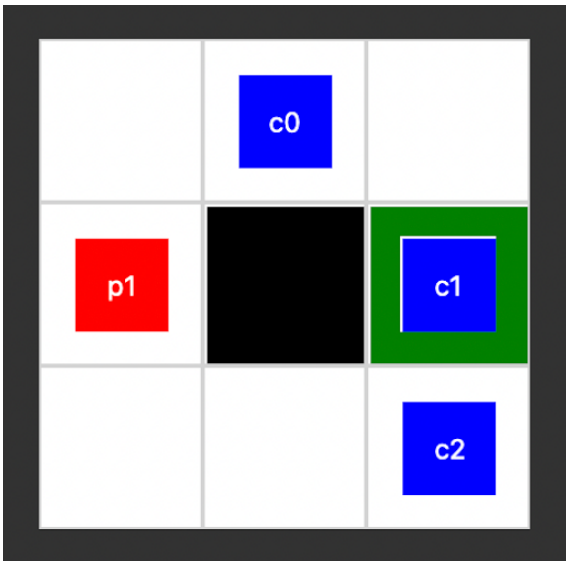
CMPSCI 687 - CRATE SLIDING PROBLEM

Harsh Seth and Mihir Thalanki

1 Problem Statement

You are given a grid-based environment representing a room. The environment contains various elements: pitchforks (in red), crates (in blue), obstacles (in black) and a target state (in green). The goal is to navigate the pitchfork in the fewest possible steps to the target location. Each cell within the grid can accommodate only one entity (either a pitchfork or a crate). Cells with obstacles cannot accommodate any entity. Movement within the grid is limited to four directions: up, down, right, left. Any pitchfork or crate can be moved in this direction to an adjacent cell as long as it is not obstructed by an obstacle or occupied by another entity. Our objective is to optimize the pathfinding strategy to minimize the number of steps taken to guide the pitchfork to the target location while avoiding obstacles and crates.



2 MDP Formulation

We identified several key features that make it conducive to represent this problem as an MDP:

- **Discrete states and actions:** The grid-based environment allows for discrete states and actions. This simplifies the state and policy representation, and makes it easier to model transitions.
- **Markovian Property:** The environment satisfies the Markov Property. This is because the future state depends only on the current state and not the entire history of states and actions leading up to this point.

We formulated the MDP with the following components:
State:
A state is represented by a list of tuples, where each item in the list represents the position of an entity in the grid. The states therefore contain the positions of every pitchfork and crate in the grid For example, the state of the environment in Fig 1 is: ((0,1), (1,0), (1,2), (2,2)). Each tuple element (x,y) signifies the position of the entity in the grid. The positions configured here correspond to the entities specified in a list [‘p1’, ‘c0’, ‘c1’, c2’].
Action:
An action is represented with a tuple (e, d), where e signifies the entity ID and d represents the direction the entity is being moved. The direction has to be chosen from [‘U’, ‘D’, ‘R’, ‘L’] representing the 4 possible

directions. For example, an action ('p1', 'U') moves the pitchfork 'p1' upward.
Transition Probabilities:
We have assumed the environment's transition dynamics to be deterministic. For a given entity, If the cell in the direction of the action is unoccupied, the probability of the entity moving to the new cell is 1. If however, the cell is occupied with another entity or obstacle, the probability is 0. For example, if 'p1' (pitchfork) moves upward 'U', and the cell above it is unoccupied, it transitions to that cell with probability 1. Additionally, if an entity moves into a wall, it remains in the same state.
Reward:
In our environment, the reward function can be easily adapted. We have used a default reward function of -1 for every move. This happens till the state reaches a terminal state. A terminal state is a state in which any of the pitchforks have reached the goal state.

3 Assumptions

- Each entity in the environment occupies only a single block of space
- Each moveable entity can move in all 4 directions
- The number of pitchforks and crates is constant
- The number and positions of obstacles and goal states are constant

4 Implementation of the domain

To implement the domain we used Python. We created a class EnvMap that encapsulates the states, actions, and dynamics of the environment. We ensured that our code is modular and extendable. The code is extremely flexible and allows us to easily adapt it to represent different grid configurations.
States:

- The class stores and manages the states of all the distinct components of the environment. Specifically, it maintains and updates the states of OBSTACLE_STATES, CRATE_STATES, PITCHFORK_STATES, GOAL_STATES. Each of these are dictionaries with the entity ID as the key and its current position as the value.
- Note that OBSTACLE_STATES and GOAL_STATES do not change through the object's lifespan. Conversely, CRATE_STATES and PITCHFORK_STATES change as these entities are movable.
- Therefore the state of the agent can be represented with CRATE_STATES and PITCHFORK_STATES
- The getCurrentState() function creates a state representation in the form of a list of tuples. Each tuple represents the position of a movable entity. The order of these tuples is determined by the order of the entities in the MOVABLE_ENTITIES list. This also enables easy retrieval of entity positions based on the current state.

