ASSIGNMENT 3 REPORT

sreejaed - 50475250

sriinith - 50478024

PART I: Defining a RL Environment

1. The environment that we defined is a 4x4 gridworld where an agent can move in four directions up, down, left, or right. The environment contains several objects placed at different locations including a goal, three batteries, and two rocks.

There are 16 states in total and are represented as a 4x4 grid where each cell can have one of the following values:

- 1: the current position of the agent
- 0.5: the location of the goal or battery
- -0.5: the location of the rock

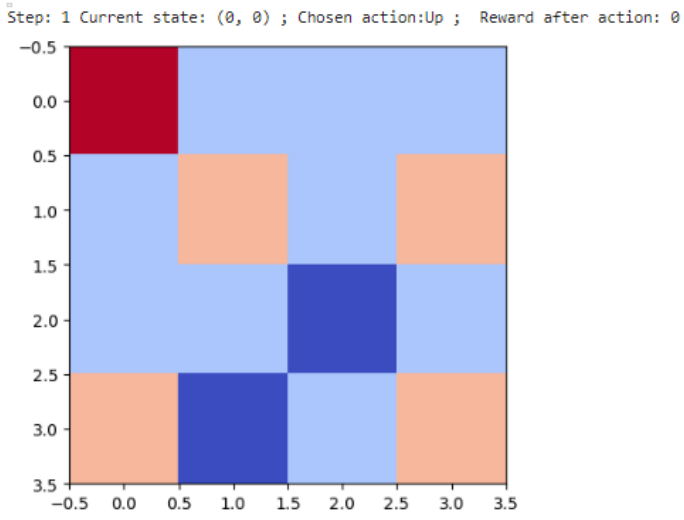
The actions available to the agent are discrete and include moving up, down, left, or right.

The rewards for the agent depend on its current position in the gridworld:

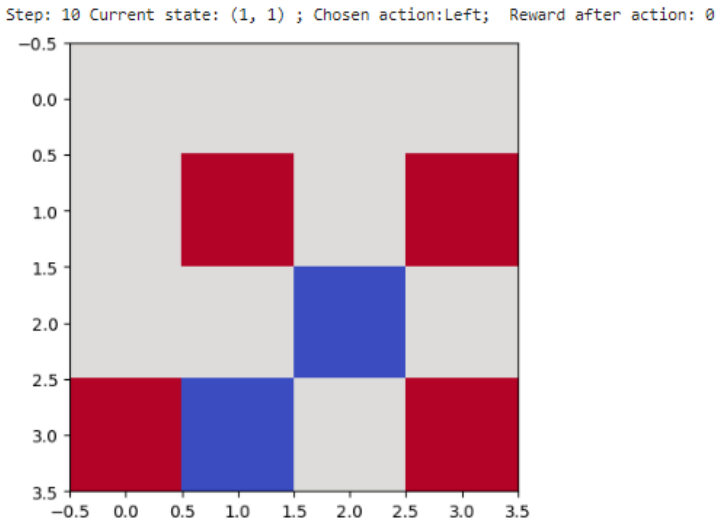
- Moving to the goal yields a reward of 25
- Moving to a battery location yields a reward of 5
- Moving to a rock yields a penalty of –6
- Otherwise, the agent receives a reward of 0

The main objective of the agent is to reach the goal location within a limited number of timesteps while avoiding the rocks and maximizing the reward. The environment allows a maximum of 12 timesteps, and the episode terminates either when the agent reaches the goal or after the maximum number of timesteps is reached.

2. Below is the visualization of our environment while in initial state where the agent position is (0,0)



Below is the visualization of the current state after running the random agent 10 times.



3. The safety of our environment is ensured in following ways:
    - The action space is defined as a discrete space of size 4, representing the four possible actions the agent can take in the environment.
    - Any other actions that are not within this defined action space will be rejected by the environment, ensuring that the agent only chooses actions that are allowed.
    - The agent's position is clipped to the 4x4 state space, ensuring that the agent navigates within the defined state space and does not move outside of the grid.
    - The environment specifies the maximum number of timesteps (12) that the agent can take before the it terminates, ensuring that the agent does not continue to take actions indefinitely, which could potentially result in unsafe behavior.

PART – II & III (SARSA AND Q-Learning Implementation)

1. Tabular methods:

In this project, we have implemented 2 of the tabular based methods namely:

SARSA, and Q-learning. Both methods consist of Q-table whose values are updated following different policies.

SARSA – In this method, the values in the Q-table are updated based on the current state-action pair, the reward received on taking that action, the next state reached, and the next action chosen depending on the policy. Hence, it's name: S (state), A (Action), R (Reward), S (next state), A(next action).

