horizontal line

RL for Lawn Mover Problem

09.04.20XX

**─**

Your Name

Your Company

123 Your Street

Your City, ST 12345

[**Introduction 2**](#_b5nbbbp2onrm)

[**Part 1: 2**](#_evndxwnruebt)

[Output 2](#_xgbzbku6wuwp)

[Description of environment 3](#_al3rhuuygfbk)

[Rewards 3](#_4u8yt8cy7pol)

[Visualization 3](#_s269k99jwc4v)

[Saftey in AI 4](#_2fu4cj3pn0u)

[**Methodology 4**](#_mmvfm98bmluu)

[SARSA 4](#_p8mom2uk3qww)

[Q-Learning 5](#_i4fwa9uvjnmr)

[**Part 2 5**](#_6s5wge64dmt5)

[Alpha vs Rewards 5](#_h222jk7n6hka)

[Beta vs Rewards 6](#_1anykxg499rt)

[Eps vs Rewards 7](#_kh2dzvwgj3ph)

[**Part 3 - Q-Learning 8**](#_oedbyzts577k)

[Alpha vs Rewards - q learning 8](#_jyps3iokv6fg)

[Beta vs Rewards - q learning 9](#_3nalv1wp8ckb)

[Eps vs Rewards - q learning 10](#_5ahpzy5t8n5r)

[**Comparison of Models 12**](#_995tnba88g34)

[Reward vs Episodes and No of Actions - Q Learning 12](#_mjzekiwbqlex)

[Episodes vs Rewards Per Action - Q Learning 12](#_1kkvu33vfill)

[Reward vs Episodes and No of Actions - SARSA 13](#_r6adqq3mloxd)

[Episodes vs Rewards Per Action - SARSA 13](#_q2zasvrga0o8)

[SARSA vs Q-Learning 13](#_4a062akx7atu)

[**Bonus -Two Step SARSA 14**](#_fifu4kamf918)

[Implementation 14](#_locujbf6nlhu)

[Graphs 15](#_vpg3wj70qc4w)

# 

# Introduction

To generate a RL environment to simulate the lawn mover using the SARSA algorithm and Q learning algorithm

# Part 1:

## Output

Current State: 0 0

Action: up

New State: 0 1 Reward: 0 Done: False

Current State: 0 1

Action: up

New State: 0 2 Reward: 0 Done: False

Current State: 0 2

Action: up

New State: 0 3 Reward: 6 Done: False

Current State: 0 3

Action: right

New State: 1 3 Reward: 0 Done: False

Current State: 1 3

Action: right

New State: 2 3 Reward: 6 Done: False

Current State: 2 3

Action: right

New State: 3 3 Reward: 100 Done: True

Reward: 12

## Description of environment

The canvas is first set up by the \_built\_maze function, which then constructs the grid. The create\_rectangle and create\_oval routines are then used to construct the maze's blocks, goal, and player. Additionally, it establishes the starting spots for the player and the goal, respectively.

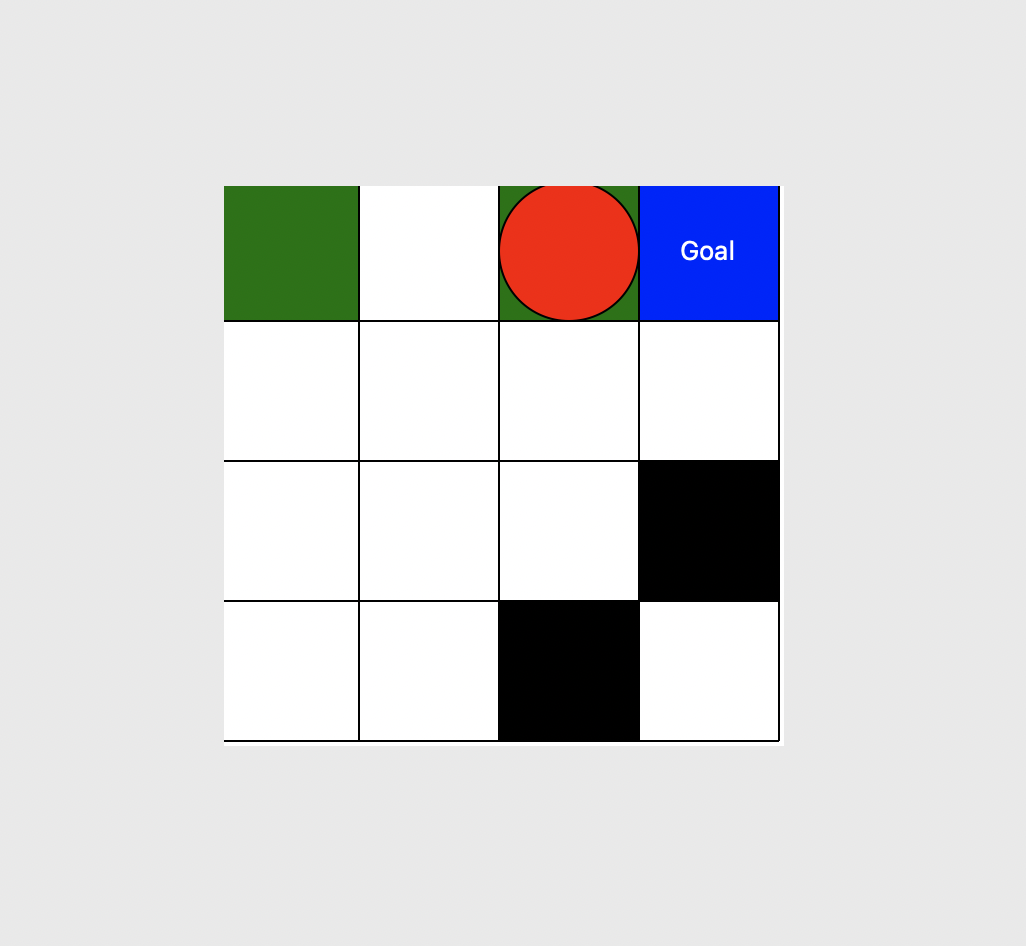
Using a point in the maze, the position\_to\_block function, a convenience function, gives the start and finish coordinates of the matching block.