Actions:

- Actions are represented as tuples, denoted by (entityID, direction). entitiyID indicates the ID of the entity upon which the action is performed. Direction indicates the direction of movement for the entity. It can either be 'U', 'D', 'R', 'L'

Reward:

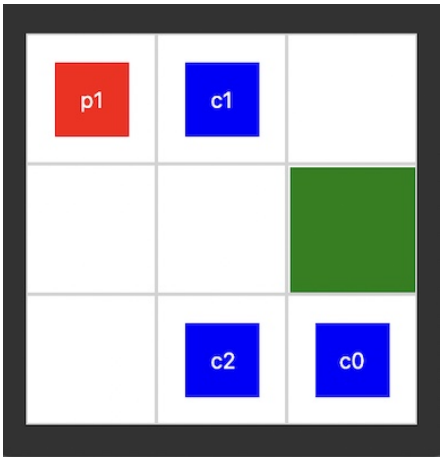
- The class has variables GOAL_REWARD and MOVE_REWARD that can be customized for fine-tuning. GOAL_REWARD specifies the reward obtained when a pitchfork reaches the goal position. MOVE_REWARD specifies the reward obtained when the agent takes an action.
- The getReward() function is responsible for determining the reward given to the agent. This reward is calculated using just the current state. This function returns GOAL_REWARD or MOVE_REWARD based on the current state of the agent.

5 Variations

To comprehensively evaluate the performance of our algorithm implementations, we conducted experiments on various configurations of the environment. We ensured the range covers different levels of difficulty. To ensure consistent experimental outcomes, we employed a fixed random seed across all trials. This allows us to directly compare the performance of various algorithms.

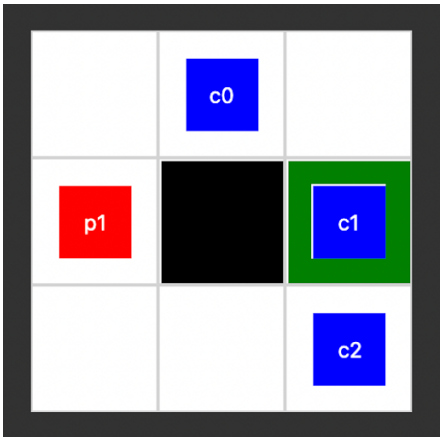
5.1 Basic Variation

This variation is a 3x3 grid with 3 crates with the goal state at (1,2) and 3 crates randomly allocated in available positions. 1 pitchfork is initialized randomly on the first column of the grid. This variation has 3024 states.



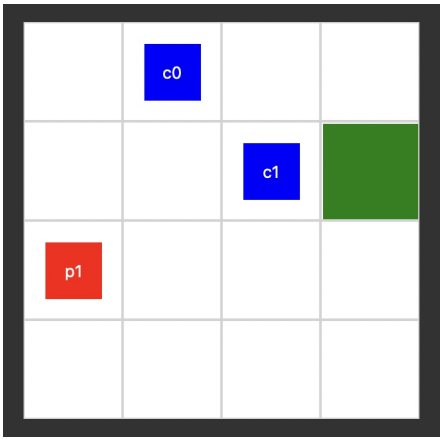
5.2 Obstacle Variation

This variation is a 3x3 grid with 3 crates with an obstacle at position (1,1) and the goal state is at (1,2).



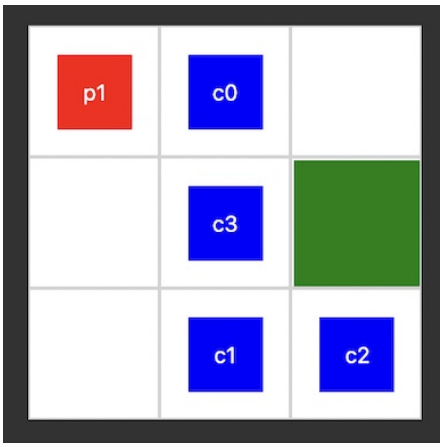
5.3 Basic - Large and Simple

This variation is a 4x4 grid with 2 crates and the goal state is at (1,3). This variation has 3360 states



5.4 Basic - Complex

This variation is a 3x3 grid with 24crates and the goal state is at (1,2). This variation has 15120 states