Q- Learning: This algorithm works by estimating the maximum rewards that are expected in the future for each state-action pair, irrespective of the policy that is followed during the learning stage.

The update function for SARSA can be represented as:

$Q(S,A) = Q(S,A) + \alpha * [R + \Upsilon * Q(S', A') - Q(S,A)]$

Where:

Q(S,A) : Represents Q value upon taking an action A and state S.

α : Learning Rate

R : immediate reward after performing action A from state S.

ϒ : Discount Factor

Q(S', A') : Represents the Q value of the next state-action pair.

The update function for Q- Learning can be represented as:

$Q(S,A) = Q(S,A) + \alpha * [R + \Upsilon * \max Q(S', A') - Q(S,A)]$

Q(S,A) : Represents Q value upon taking an action A and state S.

α : Learning Rate

R : immediate reward after performing action A from state S.

ϒ : Discount Factor

maxQ(S', A') : Represents the maximum Q value among all possible actions of the next sate.

Key features of SARSA:

- SARSA implements epsilon-greedy policy that balances the tradeoff between exploration and exploitation.
- It implements the TD (Temporal Difference) learning that updates the Q-values based on the target Q-value and the estimated Q-value.

Key Features of Q-Learning:

- Q-learning is an off-policy RL algorithm that learns Q-values by estimating the maximum expected future rewards (i.e maxQ(S',A')) regardless of the policy being followed.
- Q-Learning also implements TD (Temporal Difference) learning that updates Q-values based on the estimated Q-value and target Q-value.

Advantages:

- SARSA guarantees convergence given the condition that sufficient exploration is done.
- SARSA is proven to perform well in stochastic environments (i.e when the outcome of taking an action in a particular state is a single state)
- Q-learning is optimal in reinforcement learning environments where there is uncertainty about the rewards.
- Q-learning can be applied to both simple problems like grid environment to complex control tasks.

Disadvantages:

- SARSA may sometimes require many numbers of samples for it to learn and converge. This can lead to slow learning rate when implemented in complex environments.
- Q-learning sometimes is proven to over – estimate Q-values especially when the model is in it's early stage of learning.

2. RESULTS:

- Applying SARSA to solve the environment defined in Part 1. Plots should include epsilon decay and total reward per episode.

  Hyperparameter values:

  Learning rate: 0.15

  Epsilon: 0.99

  Discount factor: 0.95

  Timesteps: 15

  Number of episodes: 2000

EPSILON DECAY CURVE:

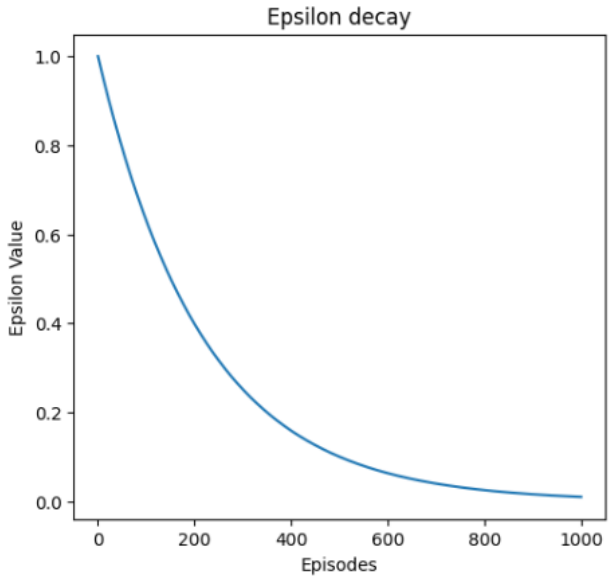


Fig: Epsilon Decay Curve: SARSA

Analysis: From the above plot, we can infer that, the epsilon decay is exponential in nature. In general, the curve depicts how exploration changes over time (i.e starting from 1st episode to going until the maximum number of episodes)

REWARDS PER EPISODE:

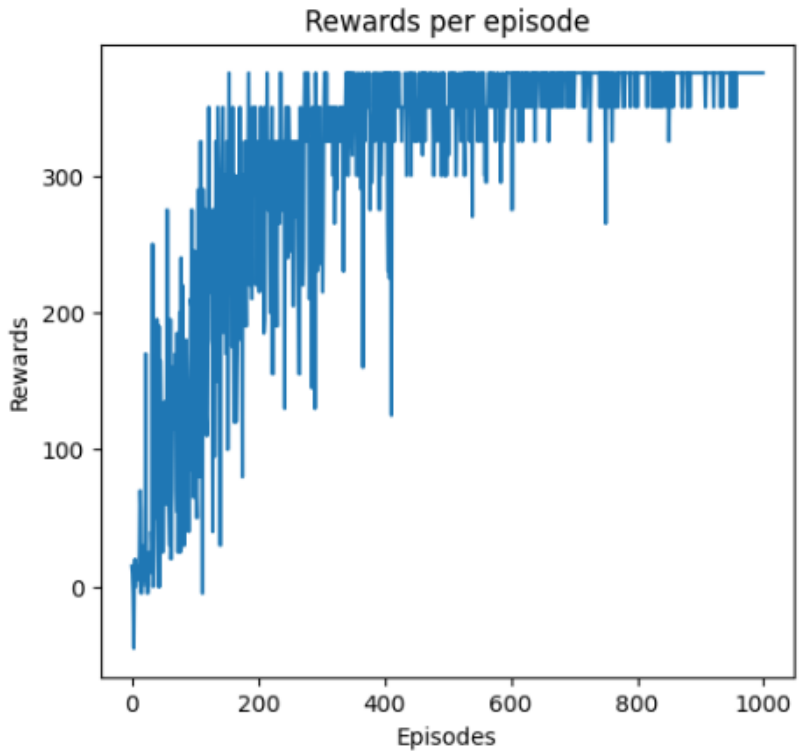


Fig: Rewards per episode: SARSA

Analysis: From the above reward graph of SARSA, we can infer that the SARSA algorithm is converging as it is reaching the maximum number of episodes that we have mentioned. During the initial stage of learning, we can see that there is a lot of fluctuations in the rewards obtained per episode.

- Applying Q-learning to solve the environment defined in Part 1. Plots should include epsilon decay and total reward per episode.
- **Hyperparameter values:**
- **Learning rate: 0.15**
- **Epsilon: 1**
- **Discount factor: 0.99**
- **Timesteps: 10**
- **Number of episodes: 1500**

EPSILON DECAY CURVE:

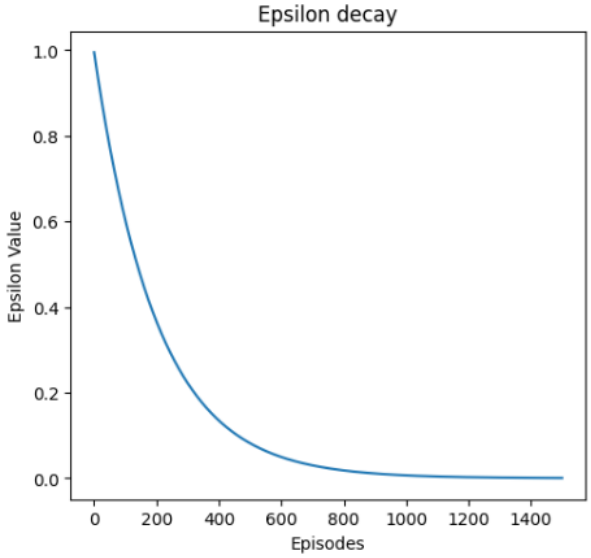


Fig: Epsilon Decay curve: Q – Learning

Analysis: From the above graph, we can infer that the epsilon decay rate is exponential in nature. We see that the epsilon value is decreasing rapidly during the initial learning stage, whereas after that, it almost remained constant.

REWARDS PER EPISODE:

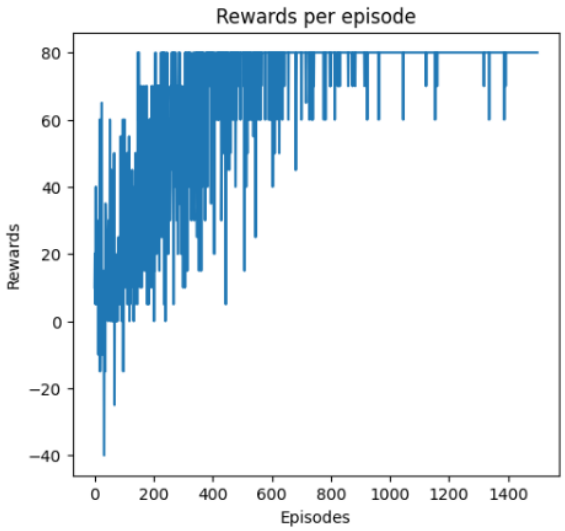
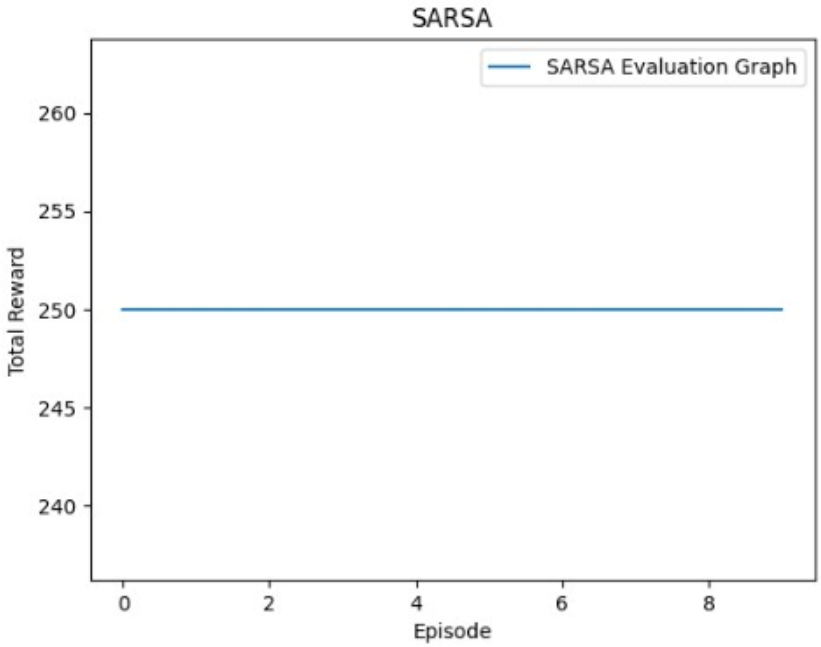


