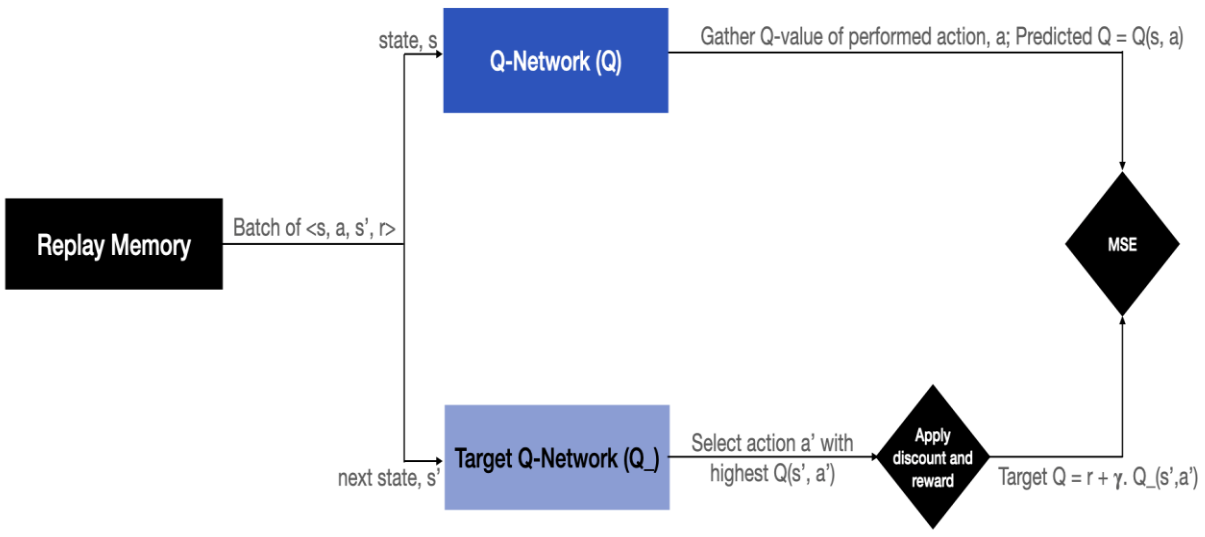
state, s
Q-Network (Q)
Gather Q-value of performed action, a; Predicted Q = Q(s, a)
Replay Memory
Batch of <s, a, s', r>
MSE
next state, s'
Target Q-Network (Q_)
Select action a' with highest Q(s', a')
Apply discount and reward
Target Q = r + γ. Q_(s',a')