6 Value Iteration

The hyperparameters here are: Discount γ and δ

- Delta denotes the max norm threshold below which, we say that the value function has converged. After running several experiments, we concluded that $\delta = 0.001$ adequately captured the convergence of the value iteration algorithm. At this threshold, the algorithm reached satisfactory results.
- For the below experiments, the obstacle variation was chosen. We chose gamma=0.7. The reward is -1 for moving and 0 for reaching the goal. We also initialized the values to 0 initially

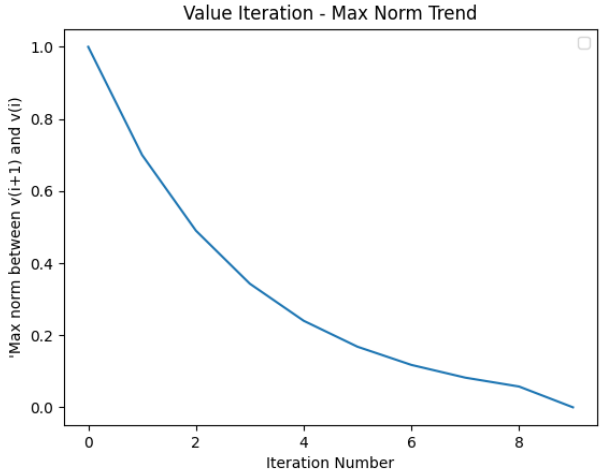


Figure 1: Max norm plotted over iterations using value iteration

- The convergence of the value iteration algorithm was investigated across multiple gamma values. The plot below illustrates this. It can be seen that $\gamma = 0.1$ converges the fastest. This means that emphasizing on short-term rewards allowed the algorithm to converge faster.

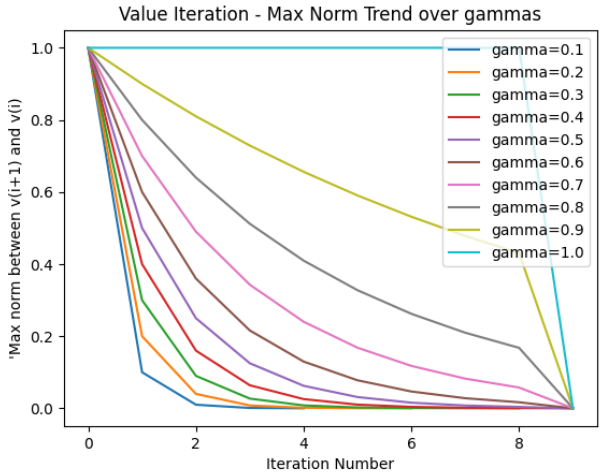


Figure 2: Max norm plotted over iterations with varying discount rates

- To evaluate the impact of different gamma values on the overall return, we plotted the average return over 10 iterations for several gamma values. For every iteration, a random initial state was chosen, and run over the optimal policy generated by the different gamma values. $\gamma = 0.7$ performed the best.

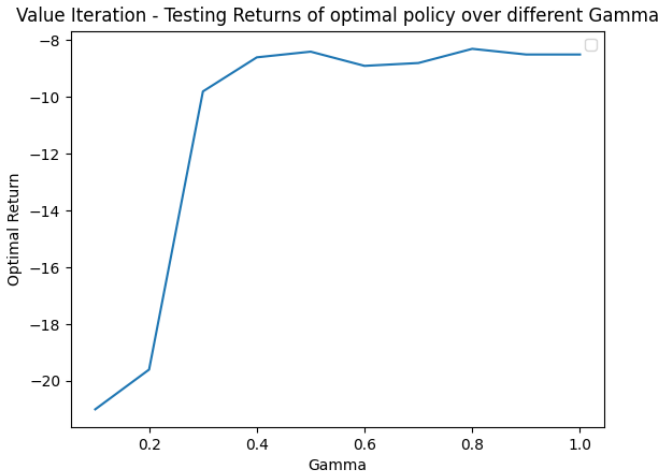


Figure 3: Average return over varying discount rates

From these experiments, we have tuned the hyperparameters to:

Hyperparameter	**Value**
Discount	0.7
Delta	0.001

6.1 Results

With MOVE_REWARD = -1 and GOAL_REWARD = 10

Variation	**Return**
Basic Variation	7
Obstacle Variation	0
Basic - Large and Simple Variation	6
Basic - Complex	5

7 Q Learning

Hyperparameters used for experimentation:

- Alpha: 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1, 9e-1, 1
- Discount Parameter: 1, 0.9, 0.75, 0.5, 0.25, 0.0
- Initial Q Values: 0.0, 50, -50, 5e3, -5e3, 5e5, -5e5
- Rewards: -1/0, 0/10, -1/10, -5/10

