**A Practical activity Report submitted**


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,**

**(A DEEMED TO BE UNIVERSITY),**

**PATIALA, PUNJAB**

**INDIA**

## Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. RL is teaching a software agent how to behave in an environment by telling it how good it's doing.
The first question arises in mind that why we are using reinforcement learning instead of supervised machine learning, the answer is, in
supervised ML algorithms need to be trained with an input and a "correct answer" called target.In this example, we don't know what the best
action to take at each stage of the game is, so a traditional approach would not be effective.

In Reinforcement Learning, we have two main components: the environment (our game) and the agent (our Snake.. or to be correct, the Deep
Neural Network that drives our Snake's actions). Every time the agent performs an action, the environment gives a reward to the agent, which
can be positive or negative depending on how good the action was from that specific state.

Deep Reinforcement Learning (DRL) combines the above ideas of RL with deep neural networks. The neural network learns the "Q function", which takes as input the current environment state and outputs a vector containing expected rewards for each possible action. The agent can then pick the action that maximizes the Q function. Based on this action, the game then updates the environment to a new state and assigns a reward (e.g. +10 for eating an apple, -10 for hitting a wall). At the beginning of training, the Q function is just approximated by a randomly initialized neural network.

**(Deep) Q Learning**
Q Value = Quality of gction
0. Init Q Value (= init model)
1. Choose action (model.predict(state))
2. Perform action
3. Measure reward
4. Update Q value (+ train model)
5.Repeat step 1 to 5

**Q Update Rule Simplified:**
Q = model . predict(state0)
Onew = R + v - max(Q(state1))

This Project is based on Reinforcement Learning which trains the snake to eat the food present in the environment.
**The Prerequisite for this project are:**

- Reinforcement Learning
- Deep Learning (Dense Neural Network)
- Pygame

## Algorithm

We have snake and food on the board randomly placed.

- calculate the state of the snake using the 11 values
- Now this current state is passed to the RL Model for the next state.
- After executing the next state calculate the reward. Rewards are defined as below:
    i. Eat food : +10
    ii. Game Over : -10
    iii. Else : 0
- Update the Q value and Train the Model.

After analysing the algorithm now we have to build the idea to proceed for coding this algorithm.

