# Cliff Climber with Q-Learning

- **Mushty Sri Sai Kaushik (UID - 116917094)**
- **Santhosh Kesani (UID - 117035605)**

## Abstract:

Q-Learning is a Machine Learning technique that attempts to find the optimal action at each step by initializing random values for each action. Each action is assigned a corresponding reward and the technique attempts to achieve the maximum possible reward and remembers the sequence of events that preceded the action which provides maximum reward. These are stored in a Q-Table. This project attempts to use Q-Learning to understand the behaviour of a car that attempts to climb a cliff. This is done by assigning positive rewards for reaching the goal and negative rewards for every step that it does not. This is assessed for various sets of hills together and different heights of hills.

## Introduction:

The cliff climbing problem is commonly applied because it requires a reinforcement learning agent to learn position and velocity, two continuous variables. In this approach, two continuous state variables are converted to discrete states by storing each continuous variable as multiple discrete states. This approach is further improved by tuning the output values of one state to evaluate another state. The cliff climbing condition can be taken as a path planning problem where the agent is learning to understand the optimal values of position and velocity it would need to achieve in order to reach the goal. This can be modified and improved to be a three-dimensional problem to make the outputs more realistic. The objective is to be able to help the agent learn as quickly as possible and the project can help us understand the behaviour of the agent to Q-learning so it can be further implemented on top of for a more versatile utility in the real world.

## Background/Related work:

The main idea behind the implementation of this project was to learn how to apply a Reinforcement Learning technique on a problem which wouldn’t really depend on the state of the environment in which it would work. Basically, Reinforcement learning is all about making decisions one after the other and in simple words it can be said that the output depends on the state of the current input and the next input depends on the output of the previous input. So, in an environment, when everything in it is broken down into "states" and "actions", the states can be considered as the observations and samplings that we pull from the environment which are the input, and the actions are the choices the agent has made based on the observation are the output. But a RL algorithm which would work without actually considering the states is Q-learning as it is a model-free reinforcement technique meaning that the ‘agent’ wouldn’t priorly need to know anything about the states and it would take actions maximizing the reward in reaching a specified goal. Also, the algorithm’s performance can be tested in any environment we may choose.

**Approach:**

As a part of our approach for the project, we had to create our own environment (figure-2) and also simulate the Q-Learning algorithm and  we made a few comparisons with the existing environment which is the environment shown in the figure 1 below and contrasted the outputs.

