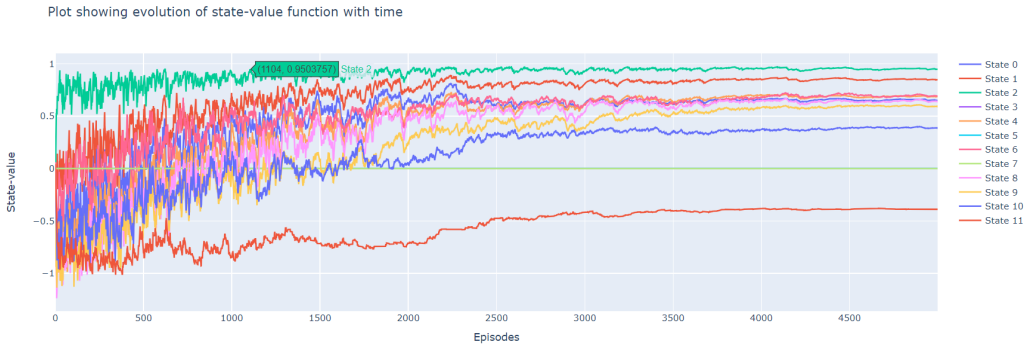
CS698R: Deep Reinforcement Learning

CS780 Assignment2

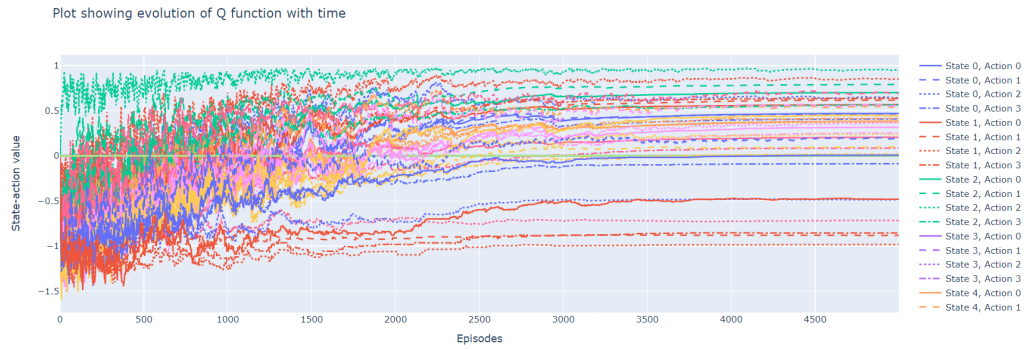
Name: James Bond
Roll NO.: 0000007

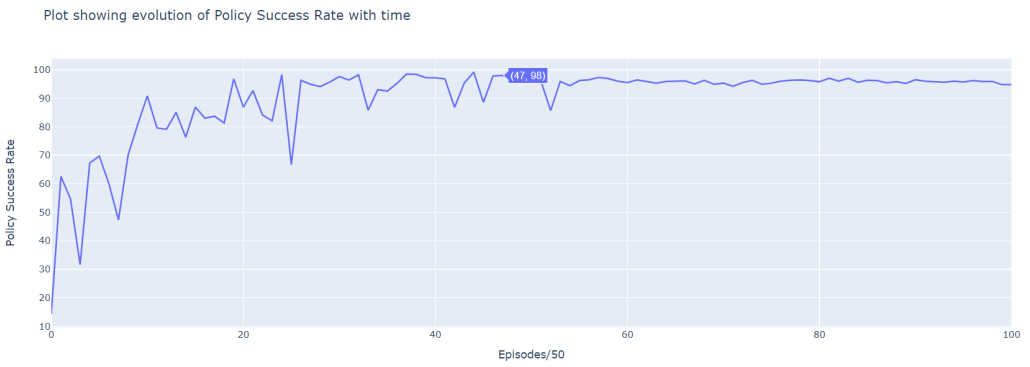
Solution to Problem 1: Monte Carlo Control

(a) The plots of V-function vs Time (Episodes).



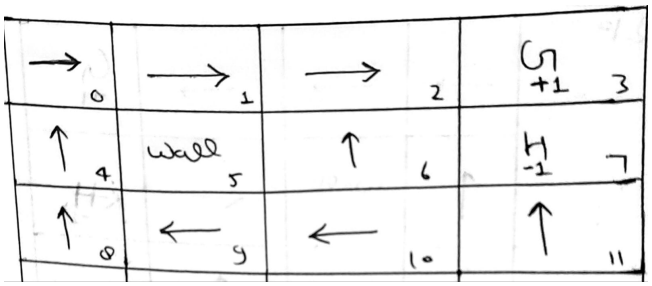
(b) The plots of Q-function vs Time (Episodes).