Metrics Used for Evaluation

- Cumulative actions by episode counts
- Steeper curve indicates better learning

Configuration used: Alpha = 1; Discount = 1; Move Reward = 0; Goal Reward = 10;(0/10) Q values = 0.0 The trial results were computed by averaging over 10 samples, 1000 episodes each) Below plots are created from the Obstacle Variation:

7.1 Variations in alpha

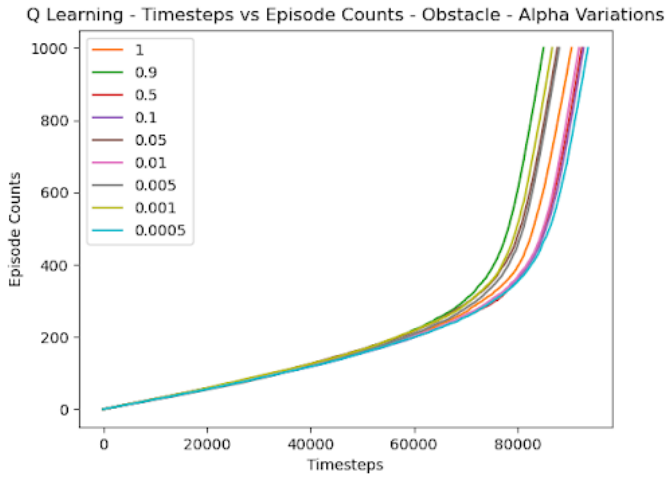


Figure 4: Plot of episode count vs timestamp over varying alpha values

7.2 Variations in discount parameter

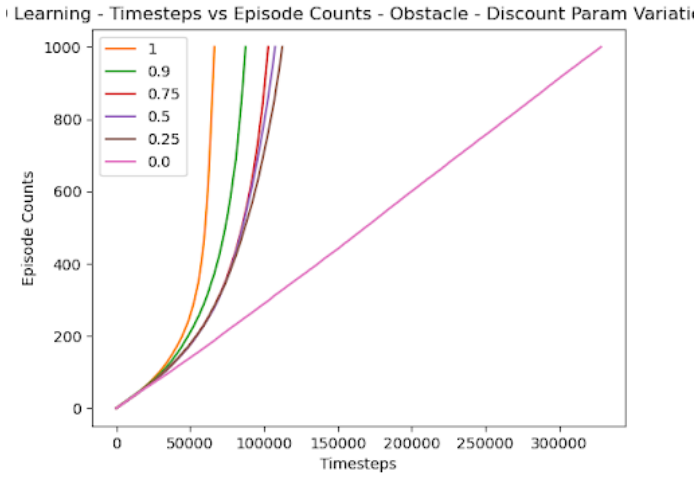


Figure 5: Plot of episode count vs timestamp over varying gamma values

7.3 Variations in Initial Q Values

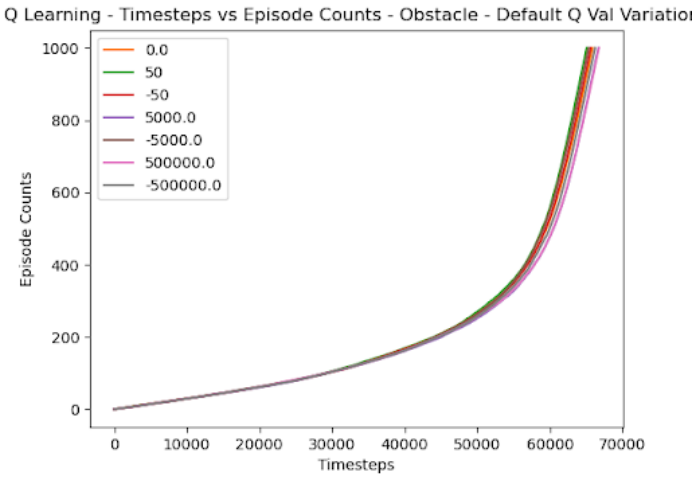


Figure 6: Plot of episode count vs timestamp over varying initial q values

7.4 Variations in Reward Specification