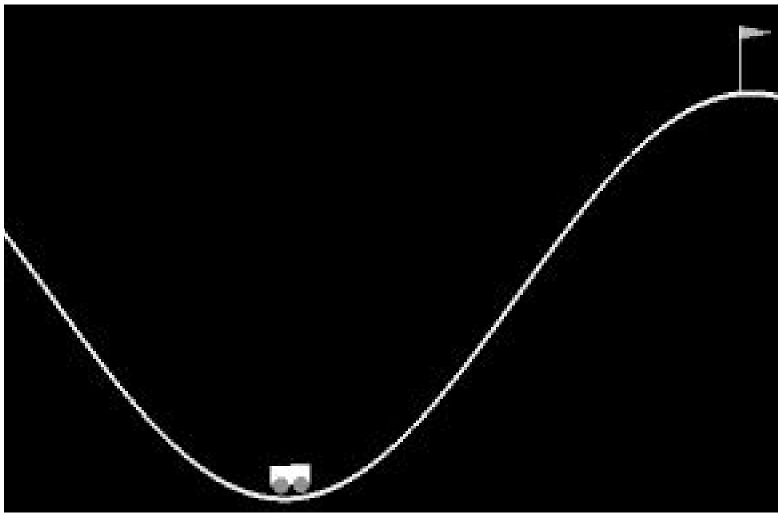


Figure - 1

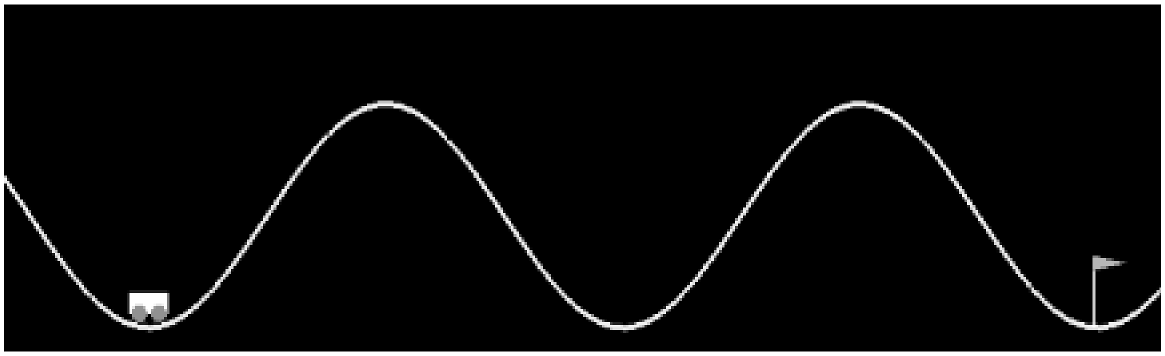


Figure - 2

In a  Q-learning algorithm, we create a table known as a *q-table* which is represented in the form of a matrix that becomes a reference table for our agent to select the best action based on the Q-value and this table follows the shape of [state, action] and we initialize our values to zero. This q-table becomes a reference table for our agent to select the best action based on the q-value and a similar approach was followed by our agent to perform it's Q-learning.
The updating and storing of the *q-values* is performed after each episode. The updating of this table occurs after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent. A terminal state for our project would represent every peak position it reaches in trying to reach the goals represented in the images. The agent will not learn much after a single episode, but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values. The next Q-value which has to generated after each episode is calculated using the following equation,

Source: https://en.wikipedia.org/wiki/Q-learning

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Figure - 3 (Q-learning equation)

Moreover, during the implementation of the algorithm there exists two ways in which the agent would interact with the environment. The first is where the agent fully utilizes the q-table and this is done by using the q-table as a reference and by viewing all possible actions for a given state. After this, the agent tries to implement the action based on the max value of those actions. This  method is known as *exploiting* since the agent tries to utilize the available information in the q-table to take an action rather than trying to learn from exploration of more state spaces. So, we can assume from this that the second way is to take action randomly instead of just selecting actions based on the max future reward. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process.So, this method is called *exploring*. For our project, we implemented a part of both of these methods, clevering utilizing the advantages of both the methods. Inorder to achieve this, we considered an epsilon value and by varying it accordingly until the desired factor is achieved which would implement both the techniques mentioned in a such that the final motion of the car in our project would be optimal and efficient.

## Implementation:

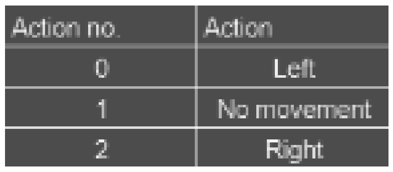
Code Implementation: The entire implementation of our project is done in python using the in-built libraries such as OpenAI gym, numpy, random and other basic libraries. The environments which we mentioned before are entirely built using OpenAI Gym module.

Algorithm Implementation: Most of the algorithm implementation is similar to what has been explained in the approach section which is the creation of the q-table, updating the values using the Q-learning equation presented and in addition to this we defined a reward system wherein a reward of -1 is given by default(while trying to learn)  and a reward of 0 is given when the agent reaches the goal i.e, there is no negative reward in reaching goal. The other basic parameters that we have used have been initialized in the program  as follows:

- Learning rate - 0.1
- Discount factor - 0.90
- Training - 8000 episodes
- Epsilon = 0.5

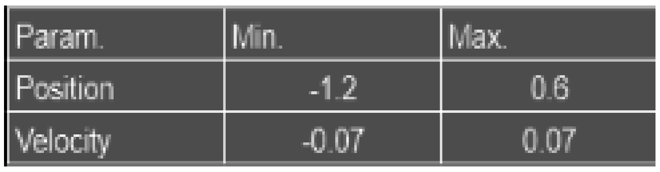
Most of the above values mentioned aren't chosen randomly but have been reached through certain trails which would give us the best results.