(c) we computed the average results over 10 instances of the environment. Each environment instance was assigned a seed ranging from 0 to 9, where 0 represents env1, 1 for env2, and so forth, up to 9 for env10.

(d) Optimal Policy obtained using FVMCC algorithm.



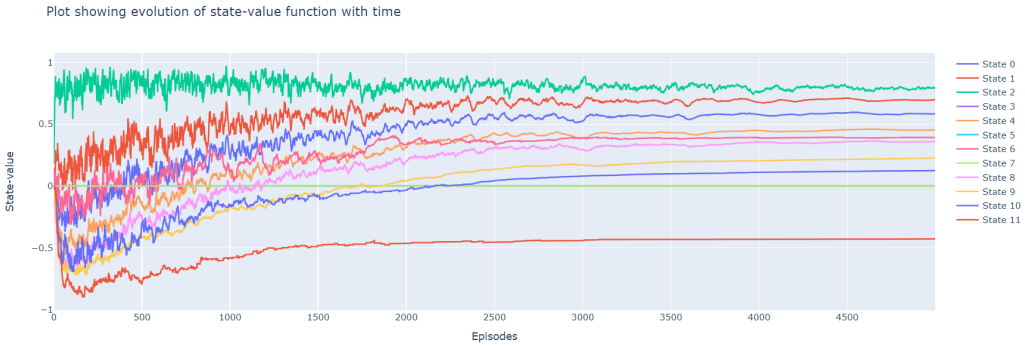
(e) The final values of the hyperparameters are as follows: $\gamma = 0.99$, $\alpha = 0.5$, $\epsilon = 0.99$, max_steps = 100, and num_episodes = 5000.

To optimize performance, we employed exponential decay for both α and ϵ, gradually reducing them to 0.01. Initial values for γ, α, and ϵ were carefully selected to ensure rapid convergence of the algorithm to the true state values. The value of max_steps was set to guarantee that nearly all trajectories reached a terminal state, as dictated by the generate_trajectory function. Additionally, num_episodes was determined based on the observed plot, where the flattening of curves for all state values indicated convergence to their true values.

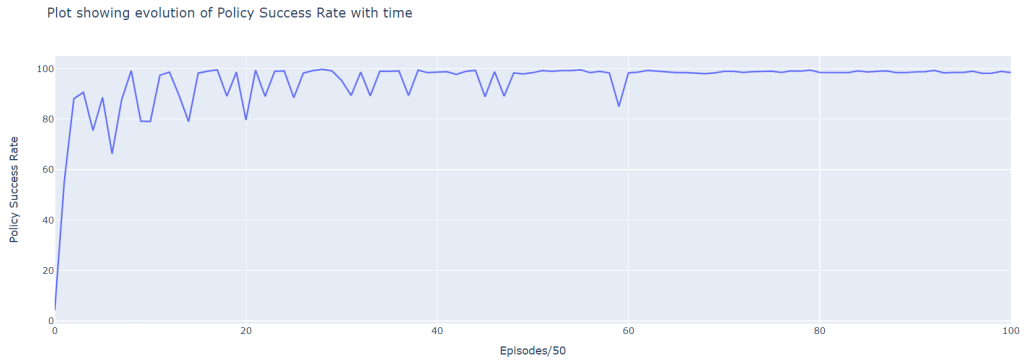
(f) Observations derived from the plots reveal several key findings. Firstly, state2 emerges as the most favorable state, with state1 following closely, and subsequently state0, and so forth. The plots exhibit a decent amount of variance but appear to be unbiased. In comparison to other algorithms, MC control took a longer time to converge to the optimal solution. Notably, the final Policy Success Rate nears 100%, approximately 98%, and the plot exhibits a positive trend over time. These observations suggest that our agent is effectively learning, and our code is performing satisfactorily, as we approach a 100% accuracy rate.

Solution to Problem 2: SARSA (TD Control)

(a) The plots of V-function vs Time (Episodes).