Fig 14: Computing Loss

6.CODING

6.1 CREATING ENVIRONMENT

```
import turtle
import random
import time
import math
import gym
from gym.utils import seeding
HEIGHT=20
WIDTH=20
PIXEL_H=20* HEIGHT
PIXEL_W=20* WIDTH
SLEEP=0.2
GAME_TITLE='Snake Game'
BG_COLOR='cyan'
SNAKE_SHAPE='square'
SNAKE_COLOR='blue'
SNAKE_START_LOC_H=0
SNAKE_START_LOC_V=0
APPLE_SHAPE='circle'
APPLE_COLOR='green'
class Snake(gym.Env):
    def __init__(self,human=False,env_info={'state_space':None}):
        super(Snake,self).__init__()
        self.done=False
        self.seed()
        self.reward=0
        self.action_space=4
        self.state_space=12
        self.total, self.maximum=0,0
        self.human=human
        self.env_info=env_info
        self.win=turtle.Screen()
        self.win.title(GAME_TITLE)
        self.win.bgcolor(BG_COLOR)
        self.win.tracer(0)
        self.win.setup(width=PIXEL_W+20, height=PIXEL_H+20)
        self.snake = turtle.Turtle()
        self.snake.shape(SNAKE_SHAPE)
        self.snake.speed(0)
        self.snake.penup()
        self.snake.color(SNAKE_COLOR)
        self.snake.goto(SNAKE_START_LOC_H,SNAKE_START_LOC_V)
        self.snake.direction='stop’
        self.snake_body=[]
        self.add_to_body()
        self.apple=turtle.Turtle()
        self.apple.speed(0)
        self.apple.shape(APPLE_SHAPE)
        self.apple.color(APPLE_COLOR)
        self.apple.penup()
        self.move_apple(first=True)
        self.dist=math.sqrt((self.snake.xcor()-self.apple.xcor())**2+(self.snake.ycor()-
self.apple.ycor()) ** 2)
        #score
        self.score=turtle.Turtle()
        self.score.speed(0)
        self.score.color('black')
        self.score.penup()
        self.score.hideturtle()
        self.score.goto(0, 100)

self.score.write(f"Total:{self.total}Highest:{self.maximum}",align='Center',font=('Courier
',15,'bold'))
        #control
        self.win.listen()
        self.win.onkey(self.go_up,'Up')
        self.win.onkey(self.go_right,'Right')
        self.win.onkey(self.go_down,'Down')
        self.win.onkey(self.go_left,'Left')
    def seed(self,seed=None):
        self.np_random,seed=seeding.np_random(seed)
        return [seed]
    def random_coordinates(self):
        apple_x=random.randint(-WIDTH/2,WIDTH/2)
        apple_y=random.randint(-HEIGHT/2,HEIGHT/2)
        return apple_x, apple_y
    def move_snake(self):
        if self.snake.direction=='stop':
            self.reward=0
        if self.snake.direction=='up':
            y=self.snake.ycor()
            self.snake.sety(y+20)
        if self.snake.direction=='right':
            x=self.snake.xcor()
            self.snake.setx(x+20)
        if self.snake.direction=='down':
            y=self.snake.ycor()
            self.snake.sety(y-20)
        if self.snake.direction=='left':
            x=self.snake.xcor()
            self.snake.setx(x-20)
    def go_up(self):
        if self.snake.direction!="down":
            self.snake.direction="up"
    def go_down(self):
        if self.snake.direction!="up":
            self.snake.direction="down"
    def go_right(self):
        if self.snake.direction!="left":
            self.snake.direction="right"
    def go_left(self):
        if self.snake.direction!="right":
            self.snake.direction="left"
    def move_apple(self,first=False):
        if first or self.snake.distance(self.apple)<20:
            while True:
                self.apple.x,self.apple.y=self.random_coordinates()
                self.apple.goto(round(self.apple.x*20),round(self.apple.y*20))
                if not self.body_check_apple():
                    break
            if not first:
                self.update_score()
                self.add_to_body()
            first = False
            return True
    def update_score(self):
        self.total+=1
        if self.total>=self.maximum:
            self.maximum=self.total
        self.score.clear()
self.score.write(f"Total:{self.total}Highest:{self.maximum}",align='Center',font=('Courier
',15,'bold'))
    def reset_score(self):
        self.score.clear()
        self.total=0
self.score.write(f"Total:{self.total}Highest:{self.maximum}",align='Center',font=('Courier
',15,'bold'))
    def add_to_body(self):
        body=turtle.Turtle()
        body.speed(0)
        body.shape('square')
        body.color('blue')
        body.penup()
        self.snake_body.append(body)
    def move_snakebody(self):
        if len(self.snake_body) > 0:
            for index in range(len(self.snake_body)-1,0,-1):
                x=self.snake_body[index-1].xcor()
                y=self.snake_body[index-1].ycor()
                self.snake_body[index].goto(x,y)
            self.snake_body[0].goto(self.snake.xcor(),self.snake.ycor())
    def measure_distance(self):
        self.prev_dist=self.dist
        self.dist=math.sqrt(
            (self.snake.xcor()-self.apple.xcor())**2+(self.snake.ycor()-self.apple.ycor())**2)
    def body_check_snake(self):
        if len(self.snake_body)>1:
            for body in self.snake_body[1:]:
                if body.distance(self.snake)<20:
                    self.reset_score()
                    return True
    def body_check_apple(self):