When the RL agent performs an action in the environment, the step function is called. The function moves the player in accordance with an input action (up, down, left, or right). The function gives a payout of 0 if the player runs into an obstacle.

## Rewards

* Goal 100
* Battery : 6
* Rocks: -5

## Visualization



## Saftey in AI

Safety is guaranteed in the defined grid-based RL environment in a number of ways. First, the environment is created so that the agent can only perform actions that are permitted by the environment's current state. As a result, the agent is prevented from acting in a way that might have harmful or unwanted results. The agent can only navigate within a certain set of states thanks to the environment's defined and constrained state-space. This aids in preventing the agent from going into potentially dangerous or unintended environments. In order to further ensure the safety of the environment, the environment's incentives and penalties are created in a way that encourages the agent to engage in safe and desirable behaviour.

# Methodology

## SARSA

An on-policy, model-free reinforcement learning technique called SARSA (State-Action-Reward-State-Action) is used to resolve Markov decision processes (MDPs). Based on the present state, current action, current reward, next state, and next action, the algorithm calculates the predicted value of a state-activity pair. Using a temporal-difference (TD) update rule, which considers the discrepancy between the expected value and the actual value, the estimated values are updated. SARSA works well for issues where the agent may interact with its surroundings and gain knowledge by making mistakes. Initialise the Q-table with random values, observe the current state, choose an action based on an epsilon-greedy policy, carry out the action, observe the next state and reward, update the Q-value based on the TD error, and so on.

