

**TRAFFIC RIDER**

**REINFORCEMENT LEARNING**

Done By:

CHITTOOR KARTHIK – 106121033

P VISHNU VARDHAN - 106121089

# **AIM**

The aim of this project is to develop an intelligent agent capable of outperforming human players in a specific task, leveraging reinforcement learning as a practical application. By implementing a reinforcement learning algorithm, the project seeks to train the agent to make optimal decisions in a dynamic environment, with the ultimate goal of achieving superior performance compared to human players. Through the iterative process of learning from its actions and experiences, the agent adapts its behavior to maximize cumulative rewards, thereby mastering the task at hand. This project not only demonstrates the potential of reinforcement learning in real-world scenarios but also highlights its ability to enhance decision-making processes beyond human capabilities, paving the way for the development of autonomous systems capable of tackling complex tasks efficiently and effectively.

# **INTRODUCTION**

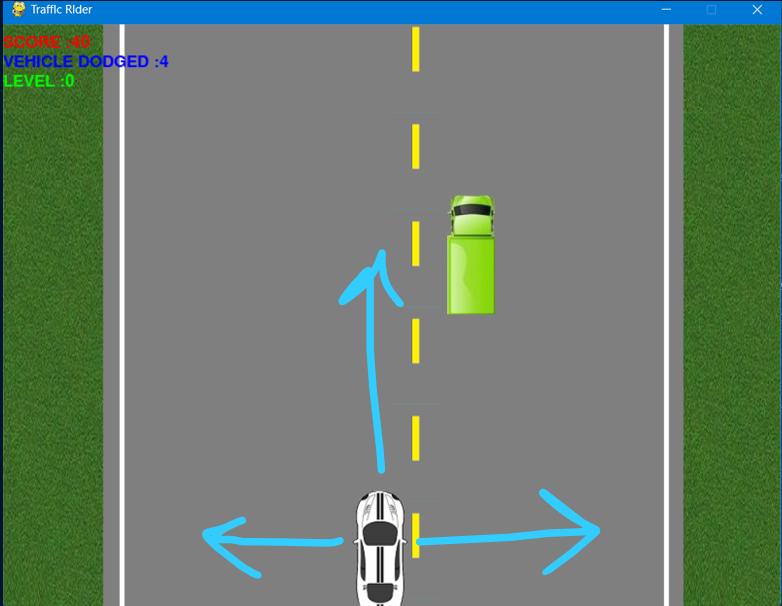
Traffic Rider is an arcade game developed using pygame package of Python. In this game, the player’s car is moving on a highway, at each the level, the player finds obstacle vehicle traffic ahead which he/she has to evade. The player is rewarded for every vehicle dodged and every level cleared. The speed of the car and the obstacles increases after each level, thereby increasing the difficulty.

This game is a way of testing the limits of human reflexes and demonstrate that an RL agent can outperform a human.

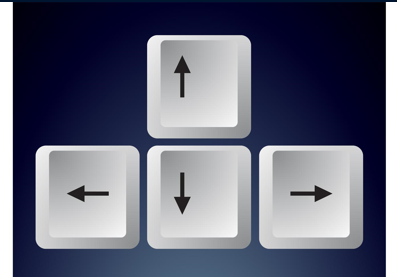
# **MAJOR STEPS:**

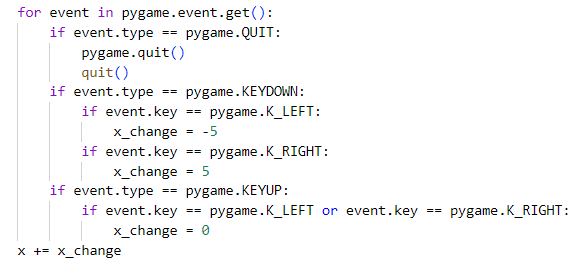
# **GAME CREATION** Building an Interactive Car Dodging Game

* The game is implemented in python using the pygame library.



* The user is represented by a white car in the bottom of the screen.
* The user can move left/ move right/ stay in the same place at each time unit. Controls are inputted via keyboard arrow buttons.
* The obstacles are randomly spawned one by one on the screen.
* On detecting a collision between the car and the obstacle, its game over.