Also, the default parameters for the fundamental problem are assigned to be as follows:



| Action no. | Action |
|---|---|
| 0 | Left |
| 1 | No movement |
| 2 | Right |

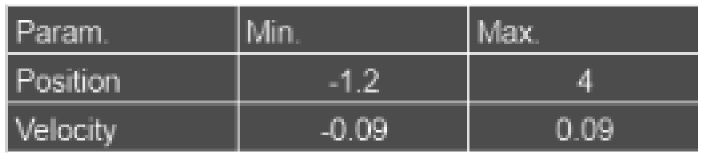
Table - 1



| Param. | Min. | Max. |
|---|---|---|
| Position | -1.2 | 0.6 |
| Velocity | -0.07 | 0.07 |

Table - 2

Here, the action space was defined to be for 3 actions with the agent having the ability to move either left, right or no movement. The minimum and maximum values for both variables of velocity and position are mentioned in table - 2 for the standard cliff car environment.



| Param. | Min. | Max. |
|---|---|---|
| Position | -1.2 | 4 |
| Velocity | -0.09 | 0.09 |

Table - 3

However, in case of table - 3 the values have been tweaked to obtain the updated environment. The outputs of the environments are contrasted in the project to understand the fundamental behaviour of the agent to the Q-learning method.

**Results:**

For the implementation in environment - 1 with varied outputs, the plot for rewards to episodes is found as follows:

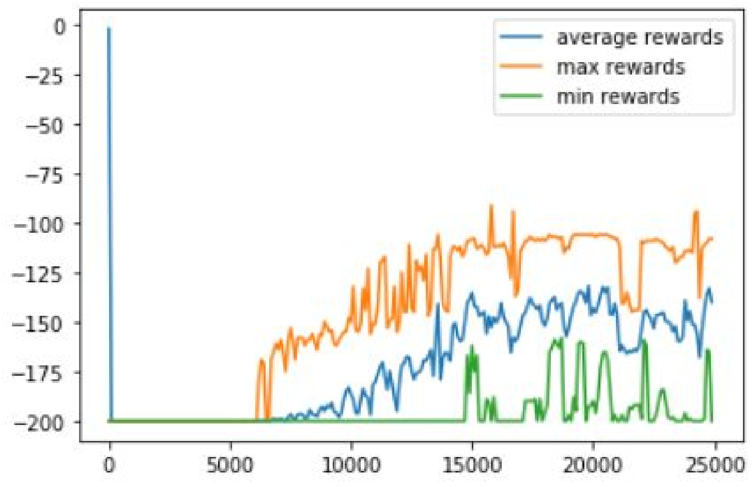


Figure - 4

The above shown figure - 4 represents the sudden increase in all three - average, maximum and minimum rewards - which is caused when the agent has reached the goal once. Following reaching the goal, the agent understands and tracks back the previous actions taken by the agent in order to reach the goal. As shown in the plot, the reward continues to increase as the episode increases, which implies that the goal is reached more often and it begins to hit a stale point at around 20,000 episodes.

Similarly, in the case of environment - 2, the outputs are shown below with a different plot but similar behaviour. The rewards are to be divided by 1000 and the image.