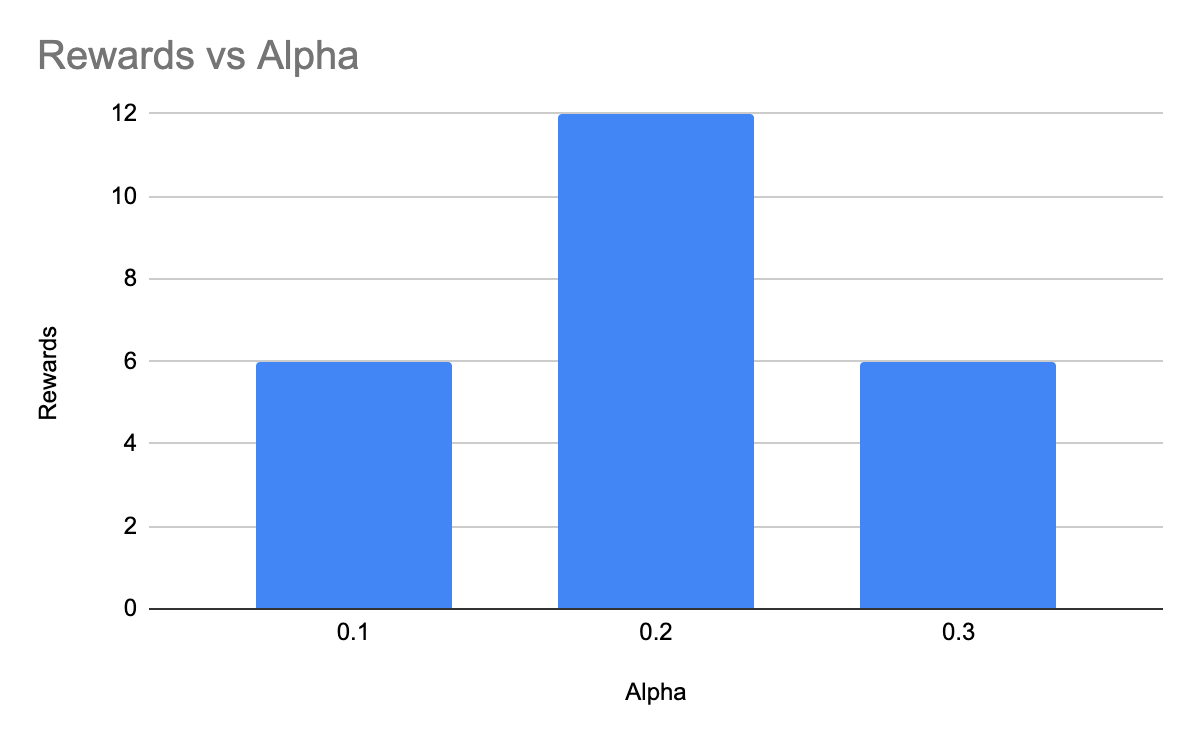
## Q-Learning

The goal of the model-free reinforcement learning algorithm known as Q-learning is to discover the best action-value function for a particular environment. The algorithm performs actions in the environment, observes the rewards and subsequent states, and iteratively updates a table of action values known as the Q-table. The Bellman equation, which explains the link between the present Q-value and the anticipated future Q-values, is the foundation for the updates. In order to balance discovering novel actions with using the most well-known action, Q-learning employs an exploration-exploitation technique. As the iterations increase to infinity, the method is guaranteed to converge to the best Q-values.

# Part 2

## Alpha vs Rewards

| Alpha | Rewards |
| --- | --- |
| 0.1 | 6 |
| 0.2 | 12 |
| 0.3 | 6 |



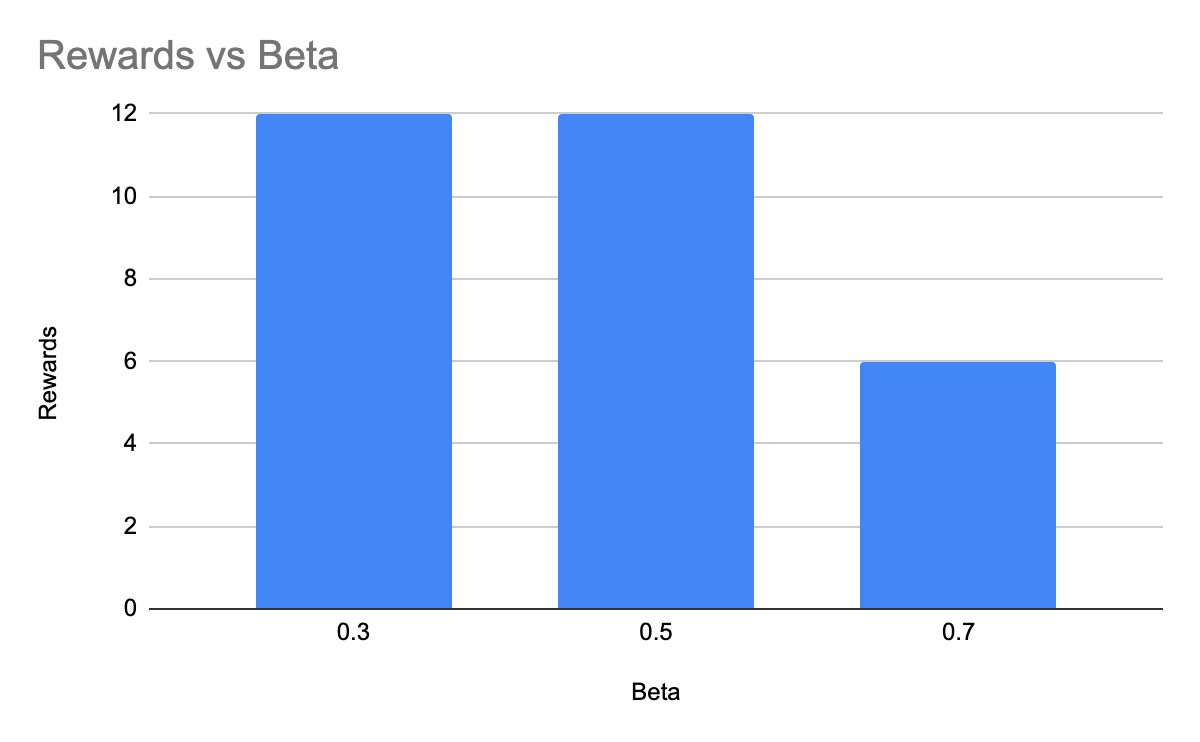
Here Alpha (α) controls the rate at which the algorithm updates the Q-values based on the observed rewards and transitions. A high value of α means that the algorithm will update the Q-values more aggressively, while a low value means that the updates will be more conservative. The choice of α depends on the specific problem and the desired learning rate.