        if len(self.snake_body)>0:
            for body in self.snake_body[:]:
                if body.distance(self.apple)<20:
                    return True
    def wall_check(self):
        if self.snake.xcor()>200 or self.snake.xcor()<-200 or self.snake.ycor()>200 or 
self.snake.ycor()<-200:
            self.reset_score()
            return True
    def reset(self):
        if self.human:
            time.sleep(1)
        for body in self.snake_body:
            body.goto(1000,1000)
        self.snake_body=[]
        self.snake.goto(SNAKE_START_LOC_H,SNAKE_START_LOC_V)
        self.snake.direction='stop'
        self.reward=0
        self.total=0
        self.done=False
        state=self.get_state()
        return state
    def run_game(self):
        reward_given=False
        self.win.update()
        self.move_snake()
        if self.move_apple():
            self.reward=10
            reward_given=True
        self.move_snakebody()
        self.measure_distance()
        if self.body_check_snake():
            self.reward=-100
            reward_given=True
            self.done=True
            if self.human:
                self.reset()
        if self.wall_check():
            self.reward=-100
            reward_given=True
            self.done=True
            if self.human:
                self.reset()
        if not reward_given:
            if self.dist< self.prev_dist:
                self.reward=1
            else:
                self.reward=-1
        if self.human:
            time.sleep(SLEEP)
            state=self.get_state()
    def step(self,action):
        if action==0:
            self.go_up()
        if action==1:
            self.go_right()
        if action==2:
            self.go_down()
        if action==3:
            self.go_left()
        self.run_game()
        state=self.get_state()
        return state,self.reward,self.done,{}
    def get_state(self):
        self.snake.x,self.snake.y=self.snake.xcor()/WIDTH,self.snake.ycor()/HEIGHT
```

```
self.snake.xsc,self.snake.ysc=self.snake.x/WIDTH+0.5,self.snake.y/HEIGHT+0.5
self.apple.xsc,self.apple.ysc=self.apple.x/WIDTH+ 0.5,self.apple.y/HEIGHT+0.5
if self.snake.y>=HEIGHT/2:
    wall_up,wall_down=1,0
elif self.snake.y<=-HEIGHT/2:
    wall_up,wall_down=0,1
else:
    wall_up,wall_down=0,0
if self.snake.x>=WIDTH/2:
    wall_right,wall_left=1,0
elif self.snake.x<=-WIDTH/2:
    wall_right,wall_left=0,1
else:
    wall_right,wall_left=0,0
body_up=[]
body_right=[]
body_down=[]
body_left=[]
if len(self.snake_body)>3:
    for body in self.snake_body[3:]:
        if body.distance(self.snake)==20:
            if body.ycor()<self.snake.ycor():
                body_down.append(1)
            elif body.ycor( )>self.snake.ycor():
                body_up.append(1)
            if body.xcor()<self.snake.xcor():
                body_left.append(1)
            elif body.xcor()>self.snake.xcor():
                body_right.append(1)
if len(body_up)>0:
    body_up=1
else:
    body_up=0
if len(body_right)>0:
    body_right=1
else:
    body_right=0
if len(body_down)>0:
    body_down=1
else:
    body_down=0
if len(body_left)>0:
    body_left=1
else:
```

```
            body_left=0
        if self.env_info['state_space']=='coordinates':
            state = [self.apple.xsc, self.apple.ysc, self.snake.xsc, self.snake.ysc,\
                    int(wall_up or body_up),int(wall_right or body_right),int(wall_down or
body_down),int(wall_left or body_left), \
                    int(self.snake.direction=='up'),int(self.snake.direction=='right'),
                    int(self.snake.direction=='down'),int(self.snake.direction=='left')]
        elif self.env_info['state_space']=='no direction':
            state = [int(self.snake.y<self.apple.y),int(self.snake.x<self.apple.x),
                    int(self.snake.y>self.apple.y),int(self.snake.x>self.apple.x), \
                    int(wall_up or body_up),int(wall_right or body_right), int(wall_down or
body_down),int(wall_left or body_left), \
                    0, 0, 0, 0]
        elif self.env_info['state_space']=='no body knowledge':
            state = [int(self.snake.y<self.apple.y),int(self.snake.x<self.apple.x),
                    int(self.snake.y>self.apple.y),int(self.snake.x>self.apple.x), \
                    wall_up,wall_right,wall_down,wall_left,\
                    int(self.snake.direction=='up'),int(self.snake.direction=='right'),
                    int(self.snake.direction=='down'),int(self.snake.direction=='left')]
        else:
            state = [int(self.snake.y<self.apple.y),int(self.snake.x<self.apple.x),
                    int(self.snake.y>self.apple.y),int(self.snake.x>self.apple.x),\
                    int(wall_up or body_up),int(wall_right or body_right),int(wall_down or
body_down),
                    int(wall_left or body_left),\
                    int(self.snake.direction=='up'),int(self.snake.direction=='right'),
                    int(self.snake.direction=='down'),int(self.snake.direction=='left')]
        return state
    def bye(self):
        self.win.bye()
if __name__=='__main__':
    human=True
    env=Snake(human=human)
    if human:
        while True:
 env.run_game()
```