Fig: Rewards per episode: Q – Learning

Analysis: From the above plot, we can conclude that the Q – Learning algorithm is converging (i.e as the number of episodes are increasing and reaching the maximum value, we can see that the rewards are constant that means that the algorithm is constant.)

- Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Plot should include the total reward per episode.




```
SARSA Evaluation Results:
Average Total Reward: 250.0
Maximum Total Reward: 250
Minimum Total Reward: 250
```

Fig: Evaluation result: SARSA

Analysis: Here we have run the environment for 10 episodes, and we observe that the rewards reach a maximum of 250.

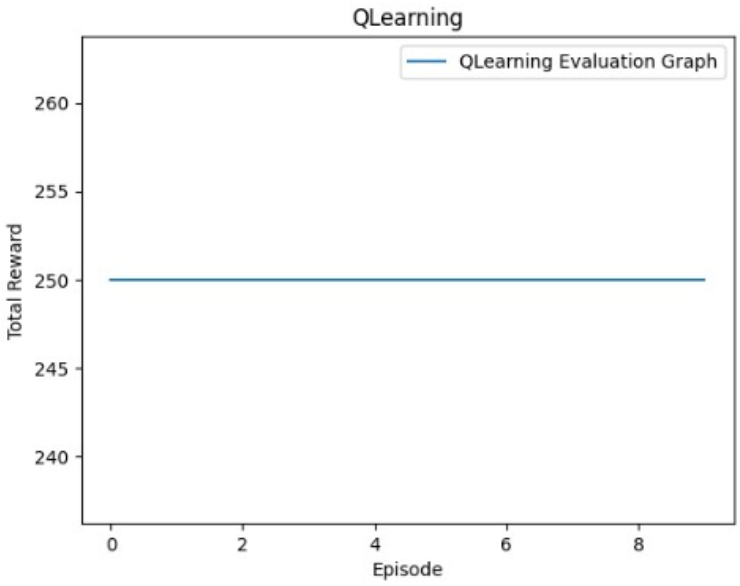


Fig: Evaluation result: Q-Learning

Analysis: Here we have run the environment for 10 episodes, and we observe that the rewards reach a maximum of 250.