**Inference:**

We can see that we need an optimal value of alpha to get to the final state, a less value of alpha means the algorithms is updating the values aggressively, So in our case 0.2 was optimal value with max rewards.

## Beta vs Rewards

| Beta | Rewards |
| --- | --- |
| 0.3 | 12 |
| 0.5 | 12 |
| 0.7 | 6 |



Beta is a hyperparameter that controls the weight given to the maximum Q-value in the next state during the update. A high value of beta gives more weight to the maximum Q-value, which can help to stabilize the learning process and prevent oscillations in the Q-values.

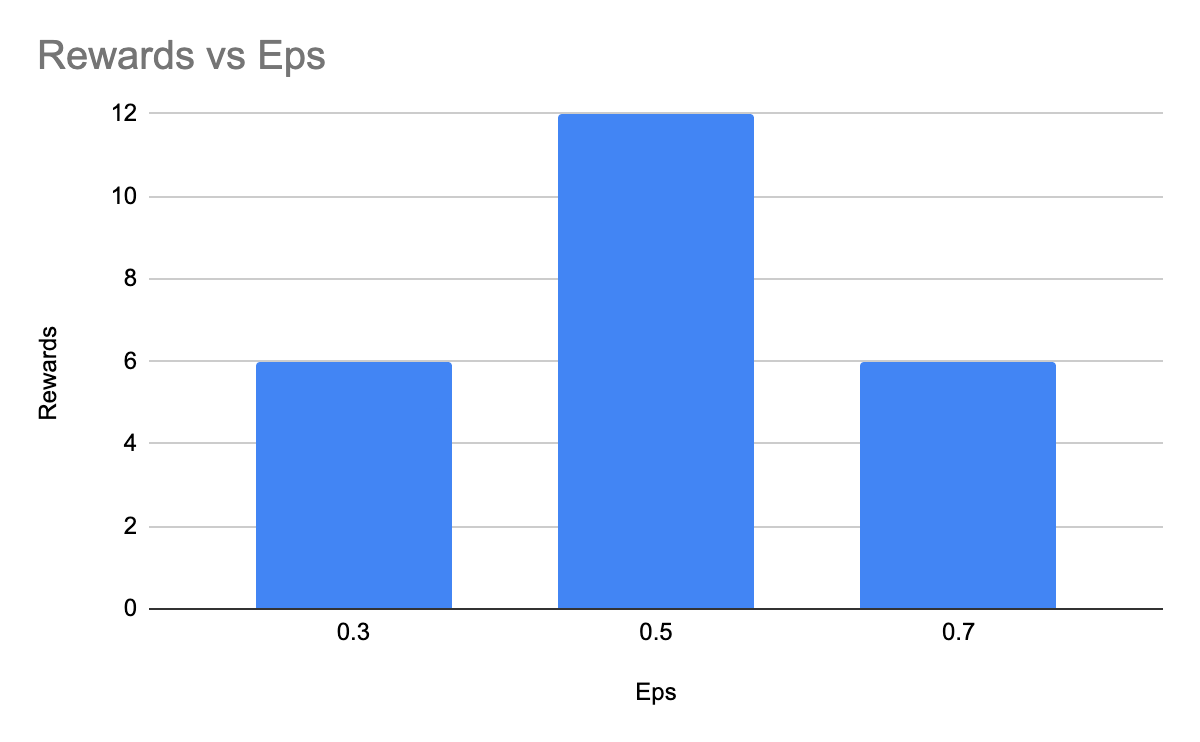
**Inference**

In our cases, we can see that the higher values of beta (0.7) was resulting on a reduced reward, which makes us choose the 0.5 value as the final optimal values

## Eps vs Rewards

Epsilon (ε) is used in the exploration-exploitation tradeoff, which refers to the balance between exploring new states and actions to learn more about the environment, versus exploiting the current knowledge to make the best decision. Epsilon determines the probability of taking a random action instead of the action with the highest Q-value, which helps the agent to explore different options.

| Eps | Rewards |
| --- | --- |
| 0.3 | 6 |
| 0.5 | 12 |
| 0.7 | 6 |



**Inference**

The higher the value of epsilon, the higher it explores the environment by making new decisions as much as possible. However Lower epsilon means that the model does not explore much. This is clearly visible on the graph above where both extremely low epsilon extremely high values are giving bad results, So we can pick the 0.5 as optimal value.