6.2 DQN ALGORITHM

```
from PROJECT.Snake import Snake
import random
import numpy as np
from keras import Sequential
from collections import deque
from keras.layers import Dense
from keras.optimizers import Adam
class DQN:
    def __init__(self,env,params):
        self.action_space=env.action_space
        self.state_space=env.state_space
        self.epsilon=params['epsilon']
        self.gamma=params['gamma']
        self.batch_size=params['batch_size']
        self.epsilon_min=params['epsilon_min']
        self.epsilon_decay=params['epsilon_decay']
        self.learning_rate=params['learning_rate']
        self.layer_sizes=params['layer_sizes']
        self.memory=deque(maxlen=2500)
        self.model=self.build_model()
    def build_model(self):
        model=Sequential()
        for i in range(len(self.layer_sizes)):
            if i==0:
model.add(Dense(self.layer_sizes[i],input_shape=(self.state_space,),activation='relu'))
            else:
                model.add(Dense(self.layer_sizes[i],activation='relu'))
        model.add(Dense(self.action_space, activation='softmax'))
        model.compile(loss='mse',optimizer=Adam(lr=self.learning_rate))
        return model
    def remember(self,state,action,reward,next_state,done):
        self.memory.append((state,action,reward,next_state,done))
    def act(self,state):
        if np.random.rand()<=self.epsilon:
            return random.randrange(self.action_space)
        act_values=self.model.predict(state)
        return np.argmax(act_values[0])
    def replay(self):
        if len(self.memory)<self.batch_size:
            return
        minibatch=random.sample(self.memory,self.batch_size)
        states=np.array([i[0] for i in minibatch])
        actions=np.array([i[1] for i in minibatch])
        rewards=np.array([i[2] for i in minibatch])
        next_states=np.array([i[3] for i in minibatch])
        dones=np.array([i[4] for i in minibatch])
        states=np.squeeze(states)
        next_states=np.squeeze(next_states)
        targets=rewards+self.gamma*(np.amax(self.model.predict_on_batch(next_states),axis=1))
* (1 - dones)
        targets_full=self.model.predict_on_batch(states)
        ind=np.array([i for i in range(self.batch_size)])
        targets_full[[ind],[actions]]=targets
        self.model.fit(states,targets_full,epochs=1,verbose=0)
        if self.epsilon>self.epsilon_min:
            self.epsilon*= self.epsilon_decay
def train_dqn(episode,env):
    sum_of_rewards=[]
    agent=DQN(env,params)
    for e in range(episode):
        state=env.reset()
        state=np.reshape(state,(1,env.state_space))
        score=0
        max_steps=10000
        for i in range(max_steps):
            action=agent.act(state)
            prev_state=state
            next_state,reward,done,_=env.step(action)
            score+=reward
            next_state=np.reshape(next_state,(1,env.state_space))
            agent.remember(state,action,reward,next_state,done)
            state=next_state
            if params['batch_size']>1:
                agent.replay()
            if done:
                print(f'final state before dying:{str(prev_state)}')
                print(f'episode:{e+1}/{episode},score:{score}')
                break
        sum_of_rewards.append(score)
    return sum_of_rewards
if __name__=='__main__':
    params=dict()
    params['name']=None
    params['epsilon']=1
    params['gamma']=.95
    params['batch_size']=500
    params['epsilon_min']=.01
    params['epsilon_decay']=.995
    params['learning_rate']=0.00025
    params['layer_sizes']=[128,128,128]
    results=dict()
    ep=50
    env_infos={'States:only walls':{'state_space':'no body knowledge'},'States: direction 0 or 1':
{'state_space': ''}, 'States: coordinates': {'state_space': 'coordinates'},'States: no direction':
{'state_space': 'no direction'}}
    env=Snake()
    sum_of_rewards=train_dqn(ep, env)
    results[params['name']]=sum_of_rewards
```