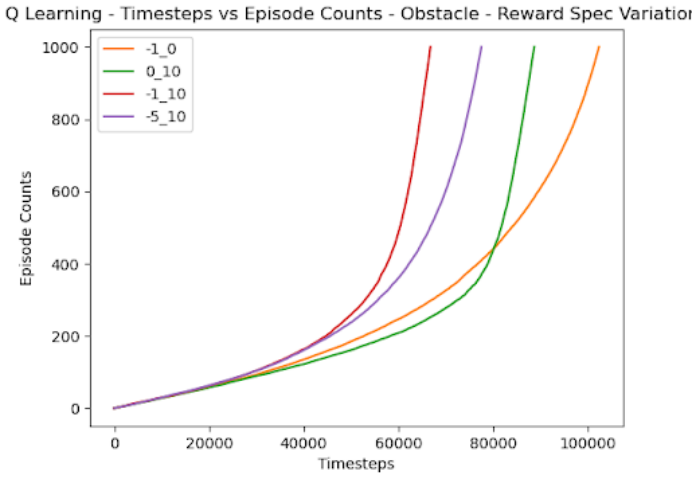


Figure 7: Plot of episode count vs timestamp over varying reward specifications

7.5 Final Results

From these experiments, we have tuned the hyperparameters to:

Hyperparameter	**Value**
Alpha	0.9
Epsilon	Decaying 1/episode count
Discount Parameter	1
Initial Q Values	50
Reward	-1/10

The graphs below were generated by averaging over 20 samples, 1000 episodes each.

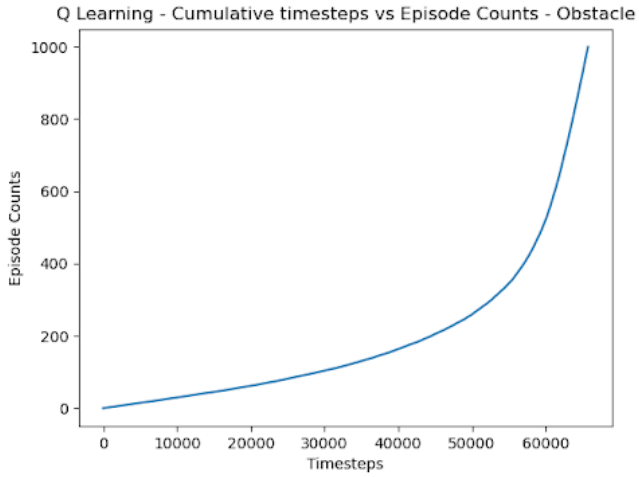


Figure 8: Learning curve

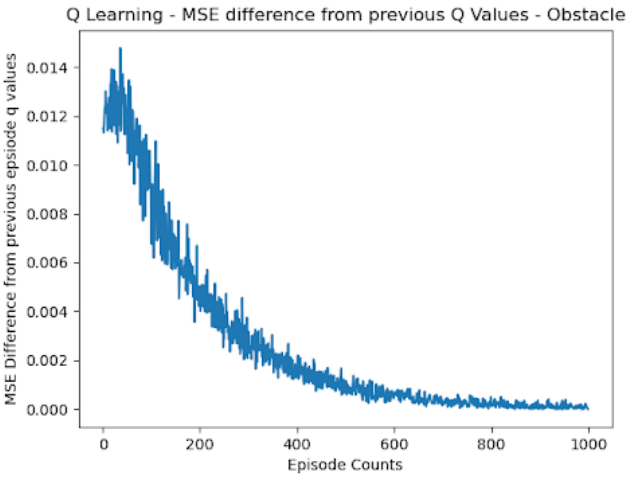


Figure 9: MSE curve

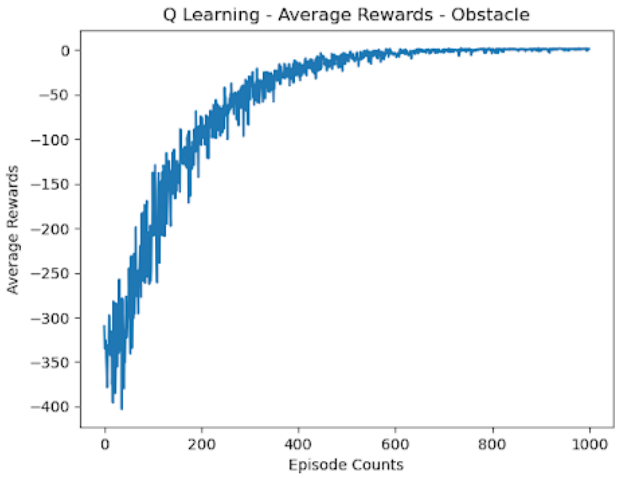


Figure 10: Average returns over episodes

Variation	**Return**
Basic Variation	4
Obstacle Variation	0
Basic - Large and Simple Variation	-1
Basic - Complex	-17

8 SARSA

Hyperparameters used for experimentation:

- Alpha: 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1, 9e-1, 1
- Discount Parameter: 1, 0.9, 0.75, 0.5, 0.25, 0.0
- Initial Q Values: 0.0, 50, -50, 5e3, -5e3, 5e5, -5e5
- Rewards: -1/0, 0/10, -1/10, -5/10

Configuration used: Alpha = 5e-1; Discount = 1; Move Reward = 0; Goal Reward = 10;(0/10) Q values = 0.0 The trial results were computed by averaging over 10 samples, 1000 episodes each) Below plots are created from the Obstacle Variation:

8.1 Variations in alpha

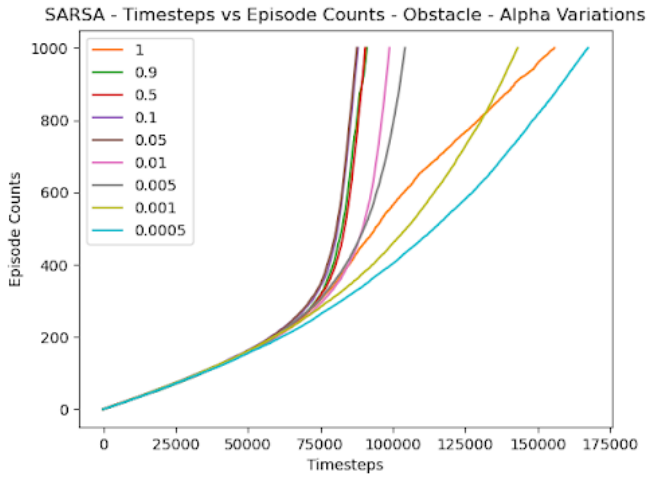


Figure 11: Plot of episode count vs timestamp over varying alpha values

8.2 Variations in discount parameter

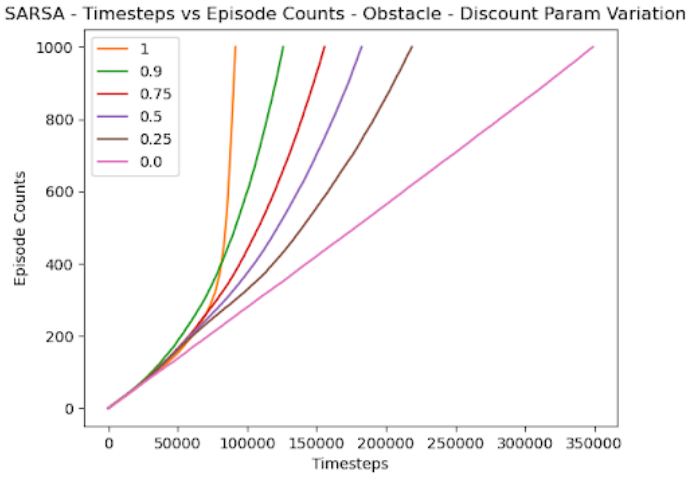


Figure 12: Plot of episode count vs timestamp over varying gamma values

8.3 Variations in Initial Q Values

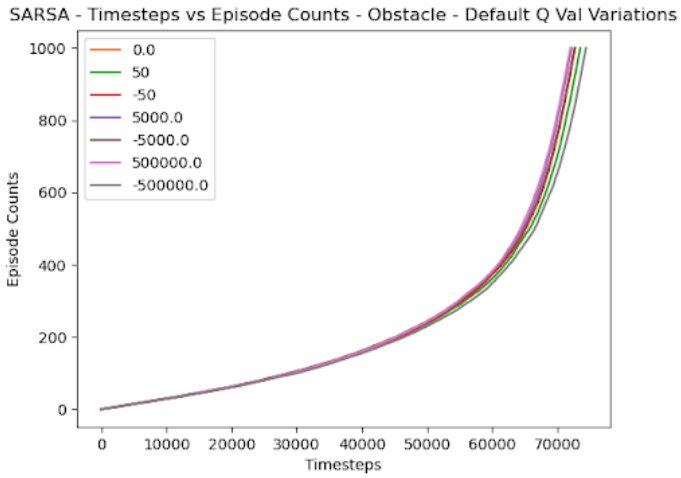