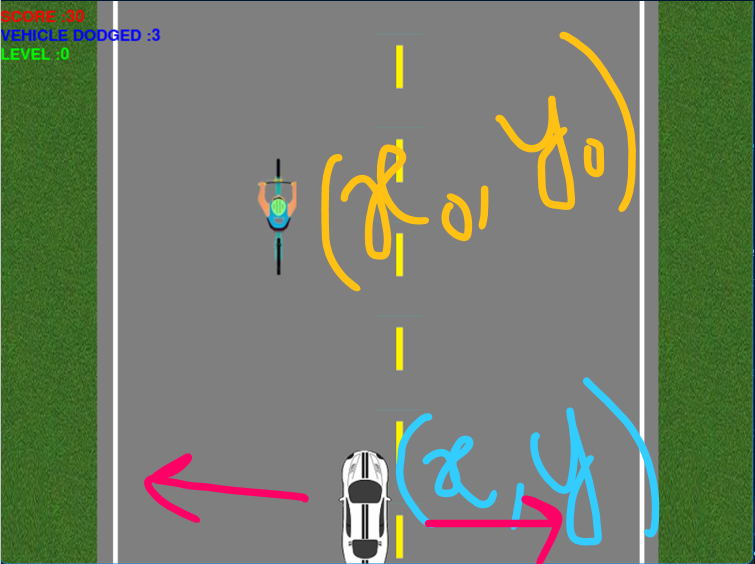




# **Q-LEARNING IMPLEMENTATION**

# Making the agent play on its own

* Define the states and actions
  + Define how the agent interacts with the game environment



**STATE: (car x, car y, obstacle x, obstacle y)**

**Each state is represented as a tuple of 4 elements like**

**{Car X-coordinate,**

**Car Y-coordinate,**

**Obstacle X-coordinate,**

**Obstacle Y-coordinate}**

**Transitions:**

**Go Left : Decrease car X-coordinate**

**Go Right: Increase car X-coordinate**

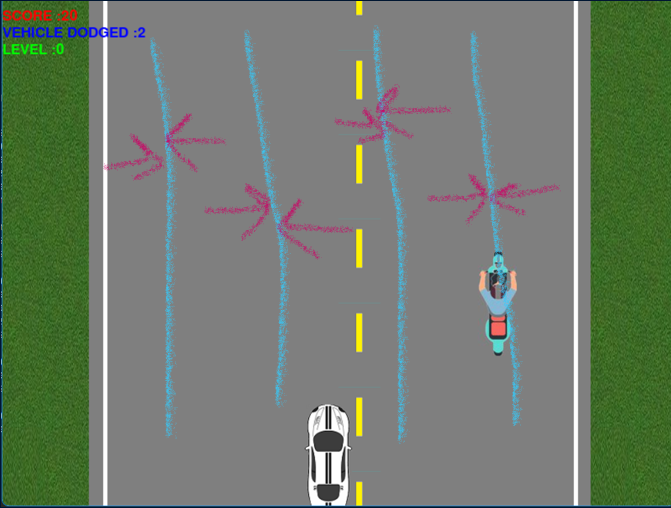
**Stay : No-change in car X-coordinate**

* State Reduction and Discretization
  + Space efficiency

Without any state reductions, we will have to store approx. 600\*800\*600\*800 states (for all possible carX,carY,obsX,obsY tuples).With these many states, initialising the Q-table itself takes a long time and we would be using so much space.







In this particular game, we realise that its sufficient to track only the X-coordinates of the car and the obstacle. This is based on the fact that it is gaurenteed that there is ATMOST ONE obstacle in the screen at any given time.So it would be enough for our car to move once on seeing the obstacle.

A simple way of reducing these X-coordinates further is to round them off to a multiple of 5 (here 5 is an arbitrary number,can choose any based on the move distance).

Space Complexity Optimisation:

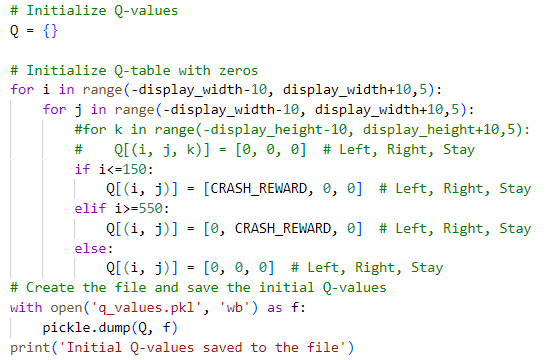
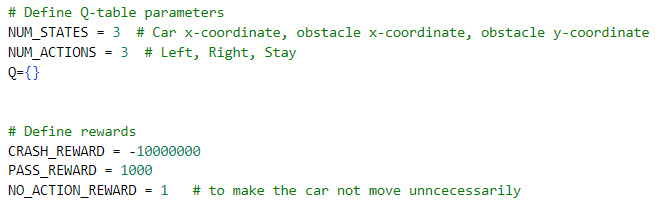
O(600\*800\*600\*800) -> O(600\*600) -> O((600/5) \* (600/5))