8.OUTPUT SCREENS

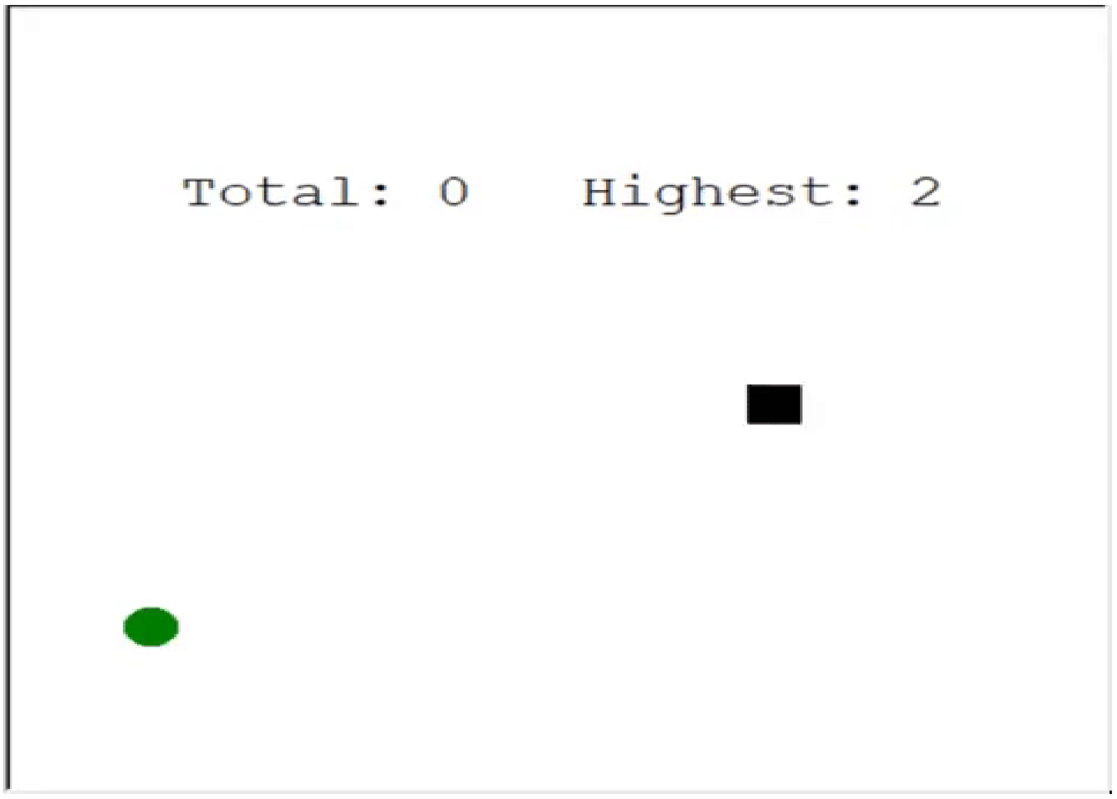

Fig 15: Training of Agent with Random moves

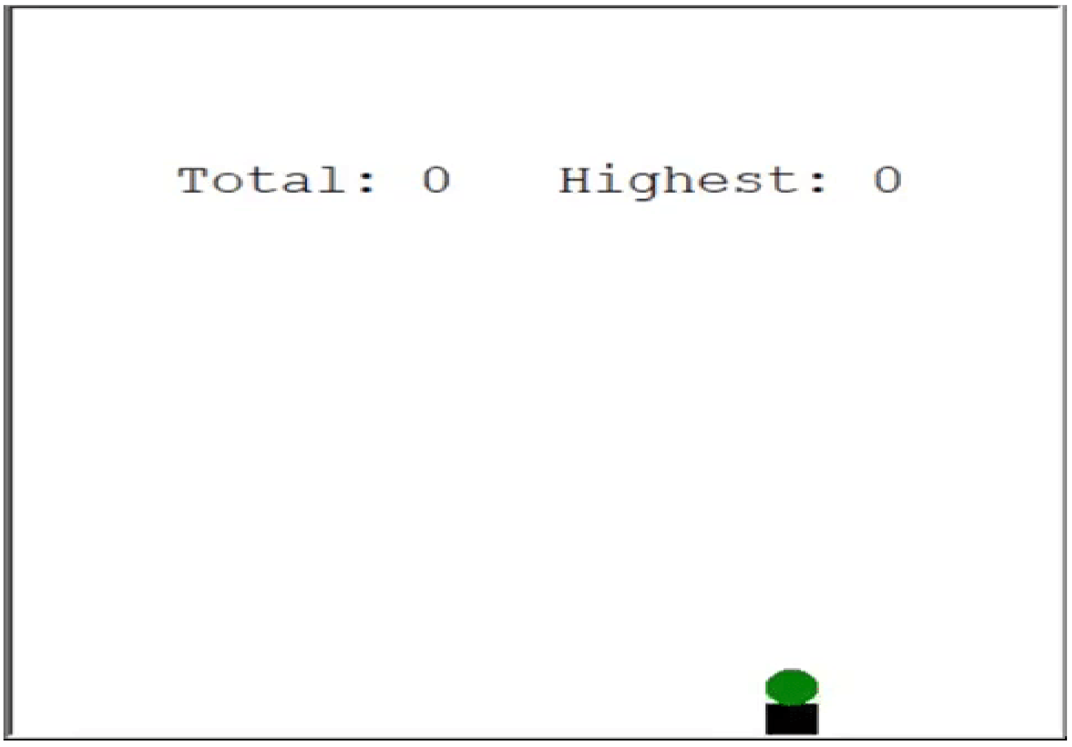

Fig 16: Testing Agent

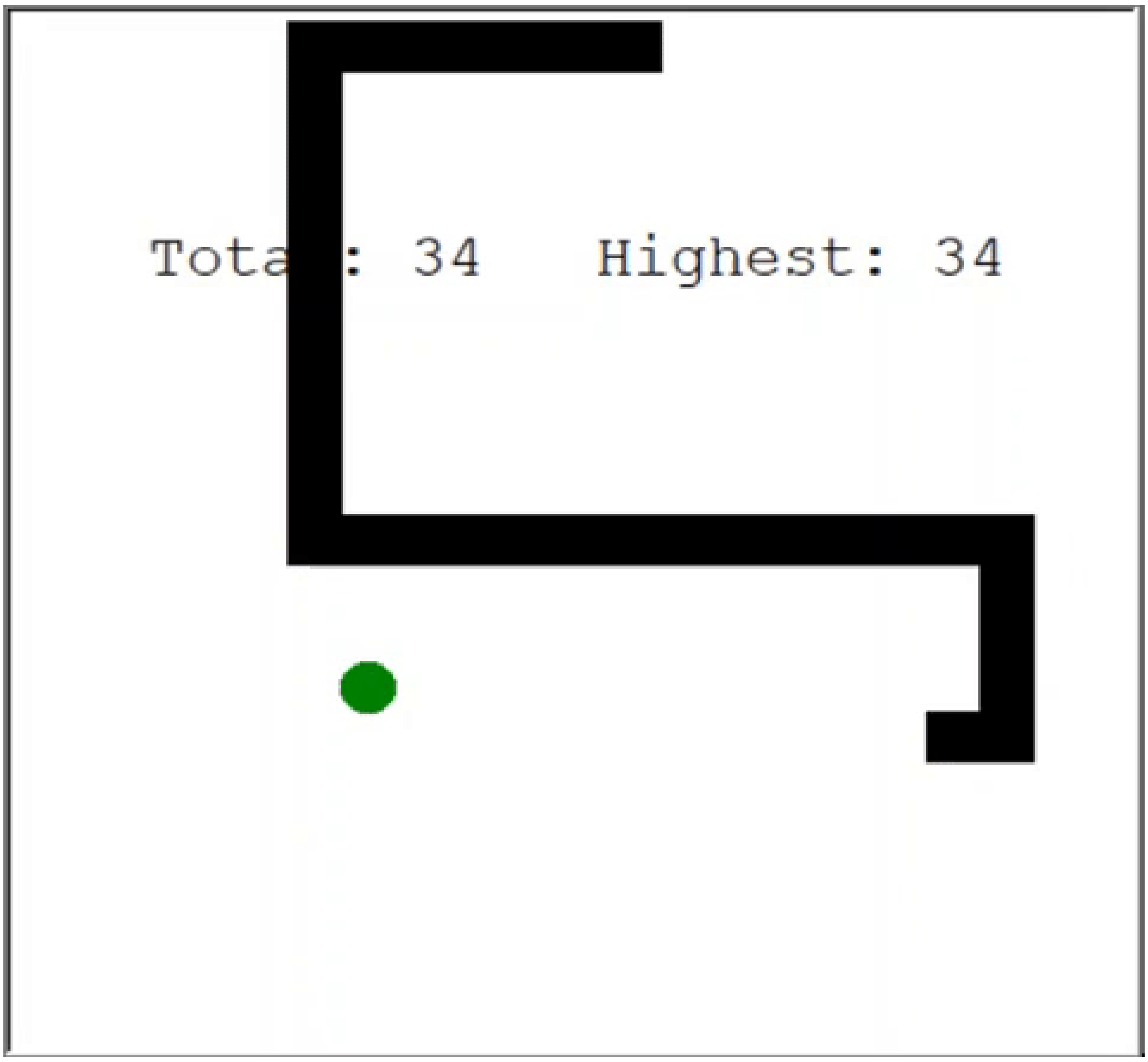

Fig 17: Playing Agent itself

9.CONCLUSION

Reinforcement Learning has a lot of potential. Game playing is one small thing that RL is good at and it has been explored extensively in recent years. Recently, a new toolkit called NLPGym has been released which gives the user an environment where they can test Reinforcement Learning on Natural Language Processing tasks. I would love to explore this myself and I am actually doing an Independent Research with Matthew Richey on this next semester, which I really look forward to. Future work building on this project might include accomplishing tasks that have a bigger action and state spaces and possibly replicating some of more complicated papers like Grandmaster level in StarCraft II using multi-agent reinforcement learning by Vinyals et. al. Future of reinforcement learning is very bright and I can see these techniques being applied in a