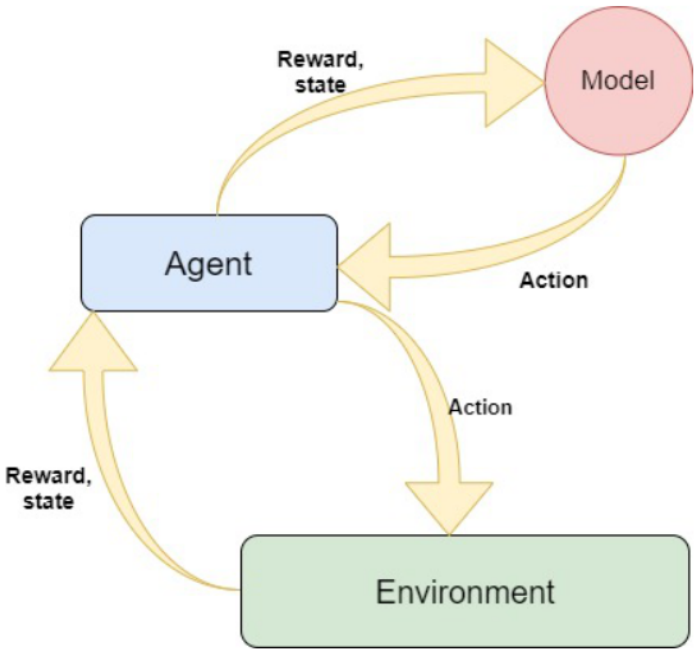
Action

[1, 0, O] -> straight
[0, 1, O] -> right turn
[0, O, 1] -> left turn

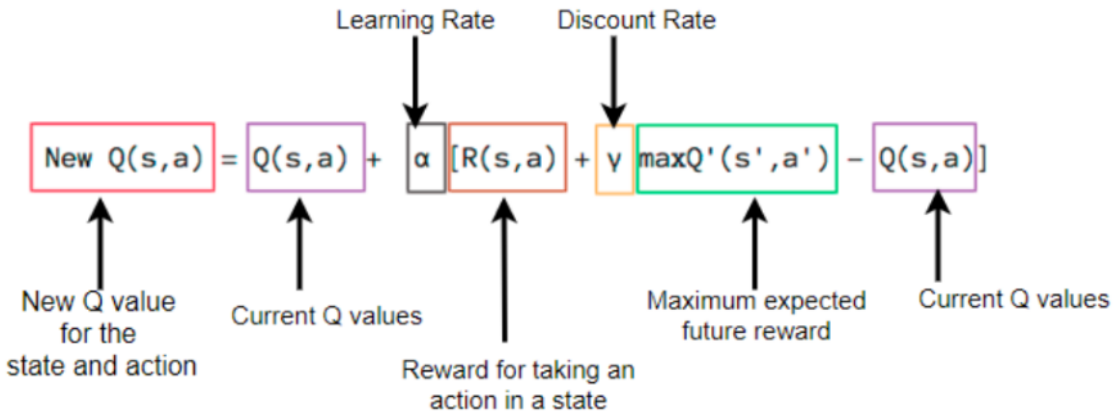
Our Project will be divided into three Modules named Agent, Game and Model

**We have to create three Modules in this project:**

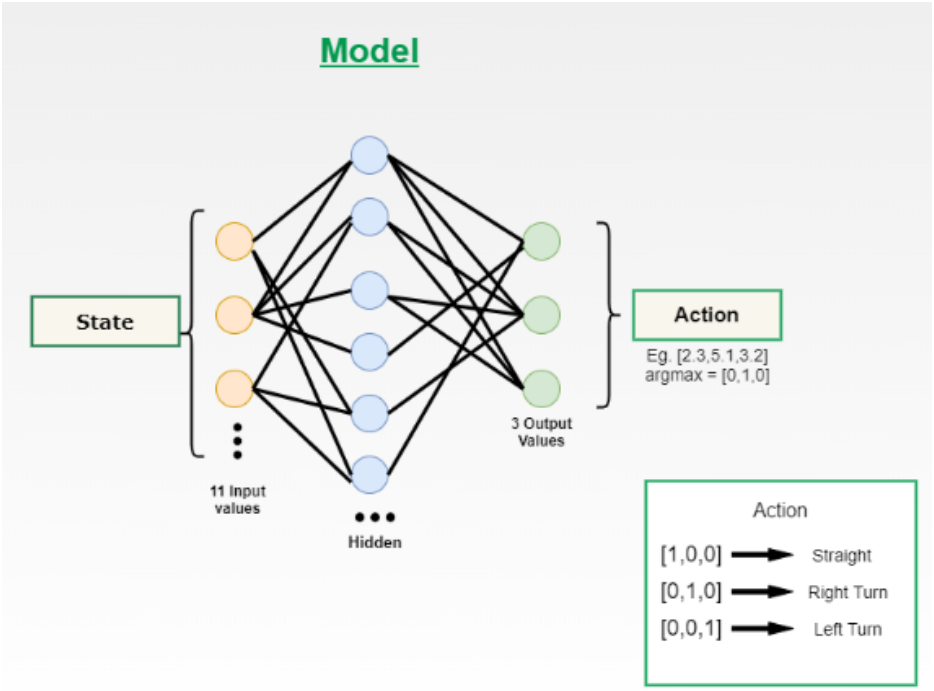
1. **The Environment** (the game that we just build)
2. **The Model** (Reinforcement model for move prediction)
3. **The Agent** (Intermediary between Environment and Model)




*Linking of Modules*




## The Model:




The model is designed using Pytorch, but you can also use TensorFlow based on your comfort.

We are using Dense neural network with an **input layer of size 11** and one **dense layer with 256 neurons** and **output of 3 neurons.** You can tweak these hyper parameters to get the best result.