3. Comparison of the performance of SARSA and Q-Learning

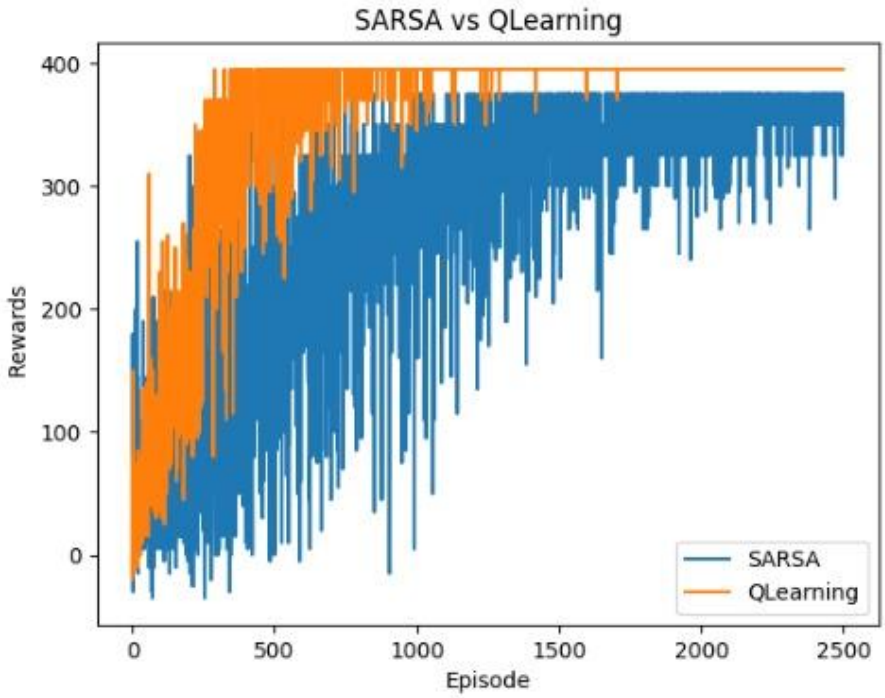


Fig: performance (SARSA vs Q-learning)

Analysis: The above plot represents the performance of SARSA, and Q-learning set under the same environment. We can interpret that as the number of episodes are increasing the Q-Learning algorithm is converging but SARSA is not converging. Also, we observe that the over the entire range of episodes, the rewards obtained per episode are fluctuating a lot compared to the Q-learning algorithm.

4. Hyper parameter tuning (SARSA):

Setup – I (Tuning parameter – Discount factor)

- Discount Factor: 0.89

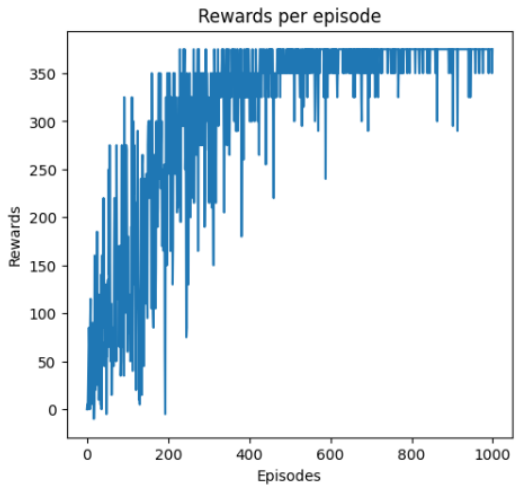


Fig: Rewards per episode – SARSA (discount factor = 0.89)

Analysis: By varying the discount factor while maintaining the other parameters like timesteps, and the number of episodes as constants, it is observed that the SARSA algorithm is not converging as even when it is reaching the maximum number of episodes.

- Discount Factor: 0.7

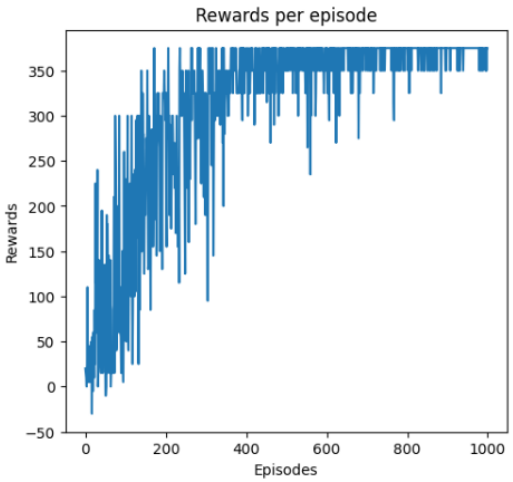


Fig: Rewards per episode – SARSA (discount factor = 0.7)

Analysis: In this plot too, by varying the discount factor to 0.7 while maintaining the other parameters like timesteps, and the number of episodes as constants, it is observed that the SARSA algorithm is not converging even when it is reaching the maximum number of episodes.

- Discount Factor: 0.5

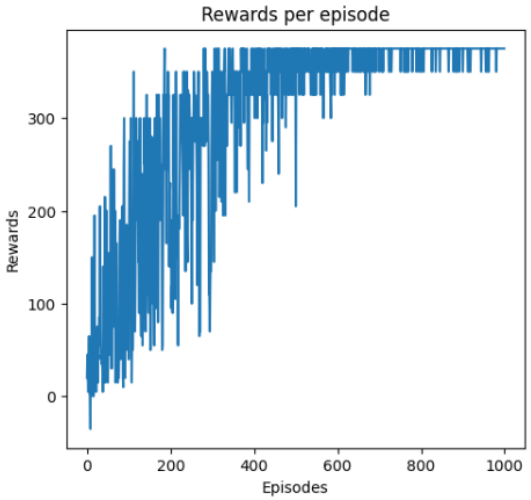


Fig: Rewards per episode – SARSA (discount factor = 0.5)

Analysis: In this plot too, by varying the discount factor to 0.5 while maintaining the other parameters like timesteps, and the number of episodes as constants, it is observed that the SARSA algorithm is not converging even when it is reaching the maximum number of episodes.