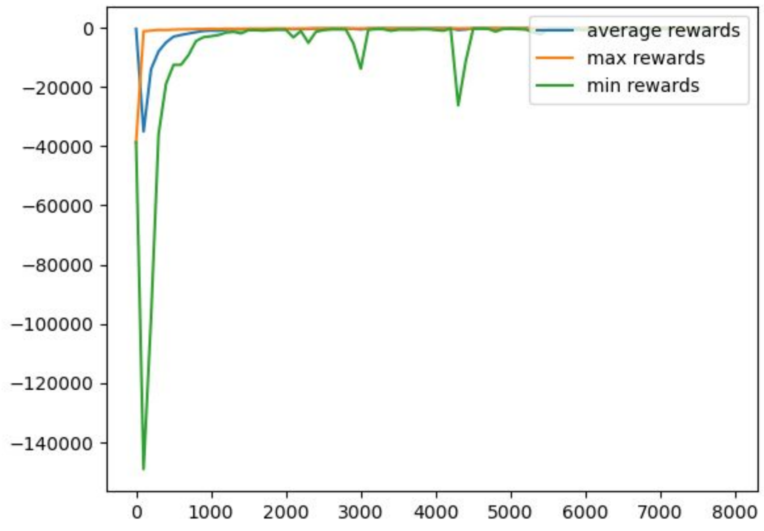


Figure - 5

After understanding the behaviour of the agent by assessing the reward function plots, the rewards were modified in order to be able to find the optimum reward calculation method to get a quicker output.
For this, we have updated the reward function as follows:

- Adjust reward based on car position:
    - reward = state_1[0] + 0.5
- Adjust reward for task completion:
    - if state_1[0] >= 0.5:
      reward += 1

On updating the reward function, the plots were found to be as follows:

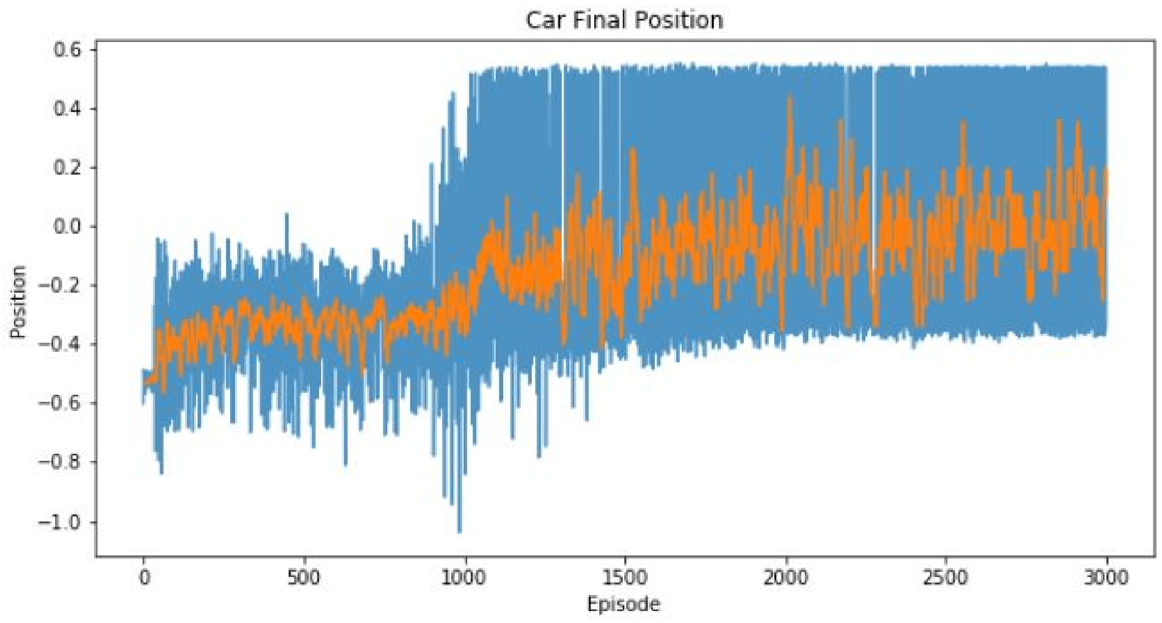


Figure - 6

The Figure - 6 shows the output for the original reward function with the plot between position and episode. Here, the orange plot represents the position of the agent. The goal position is obtained at around 1500 episodes and keep attaining at regular intervals. However, in case of the update reward function, we can find the following plot:

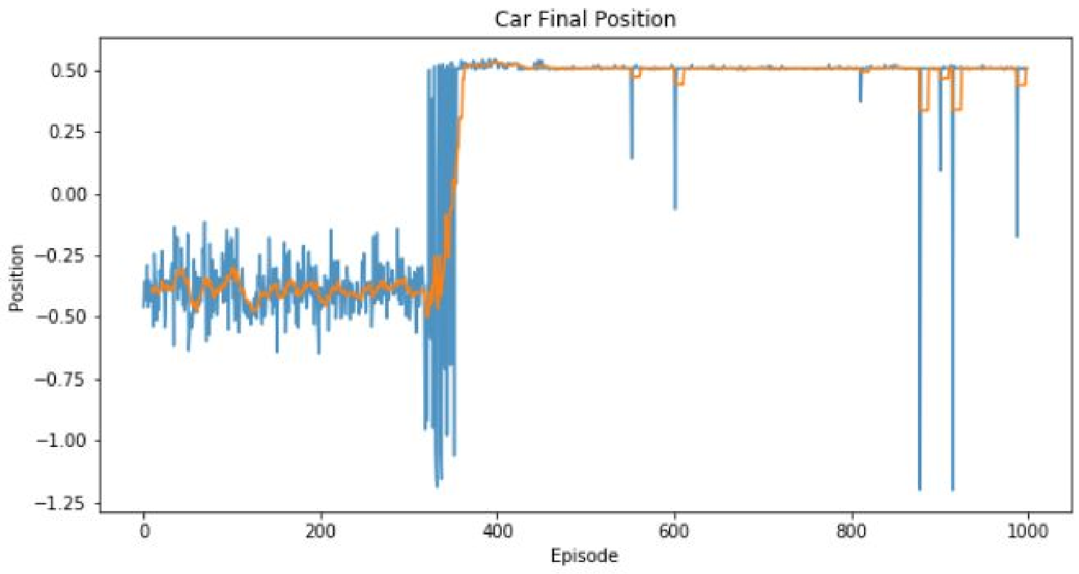


Figure - 7

In this plot, we note that the agent is consistently reaching the goal at almost every iteration in comparison to the original reward system. Thus, the updated reward system is found to be performing better w.r.t accuracy

**Analysis:**

The behaviour of the agent can be understood by representing the q-values of the q-table for each action.