**How models works ?**

- The game starts, and the Q-value is randomly initialized.
- The system gets the current state s.

- Based on s, it executes an action, randomly or based on its neural network. During the first phase of the training, the system often chooses random actions to maximize exploration. Later on, the system relies more and more on its neural network.
- When the AI chooses and performs the action, the environment gives a reward. Then, the agent reaches the new state and it updates its Q-value according to the Bellman equation.
- Also, for each move, it stores the original state, the action, the state reached after performed that action, the reward obtained and whether the game ended or not. This data is later sampled to train the neural network. This operation is called Replay Memory.
- These last two operations are repeated until a certain condition is met (example: the game ends).

**Part-I**

Creating a class named Linear_Qnet for initializing the linear neural network.
The function forward is used to take the input(11 state vector) and pass it through the Neural network and apply relu activation function and give the output back i.e the next move of 1 x 3 vector size. In short, this is the prediction function that would be called by the agent.
The save function is used to save the trained model for future use.

**Part-II**

Initialising QTrainer class
* setting the learning rate for the optimizer.
* Gamma value that is the discount rate used in Bellman equation.
* initialising the Adam optimizer for updation of weight and biases.
* criterion is the Mean squared loss function.

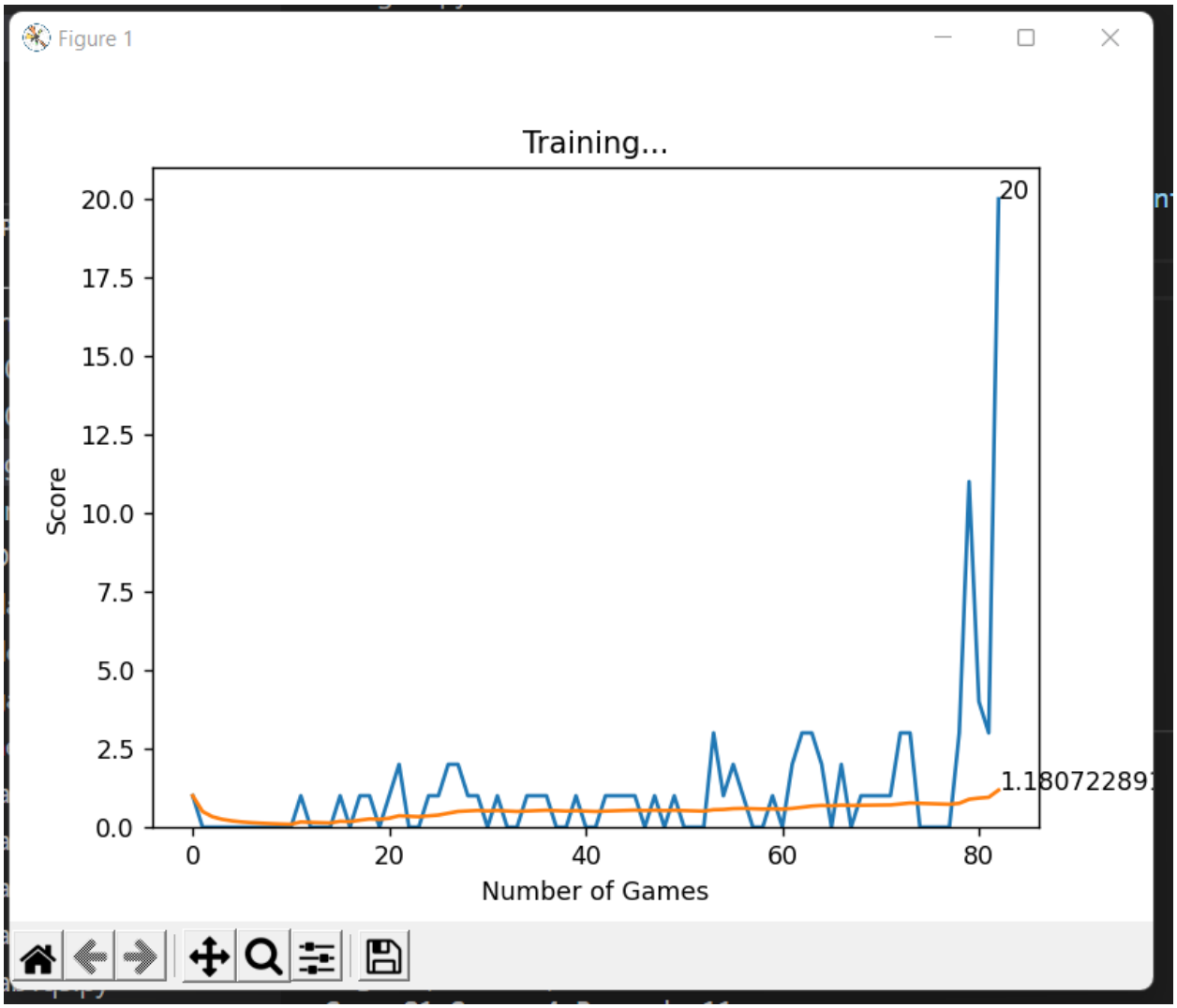
Train_step function
* As you know that PyTorch work only on tensors, so we are converting all the input to tensors.
* As discussed above we had a short memory training then we would only pass one value of state, action, reward, move so we need to convert them into a vector, so we had used unsqueezed function .
* Get the state from the model and calculate the new Q value using the below formula:

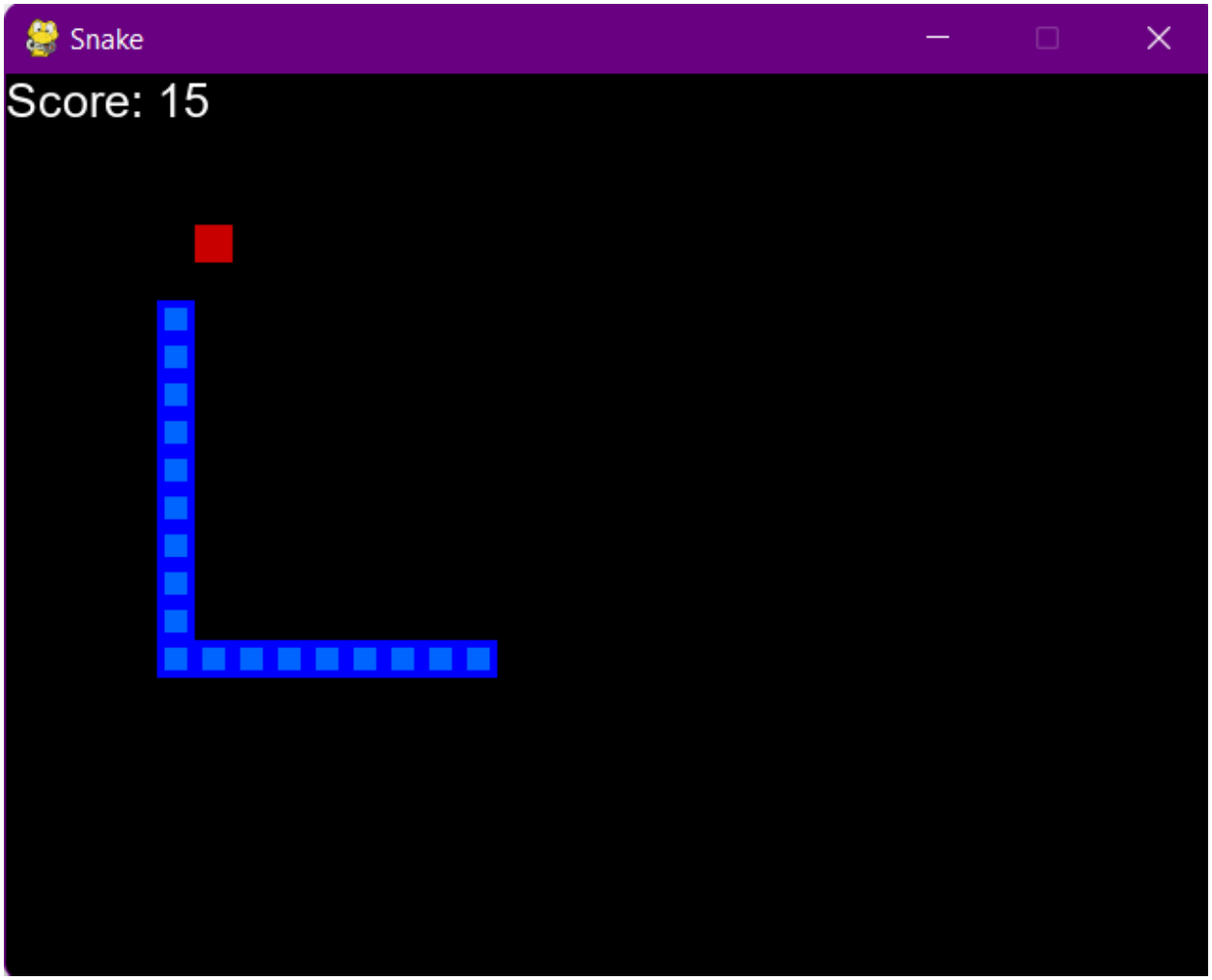
  Q_new = reward + gamma * max(next_predicted Qvalue)
* calculate the mean squared error between the new Q value and previous Q value and backpropagate that loss for weight updation.