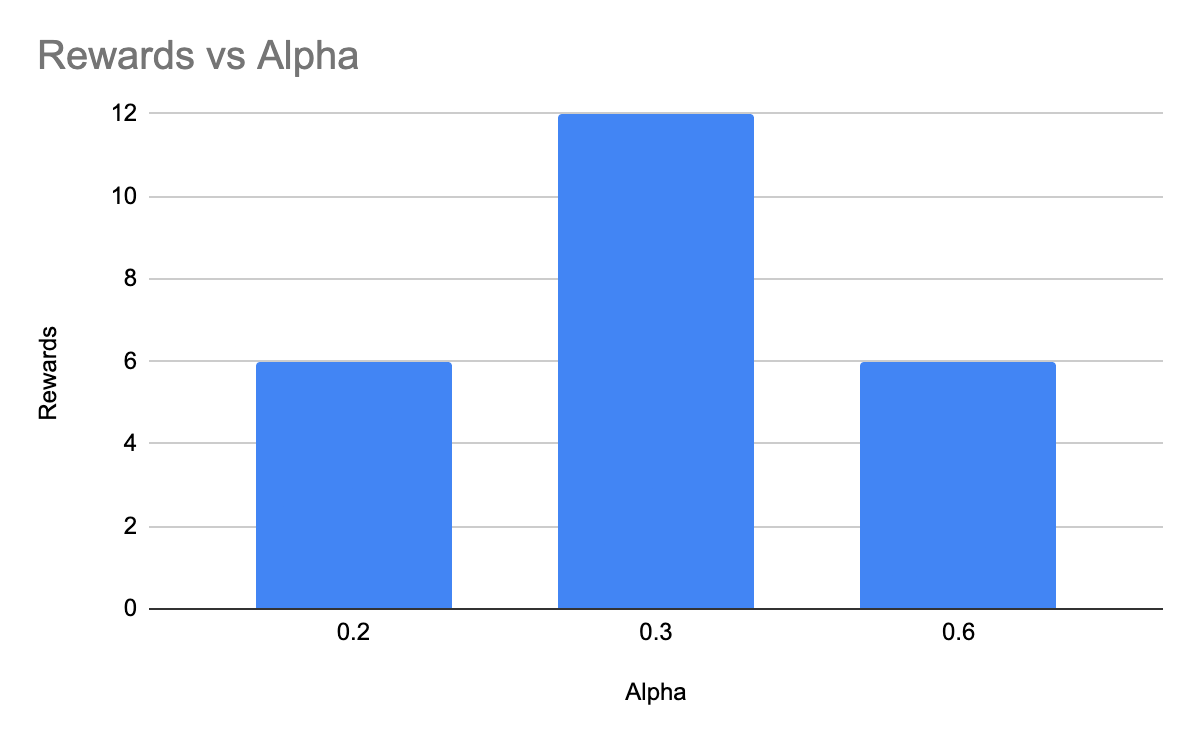
Based on our Hyperparameter tuning, we can obtain the following Hyperparameters are optimised for best performance.