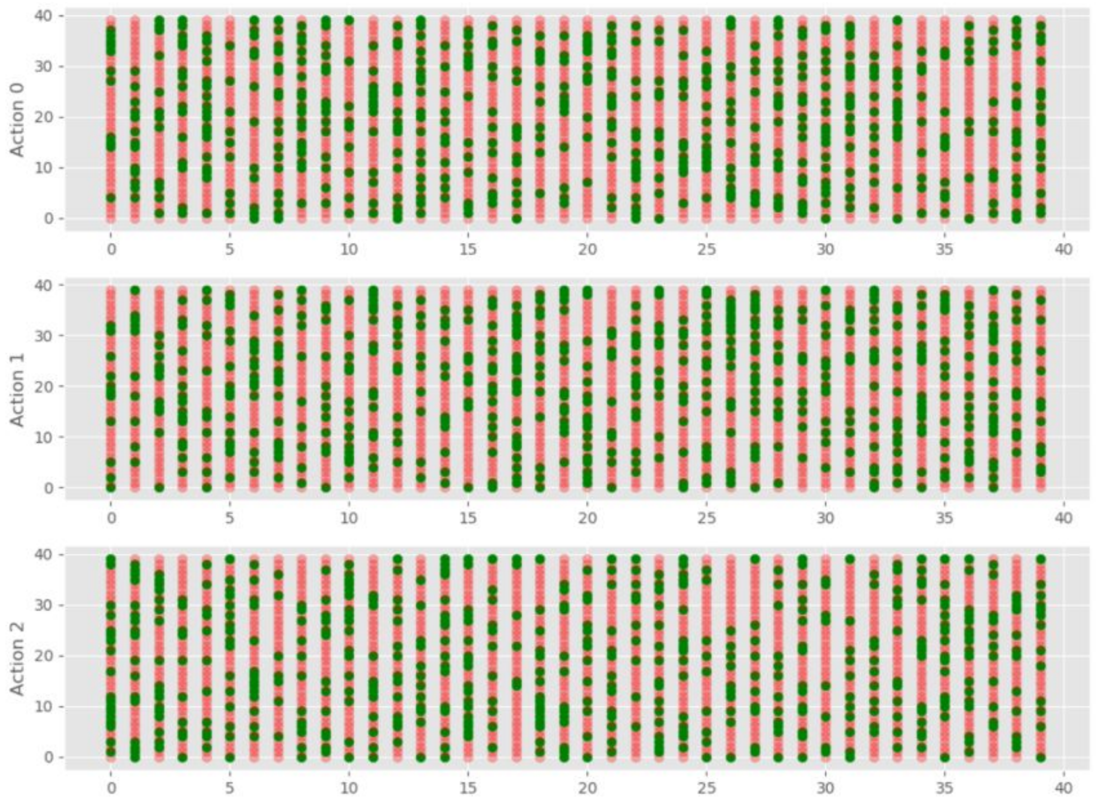


Figure - 8

The above shown figure - 8 represents the randomly assigned q-values for all three actions where the green is the chosen q-value of the action performed.

After training the agent, the plots are found to be as follows:

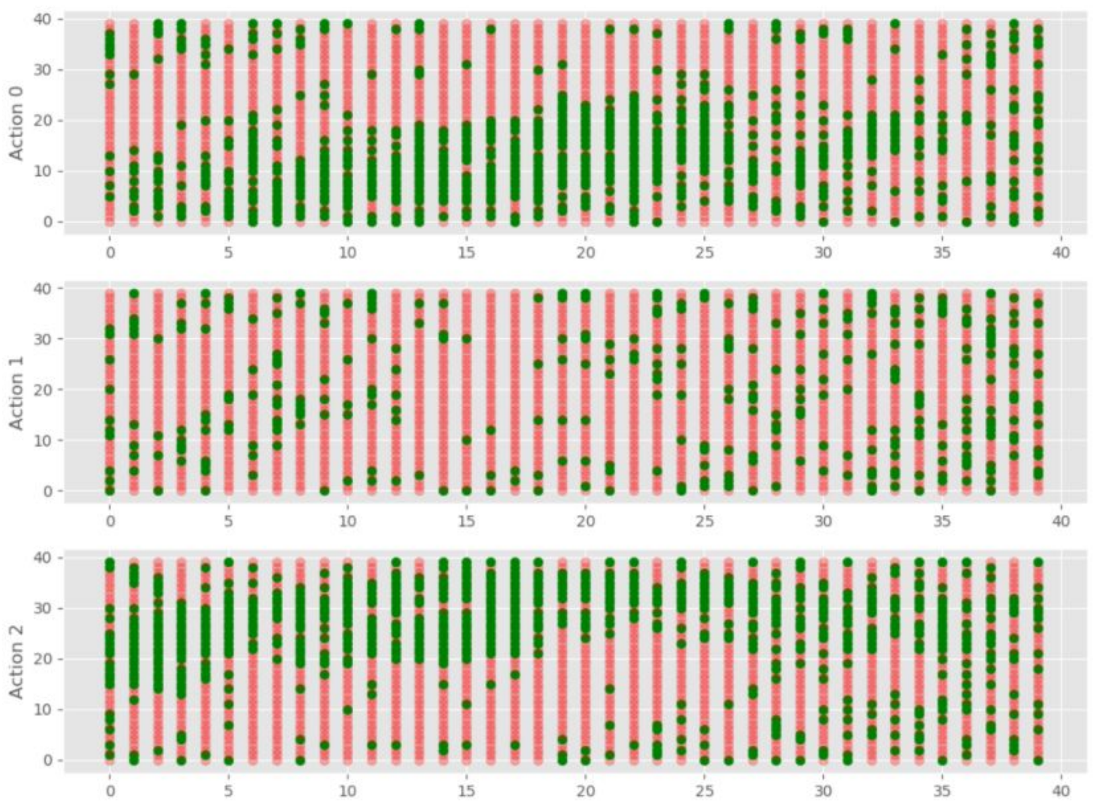


Figure - 9

This shows the Q-values of the agent after training of 8000 episodes. Thus, the increase in higher values for action - 2 (move right) and lower values for action - 0 (move left) compared to the decrease in the overall Q-values for action - 1 (no movement).
Thus the change in the behaviour before and after learning is represented in the plots to show the consistency in reaching the output once the learning is performed.