lot of fields . An interesting upgrade might be obtained passing screenshots of the current game for each iteration.The Deep Q-Learning model can be replaced with a Double Deep Q-learning algorithm, for a more precise convergence.As a conclusion, implementing machine learning in a game can help players in making better decisions, thus, improve their performance. Implementing machine learning agent on the opponent AI will also help in making the game more competitive, thus, improve the overall fun of the game.The followings are the remarks and suggestions for future works based on our review:

1. Promote a system that implements more than one machine learning algorithm based on a certain task.
2. Promote an algorithm which can speed up the learning rate without costing the obtainable rewards.
3. Promote more machine learning implementation on games with imperfect information.
4. Promote the use of reinforcement learning in different areas apart from games.

10.BIBILOGRAGHY

1. Introduction to Various Reinforcement Learning Algorithms.. Towards Data Science. from https://towardsdatascience.com/introduction-to-various- reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg- 72a5e0cb6287
2. https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
3. Rajat H., (2018). Choosing the Right Machine Learning Algorithm. Hackernoon. Retrieved from https://hackernoon.com/choosing-the- right-machine-learning-algorithm-68126944ce1f
4. https://www.kaggle.com/learn/intro-to-game-ai-and-reinforcement-learning
5. Abolfazl R. (2018). How to choose machine learning algorithms?. Medium. Retrieved from https://medium.com/@aravanshad/how-to-choose-machine-learning-algorithms9a92a448e0df
6. Daniil K. (2017). Machine Learning Algorithms: Which One to Choose for Your Problem. Stats and Bots. Retrieved from https://blog.statsbot.co/machine-learning-algorithms-183cc73197c
7. Model Free Reinforcement learning algorithms. Medium. Retrieved fromhttps://medium.com/deep-math-machine-learning-ai/ch-12-1- model-free-reinforcement-learning-algorithms-monte-carlo-sarsa-q- learning-65267cb8d1b4
8. Robot Control in Human Environment Using Deep Reinforcement Learning, https://ieeexplore.ieee.org/document/8961517