Setup – II (Tuning parameter – Number of episodes)

- Number of episodes: 1500

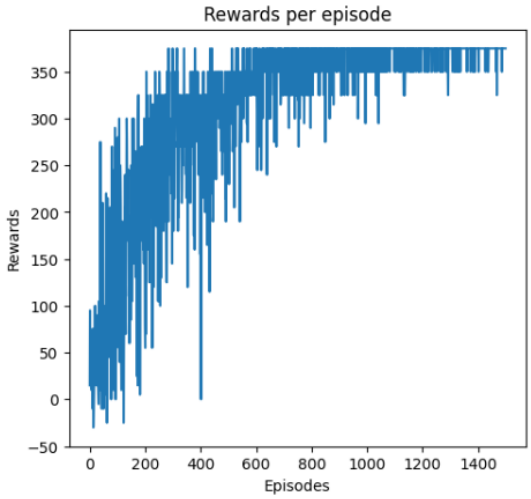


Fig: Rewards per episode – SARSA (Number of episodes = 1500)

Analysis: Varying the number of episodes while maintaining the other parameters like discount factor and timesteps as constants, it is observed that the SARSA algorithm is not converging as even when it is reaching the maximum number of episodes

- Number of episodes: 5000

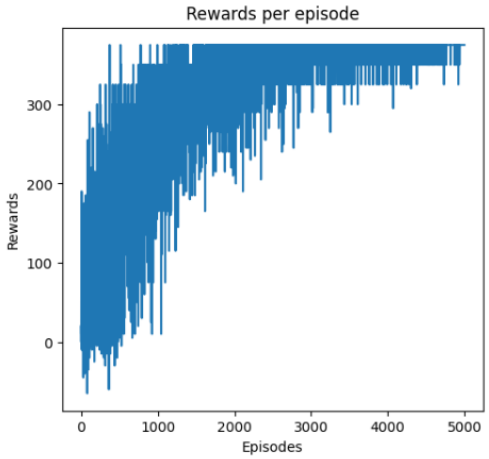


Fig: Rewards per episode – SARSA (Number of episodes = 5000)

Analysis: It is observed that the SARSA algorithm is not converging as even when it is reaching the maximum number of episodes. Additionally, we see a lot of fluctuations in the rewards per episode until the very end.

- Number of episodes: 2000

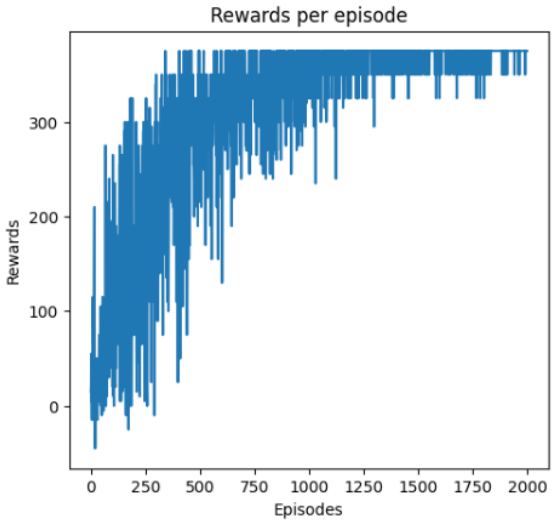


Fig: Rewards per episode – SARSA (Number of episodes = 2000)

Analysis: In this plot too, we observe that the SARSA algorithm is not converging as even when it is reaching the maximum number of episodes. Though there reward fluctuation rate has come down as the number of episodes increased, still it is not converging.

MOST EFFICENT HYPER PARAMETER SETUP:

From the above 2 setups, we can conclude that we got a better result (as the algorithm is converging) in our based model. The values of hyper parameter in our based model are

- discount factor: 0.95
- timesteps = 15
- number of episodes = 1000

4. Hyper parameter tuning (Q - Learning):

Setup – I (Tuning parameter – Discount factor)