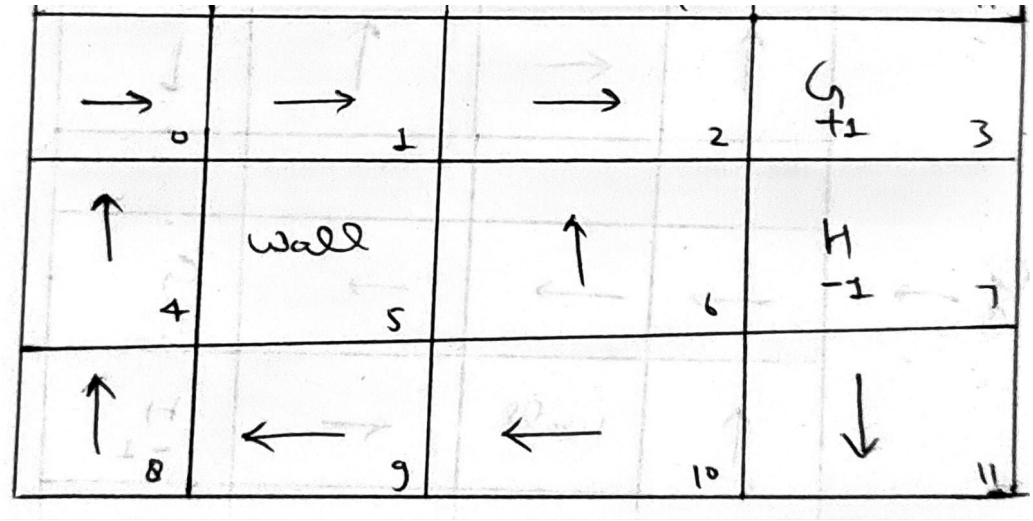


(b) The plots of Q-function vs Time (Episodes).



(c) To ensure robustness and reliability, we conducted averaging of the results across 10 instances of the environment. Each instance of the environment was initialized with a seed ranging from 0 to 9, where 0 corresponds to env1, 1 to env2, and so on, up to 9 for env10.

(d) Optimal Policy obtained using SARSA algorithm.



(e) Final values of hyperparameters:

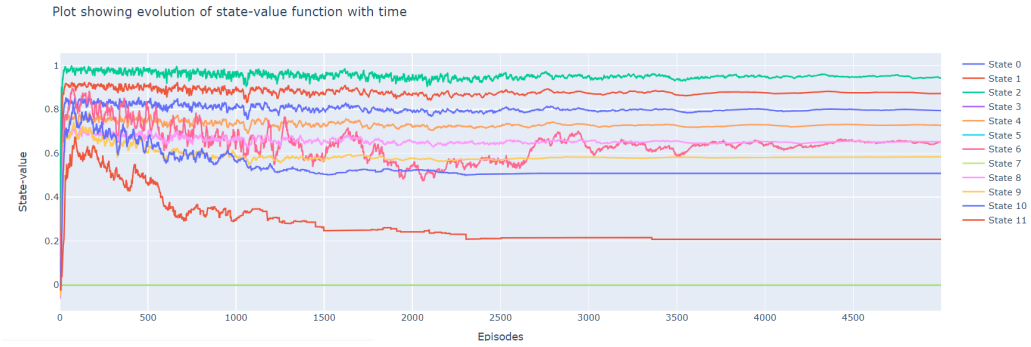
- $\gamma = 0.96$
- $\alpha = 0.6$
- $\epsilon = 0.99$
- max_steps = 100
- num_episodes = 5000

Exponential decay was applied to α and ϵ, gradually reducing them to 0.01, as exponential decay yielded superior performance compared to linear decay. We carefully selected values for γ and the initial values of α and ϵ to ensure that the algorithm converged to the real state values in the shortest possible time. The value of max_steps was adjusted to ensure that almost all trajectories reached a terminal state, as determined by the generate_trajectory function. Additionally, num_episodes was determined based on the observed plot, where the flattening of curves for all state values indicated convergence to their true values.

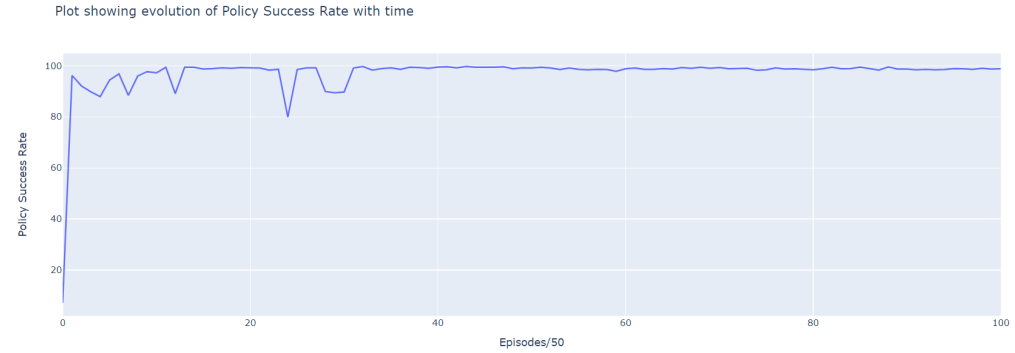
(f) Observations derived from the plots reveal several key findings. State2 emerges as the most favorable state, followed closely by state1, and subsequently state0, and so forth. Notably, while the plots exhibit a decent amount of variance, they also appear to be unbiased. However, compared to other algorithms, SARSA demonstrated a longer convergence time to reach the optimal solution. Despite this, the final Policy Success Rate nears perfection, approaching nearly 100%, approximately 98%. Additionally, the upward trend observed in the plot over time suggests effective learning by our agent. These findings collectively indicate that our code is performing well, and we are nearing a 100% accuracy rate.