* Alpha - 0.2
* Beta - 0.3
* Eps - 0.5

# Part 3 - Q-Learning

## Alpha vs Rewards - q learning

| Alpha | Rewards |
| --- | --- |
| 0.2 | 6 |
| 0.3 | 12 |
| 0.6 | 6 |



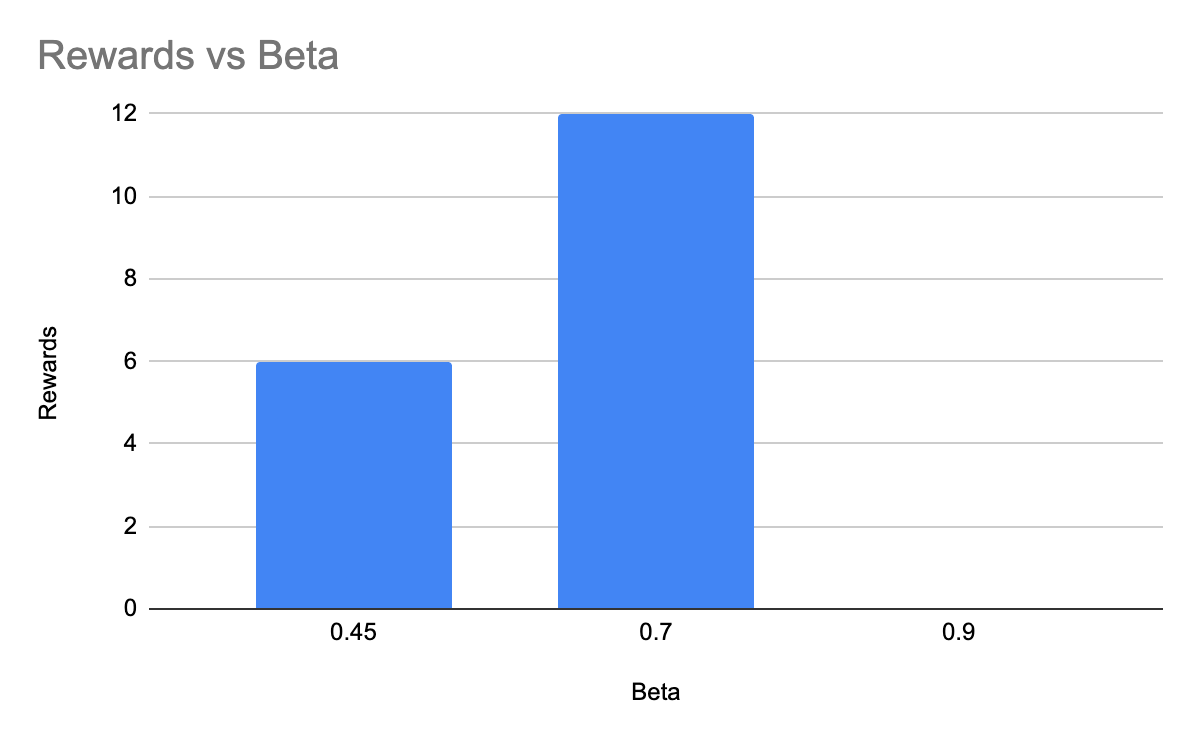
Here Alpha (α) controls the rate at which the algorithm updates the Q-values based on the observed rewards and transitions. A high value of α means that the algorithm will update the Q-values more aggressively, while a low value means that the updates will be more conservative. The choice of α depends on the specific problem and the desired learning rate.

**Inference:**

We can see that we need an optimal value of alpha to get to the final state, a less value of alpha means the algorithms is updating the values aggressively, So in our case 0.3 was optimal value with max rewards.

## Beta vs Rewards - q learning

| Beta | Rewards |
| --- | --- |
| 0.45 | 6 |
| 0.7 | 12 |
| 0.9 | 0 |



Beta is a hyperparameter that controls the weight given to the maximum Q-value in the next state during the update. A high value of beta gives more weight to the maximum Q-value, which can help to stabilize the learning process and prevent oscillations in the Q-values.