- Discount Factor: 0.95

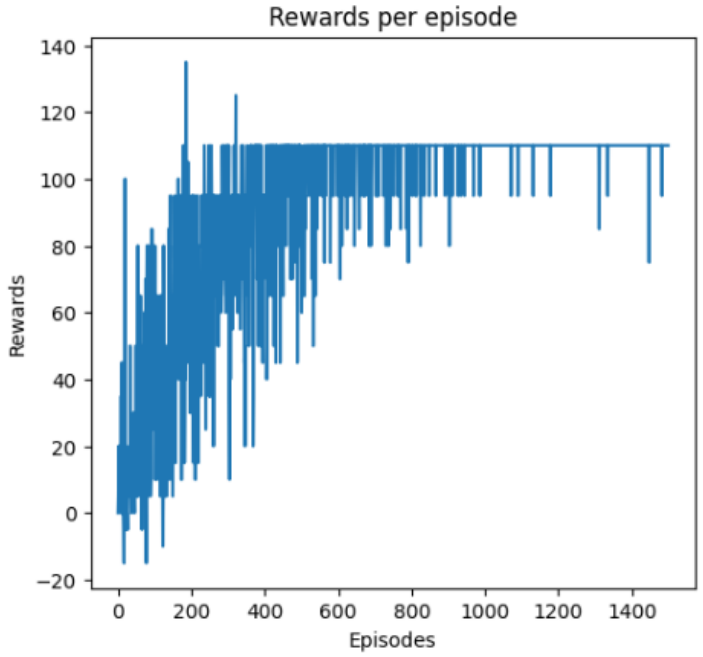


Fig: Rewards per episode – Q-Learning (discount factor = 0.95)

Analysis: From the above graph of rewards per episode of Q-Learning, on varying the discount factor and maintaining all the other parameters as constants, we observe that the Q-learning algorithm is not converging.

- Discount Factor: 0.90

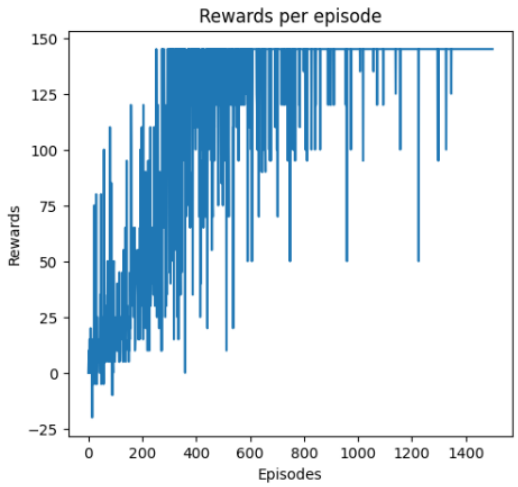


Fig: Rewards per episode – Q-Learning (discount factor = 0.90)

Analysis: From the above graph we observe that we observe that the Q-learning algorithm is converging. We notice that by reducing the discount factor by 0.05, we observe that the algorithm is converging.

- Discount Factor: 0.89

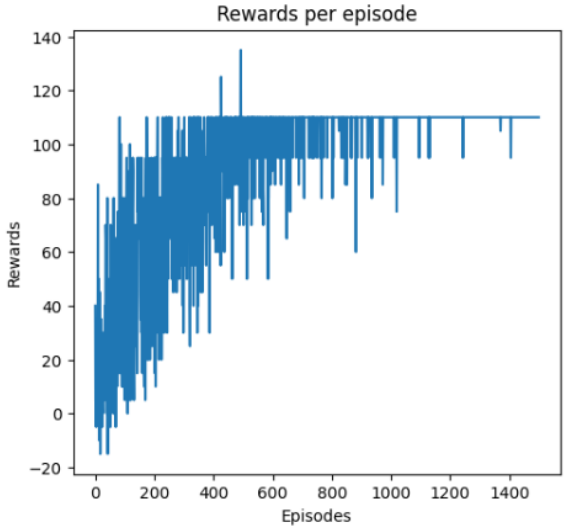


Fig: Rewards per episode – Q-Learning (discount factor = 0.89)

Analysis: From the above graph we observe that we observe that the Q-learning algorithm is converging when the number of episodes are 1500.

Setup – II (Tuning parameter –Timesteps)

- Timesteps: 30

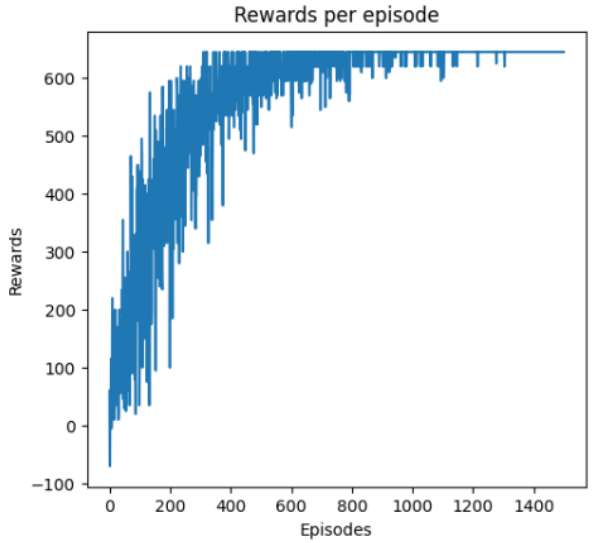


Fig: Rewards per episode – Q-Learning (Time steps= 30)

Analysis: From the above graph of rewards per episode of Q-Learning, on varying the timesteps and maintaining all the other parameters such as discount factor and number of episodes as constants, we observe that the Q-learning algorithm is converging.