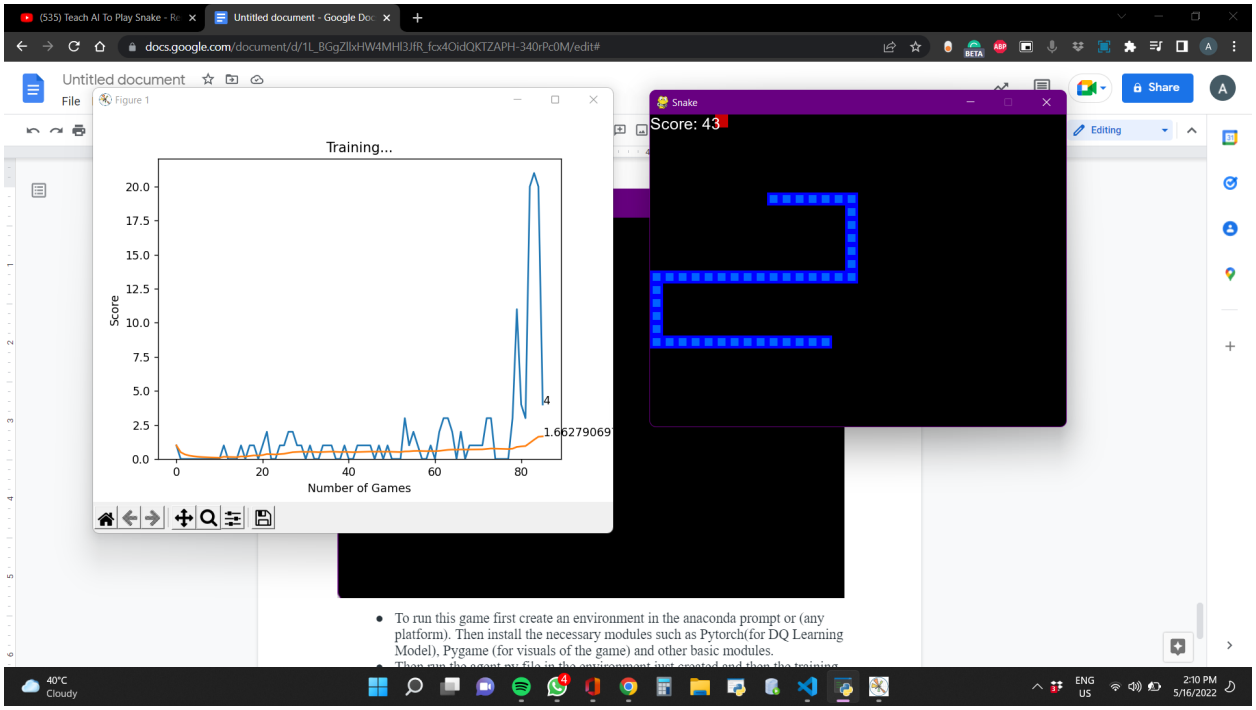
## The Agent

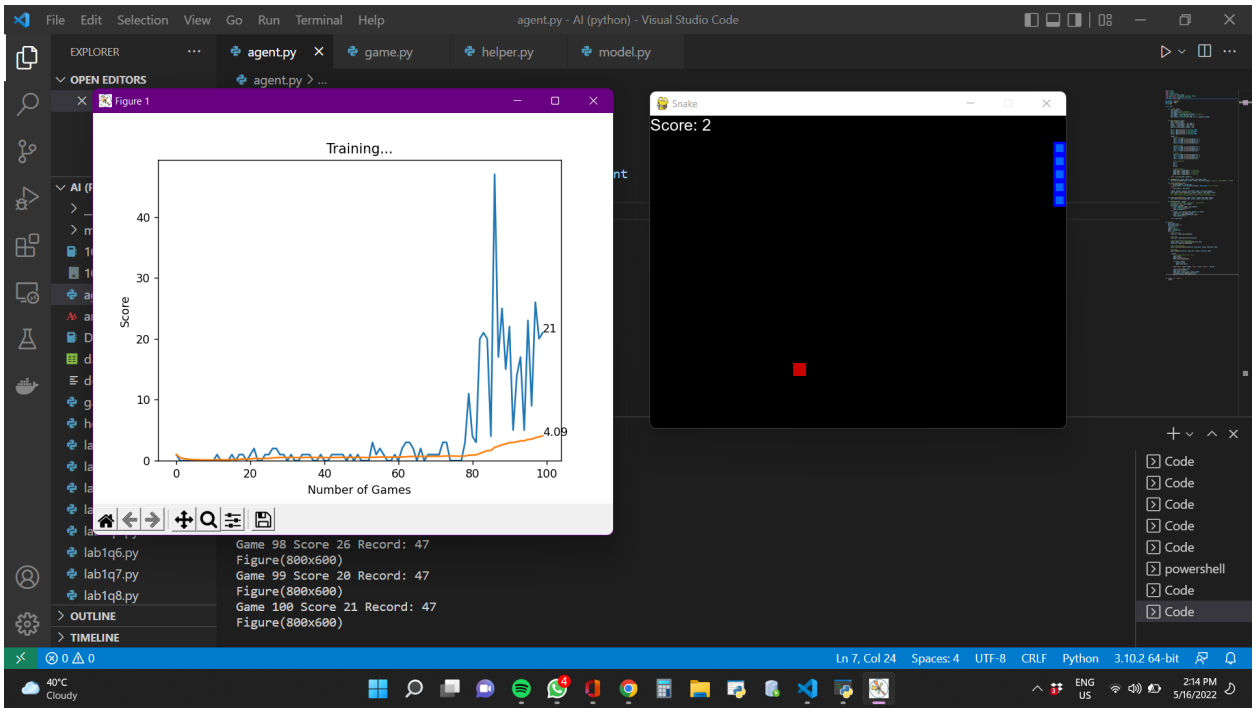
- Get the current state of the snake from the environment.
- Call model for getting the next state of the snake
- Play the step predicted by the model in the environment.
- Store the current state, move performed and the reward.
- Train the model based on the move performed and the reward obtained by the Environment. (Training short memory)

## Output:




Snake
Score: 15




- To run this game first create an environment in the anaconda prompt or (any platform). Then install the necessary modules such as Pytorch(for DQ Learning Model), Pygame (for visuals of the game) and other basic modules.
- Then run the agent.py file in the environment just created and then the training will start, and you will see the following two GUI one for the training progress and the other the snake game driven by AI.

- After achieving certain score you can quit the game and the model that you just trained will be stored in the path that you had defined in the save function of models.py.