Figure 13: Plot of episode count vs timestamp over varying initial q values

8.4 Variations in Reward Specification

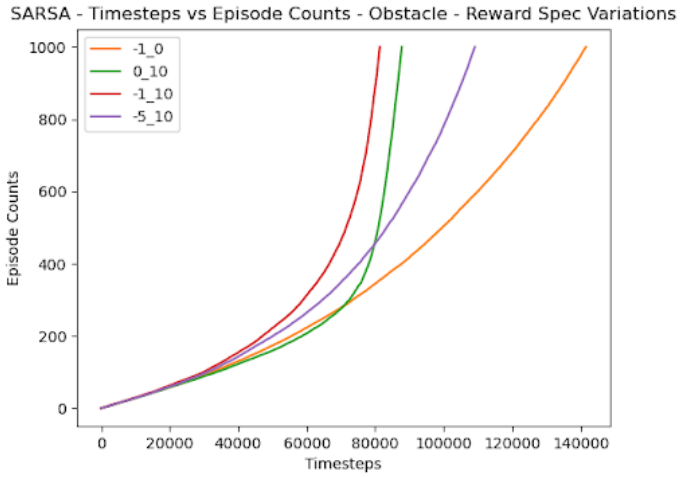


Figure 14: Plot of episode count vs timestamp over varying reward specifications

8.5 Final Results

From these experiments, we have tuned the hyperparameters to:

Hyperparameter	**Value**
Alpha	5e-1
Epsilon	Decaying 1/episode count
Discount Parameter	1
Initial Q Values	5e-3
Reward	-1/10

The graphs below were generated by averaging over 20 samples, 1000 episodes each.

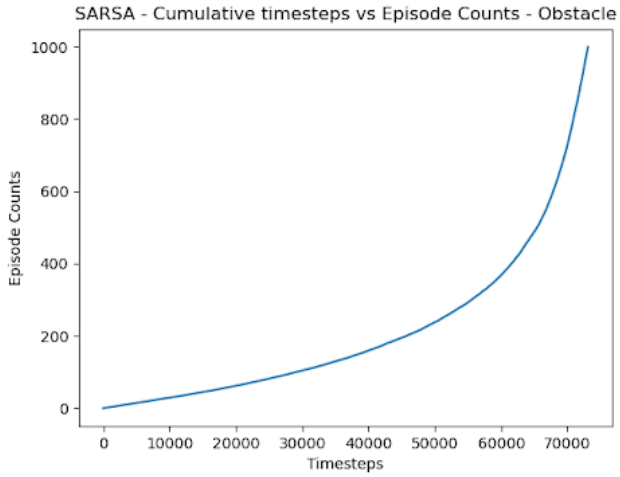


Figure 15: Learning curve

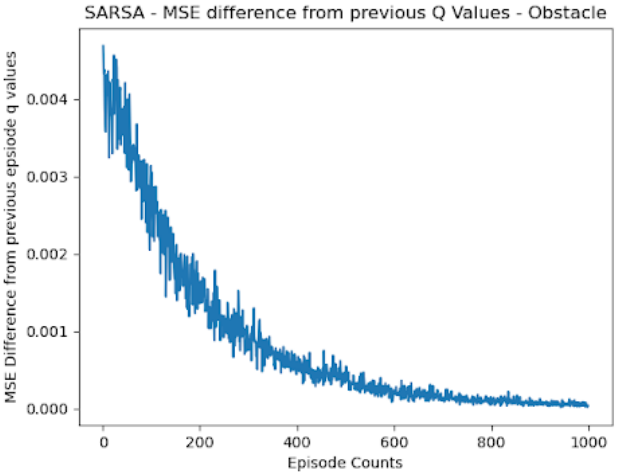


Figure 16: MSE curve

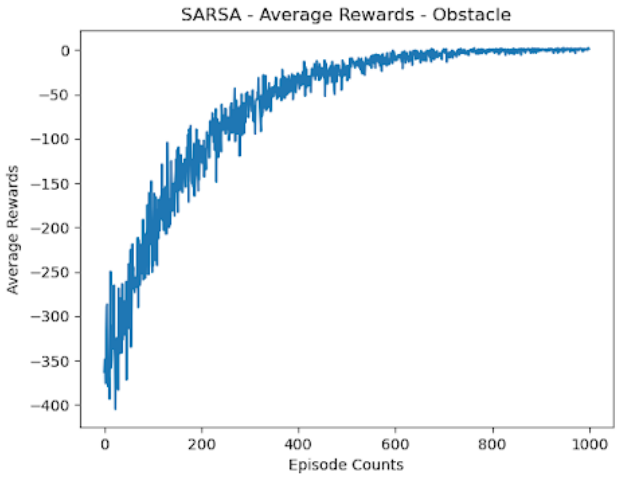


Figure 17: Average returns over episodes

Variation	**Return**
Basic Variation	7
Obstacle Variation	0
Basic - Large and Simple Variation	2
Basic - Complex	-31