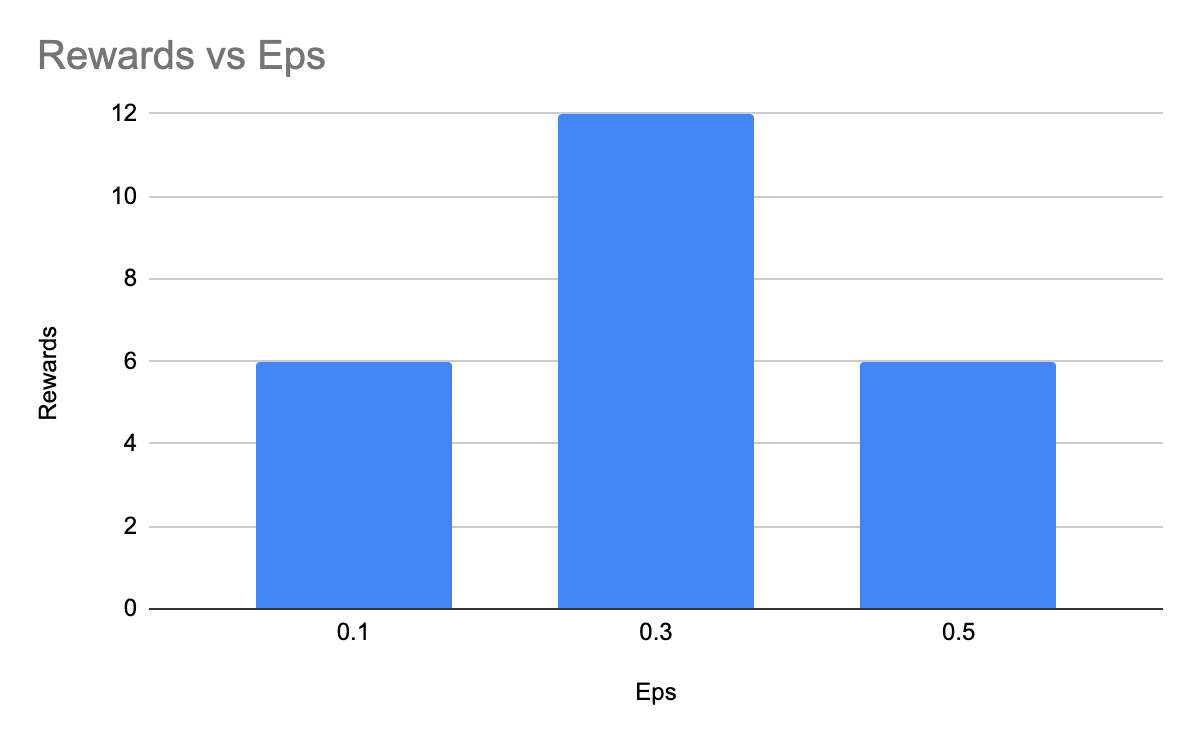
**Inference**

In our cases, we can see that the higher values of beta (0.9) was resulting on a reduced reward, which makes us choose the 0.7 value as the final optimal values

## Eps vs Rewards - q learning

Epsilon (ε) is used in the exploration-exploitation tradeoff, which refers to the balance between exploring new states and actions to learn more about the environment, versus exploiting the current knowledge to make the best decision. Epsilon determines the probability of taking a random action instead of the action with the highest Q-value, which helps the agent to explore different options.

| Eps | Rewards |
| --- | --- |
| 0.1 | 6 |
| 0.3 | 12 |
| 0.5 | 6 |



**Inference**

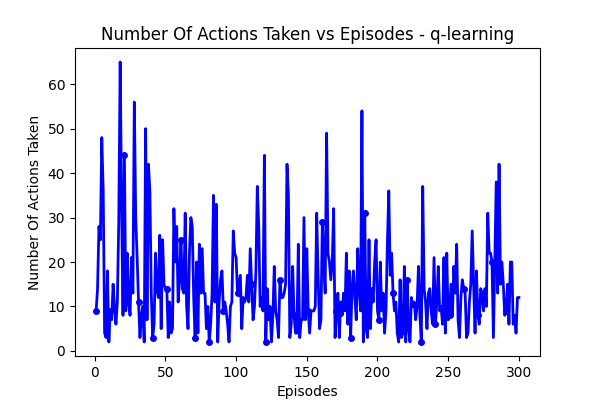
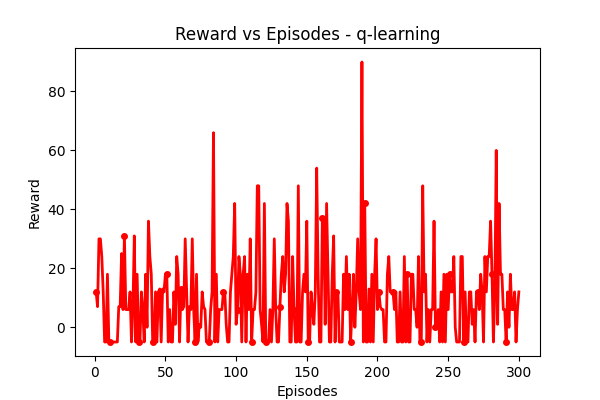
The higher the value of epsilon, the higher it explores the environment by making new decisions as much as possible. However, Lower epsilon means that the model does not explore much. This is clearly visible on the graph above where both extremely low epsilon extremely high values are giving bad results, So we can pick the 0.3 as optimal value.

Based on our Hyperparameter tuning, we can obtain the following Hyperparameters are optimised for best performance.