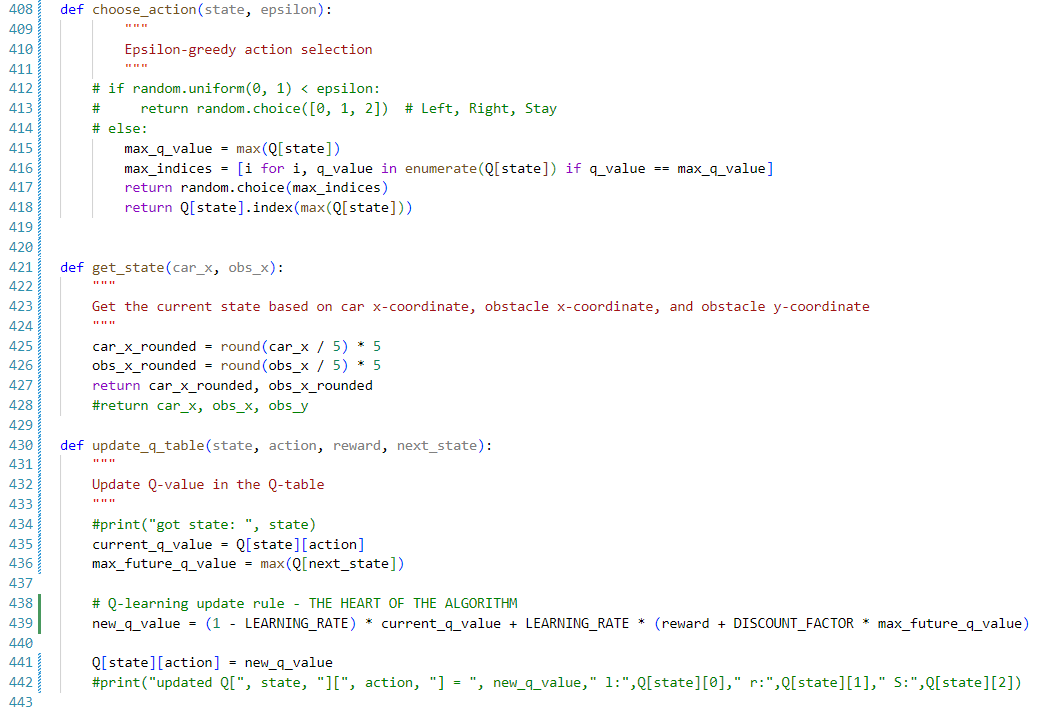
(Continuos 4 size tuple) -> (Continuos 2 size tuple) -> (Discrete 2 size tuple)

Reduced State: (Car X-coordinate, Obstacle X-coordinate)

Round off Co-ordinates for discretizing the state space

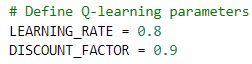
* Q-Table
  + Decide which transition is best at each state by choosing the transition with the maximum aggregate Reward value



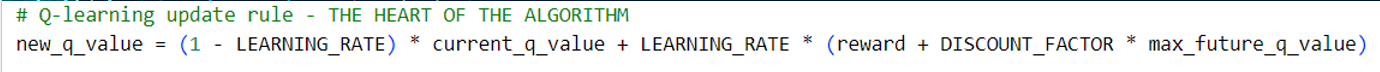
****

Sample Q-Table values:

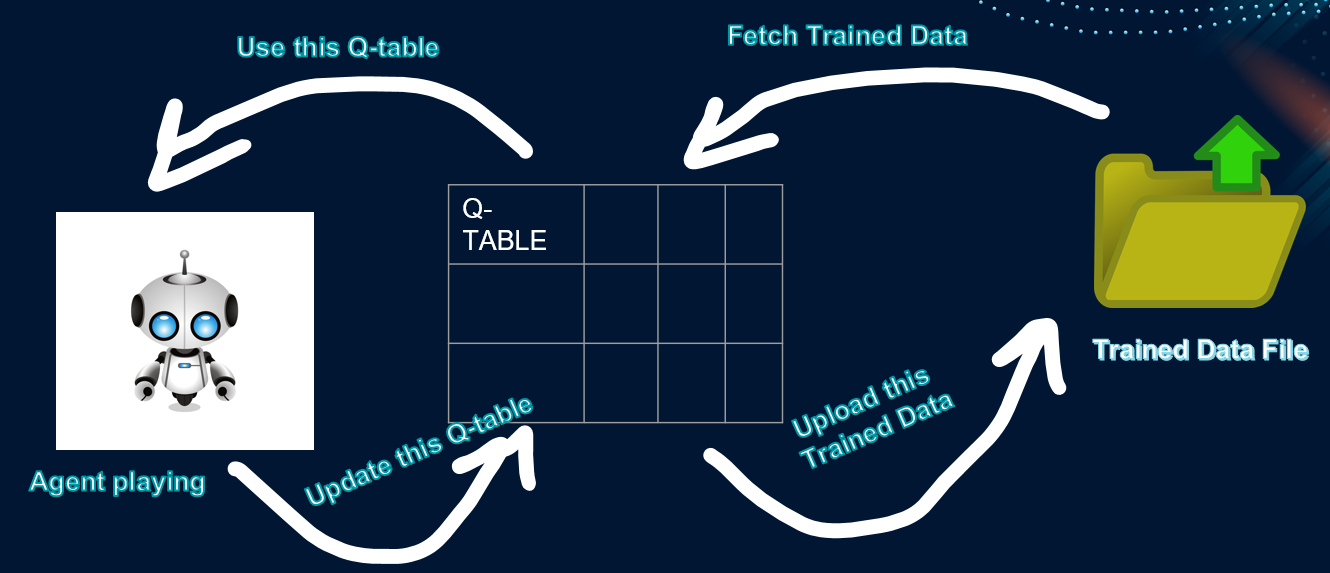


****

* **Choose action with maximum reward**

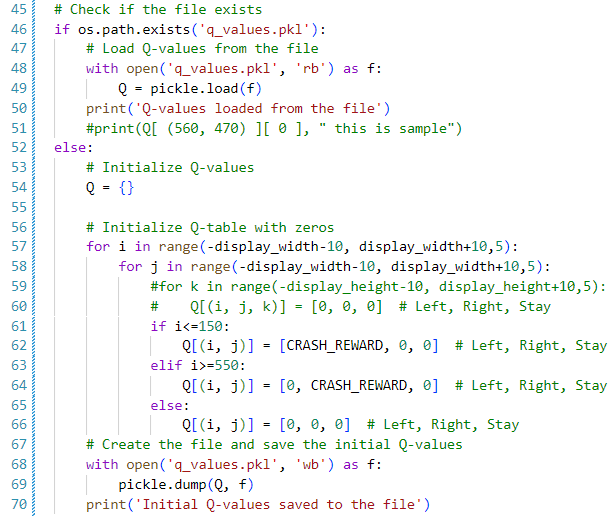


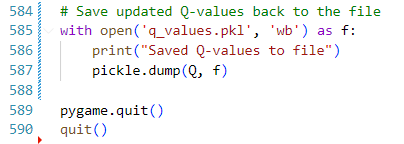
# **TRAINING THE AGENT**

Till desired performance and convergence of the Q-table values

In order to store the trained values, the pickle module is used.



The pickle module allows us to store the Q-table in binary format.The agent first loads this previously trained data into a local variable Q (its own Q-table). Then updates the values in its local Q-table. After the user quits the game, this newly update Q-table will be used to overwrite the contents of the pickle file again. 



**OUTPUT VIDEO:**

[**https://drive.google.com/file/d/1c-fUdxlWKuE8y0sSpas5Da3q6C6iuuek/view?usp=sharing**](https://drive.google.com/file/d/1c-fUdxlWKuE8y0sSpas5Da3q6C6iuuek/view?usp=sharing)

Ctrl +Click on the above link to see the output video.

**CODE FOLDER:**

[**https://drive.google.com/drive/folders/1i0BdQJFJxiLJ7dfxzmTkyLU\_uv6TKLAI?usp=sharing**](https://drive.google.com/drive/folders/1i0BdQJFJxiLJ7dfxzmTkyLU_uv6TKLAI?usp=sharing)

Ctrl +Click on the above link to see the code folder via google drive.

**PPT LINK:**

[**https://docs.google.com/presentation/d/1VQVlzQWHRg8Q38L9NTMyoGN7nodlsOdP/edit?usp=sharing&ouid=105604431366413362641&rtpof=true&sd=true**](https://docs.google.com/presentation/d/1VQVlzQWHRg8Q38L9NTMyoGN7nodlsOdP/edit?usp=sharing&ouid=105604431366413362641&rtpof=true&sd=true)

Ctrl +Click on the above link to see the power point presentation via google drive.