9 SARSA with eligibility traces

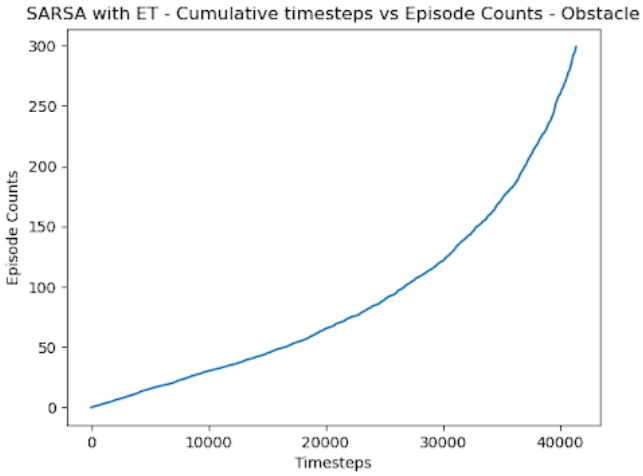


Figure 18: Learning curve

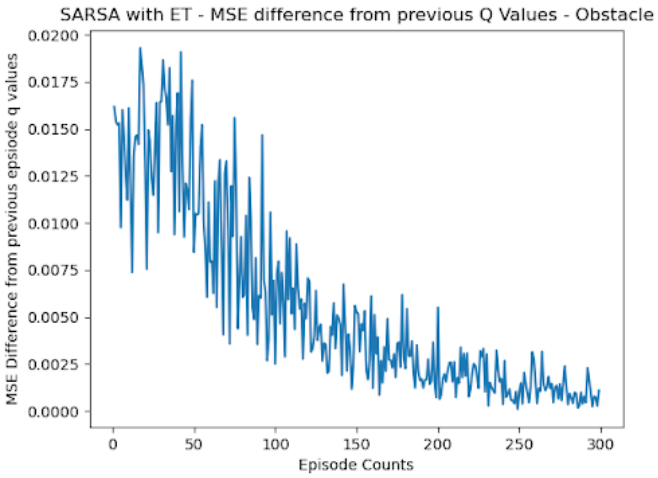


Figure 19: MSE curve

It can be seen that the value function is converging. Unfortunately, the curve is not smooth as we did not have enough iterations due to computational constraints

10 Q Learning with eligibility traces

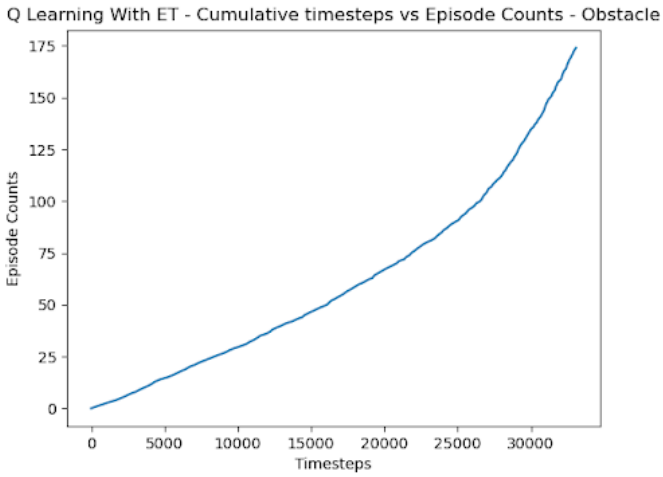


Figure 20: Learning curve

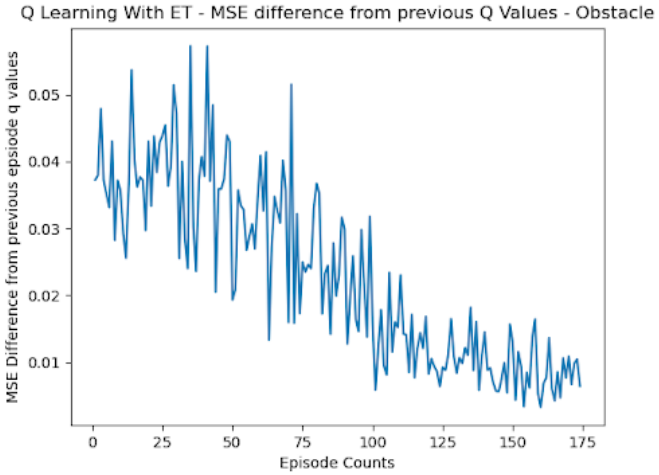


Figure 21: MSE curve

It can be seen that the value function is converging. Unfortunately, the curve is not smooth as we did not have enough iterations due to computational constraints