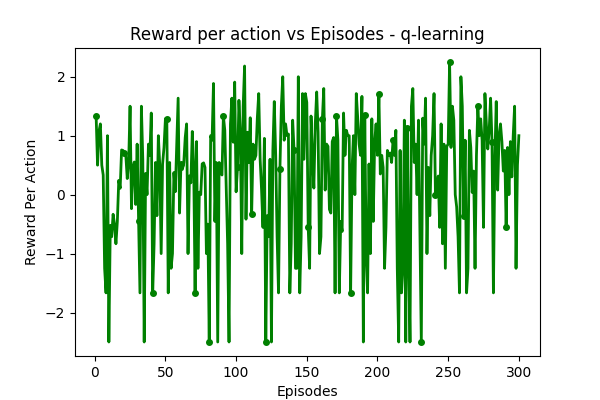
* Alpha - 0.3
* Beta - 0.7
* Eps - 0.3

# Comparison of Models

## Reward vs Episodes and No of Actions - Q Learning



## Episodes vs Rewards Per Action - Q Learning



## 

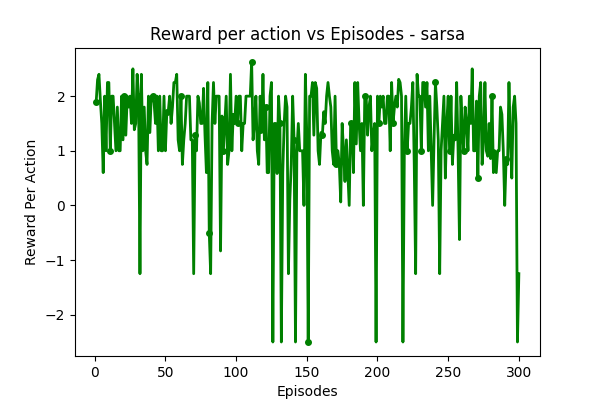
## 

## 

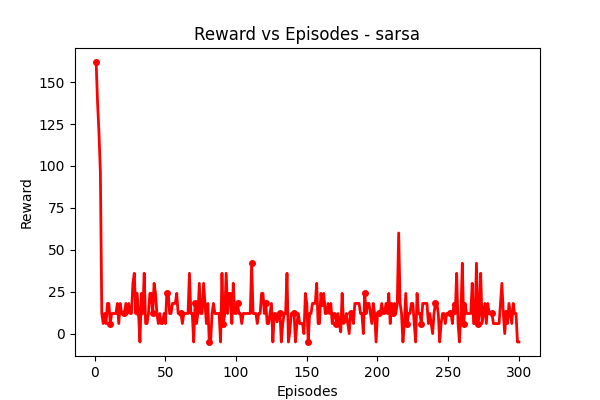
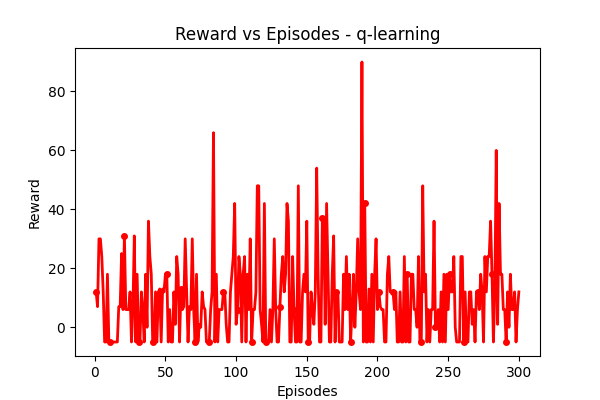
## Reward vs Episodes and No of Actions - SARSA

## 

## Episodes vs Rewards Per Action - SARSA



## SARSA vs Q-Learning



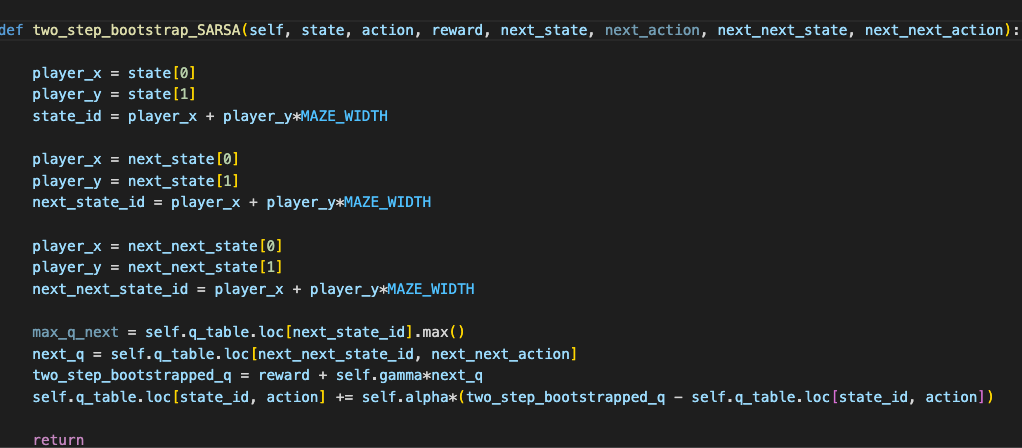
**Inference on Comparison**

SARSA is an on policy algorithm, which means it follows policy by choosing next action based on that policy and updates q table based on the given action. However Q learning is an off policy algorithm. IT learns the optimal policy by updating the Q-values based on the maximum Q-value of the next state and all possible actions. This can make Q-learning more efficient in terms of learning, but it can also make it more sensitive to exploration-exploitation trade-offs and reward shaping.

THis can also be evaluated from our models. For a fixed training episodes, the Qlearning behaved well for a higher Beta value (because it needs to focus more on the existing best value in the table) where as SARSA follows the action to make the decision. However, this can lead to additional exploration from the qlearning side which can make it highly oscillatory as referred from the graph above.

# Bonus -Two Step SARSA

## Implementation



## Graphs