Following the learning, the agent was found to perform better in the environment - 1. The goals were reached quicker and the video outputs for the same are in the .zip file uploaded along with this submission with the name ENPM690_final.zip. These include the README file explaining the dependencies to run the code.

The outputs of the agent that reach the goal is shown:

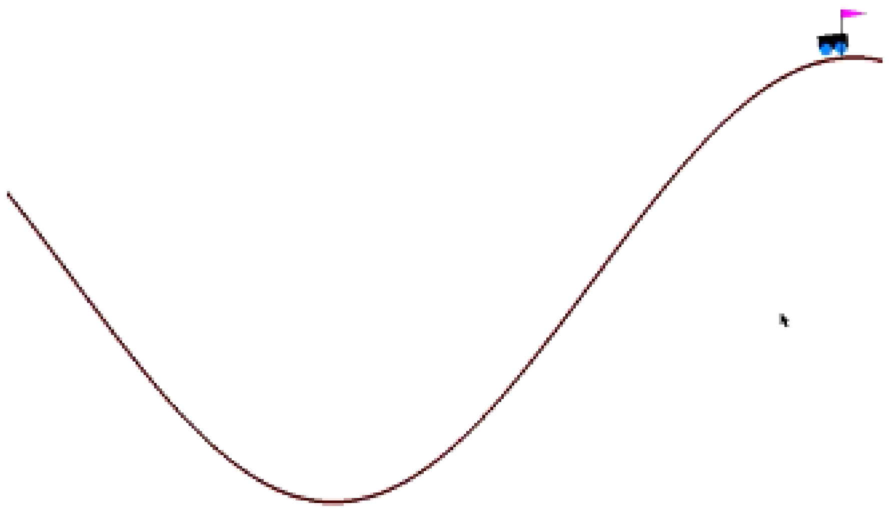


Figure - 10 (Output for environment - 1)

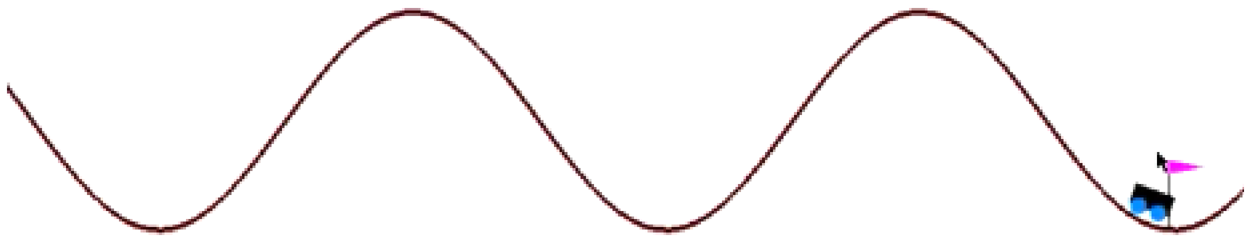


Figure - 11 (Output for environment - 2)

## Conclusions:

The behaviour of an agent learning to reach a goal using Q-learning is studied in the project in various environments by modifying various parameters. The plots represent the changes of the environments and how a Q-learning agent performs its learning. This can be further improved as to be discussed in the conclusions.

## Future Work:

The current project gives a representation of the various modifications and updates made to the standard cliff car problem by changing the environment and the vehicle parameters, gola position, reward system etc. The optimum conditions for each required output are also found after studying their behaviour. This can be further improved by updating this to a path planning problem in a three-dimensional space that enables vehicles/robots to learn traversing in uneven terrain in quick time. This can also be modified to be a game with the agent needing to avoid obstacles while climbing cliffs like the familiar T-Rex game.

## Bibliography:

- Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction, Section 6.5, Second Edition, MIT Press, Cambridge, MA, 2017.
- http://gym.openai.com/docs/
- https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
- https://en.wikipedia.org/wiki/Q-learning