- Timesteps: 25

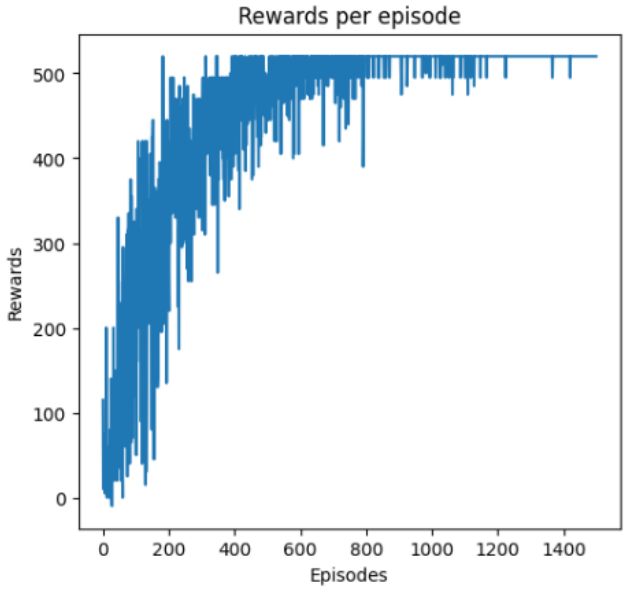


Fig: Rewards per episode – Q-Learning (Time steps= 25)

Analysis: We observe that the Q-learning algorithm is converging when the discount factor and number of episodes are constant and the timestep is set to 25.

- Timesteps: 20

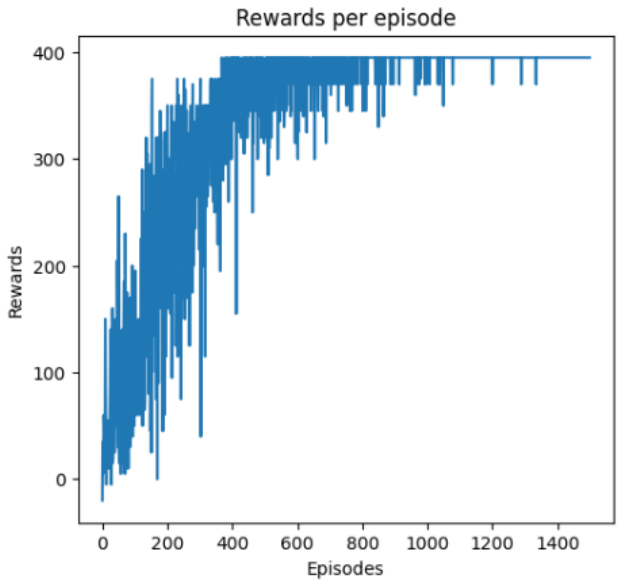


Fig: Rewards per episode – Q-Learning (Time steps= 20)

Analysis: We observe that the Q-learning algorithm is converging as the number of episodes are increasing, when the discount factor and number of episodes are constant and the timestep is set to 20.

MOST EFFICIENT HYPER PARAMETER SETUP:

From the above 2 setups, we can conclude that we got a better result (as the Q- Learning algorithm is converging) in setup – II when the number of timesteps are 30. The values of hyper parameter in our based model are

- discount factor: 0.90
- timesteps = 30
- number of episodes = 1500

REFERENCES:

- spring_23_rl_random_agent (4).ipynb - Colaboratory (google.com)
- SuttonBartoIPRLBook2ndEd.pdf (stanford.edu)
- Gymnasium Documentation (farama.org)
- https://towardsdatascience.com/implement-grid-world-with-q-learning-51151747b455
- https://medium.com/analytics-vidhya/grid-a-grid-world-environment-based-on-openai-gym-725189dcada6
- https://scholar.uwindsor.ca/cgi/viewcontent.cgi?article=9763&context=etd
- https://www.researchgate.net/publication/326634150_Implement_Deep_SARSA_in_Grid_World_with_Changing_Obstacles_and_Testing_Against_New_Environment_The_Selected_Papers_of_The_First_International_Conference_on_Fundamental_Research_in_Electrical_Engineering

CONTRIBUTION:

Team Member	**Assignment Part**	**Contribution**
Sreeja Edupuganti	PART – II, PART – III, Report – PART I & III	50%
Sriinitha Chinnapatlola	PART – I, PART – III, Report – PART II & III	50%