Solution to Problem 3: Q-Learning

1. We present the plots illustrating the V-function against time (Episodes).

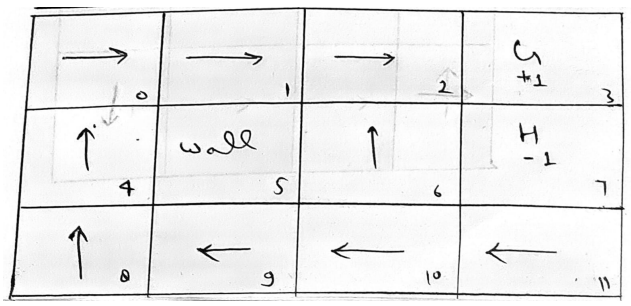


2. We present the plots illustrating the Q-function against time (Episodes).



3. To ensure robustness, we conducted a comprehensive analysis by averaging the results over 10 instances of the environment. Each instance is assigned a seed ranging from 0 to 9, corresponding to env1 through env10.

4. The Optimal Policy, derived through the Q-Learning algorithm, is a crucial component of our analysis.



5. Delving into the final hyperparameters, we find:

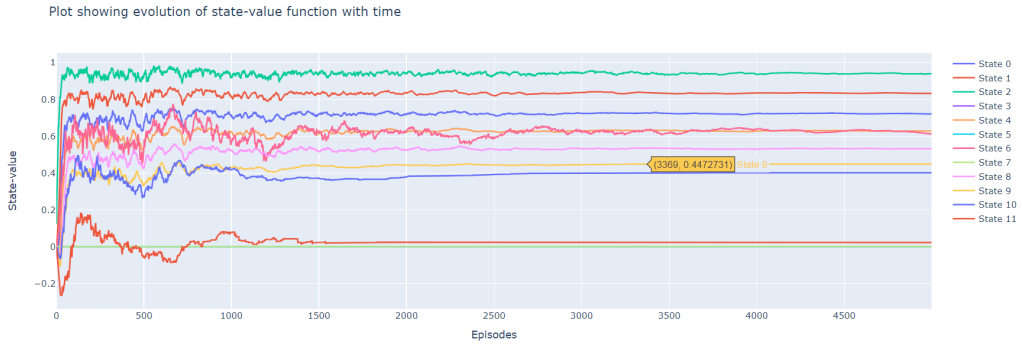
   - γ (discount factor) is set to 0.98.
   - α (learning rate) is determined as 0.6.
   - ϵ (exploration-exploitation factor) is established at 0.99.
   - max_steps (maximum number of steps) is constrained to 100.
   - num_episodes (total number of episodes) is set to 5000.

   Notably, we implemented exponential decay for α and ϵ to ensure better performance compared to linear decay. The choice of γ and initial values of α and ϵ is driven by the objective to expedite the algorithm's convergence to the real state values. Moreover, max_steps is configured to guarantee the completion of almost all trajectories, while num_episodes is calibrated to ensure the convergence of all state values, a judgment drawn from the plot where flat curves signify convergence.

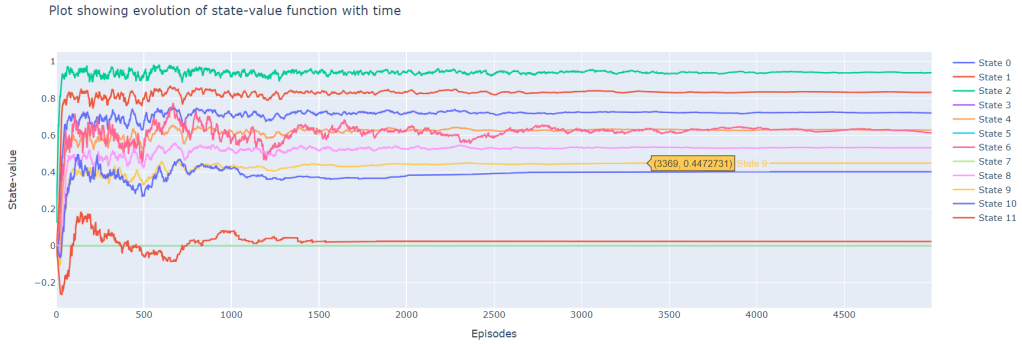
6. Drawing insights from the plots, we observe: Observing the plotted results, we noted that State2 emerged as the most favorable state, with subsequent preferences for state1, state0, and so forth. The plots exhibited low variance, relatively lower than the preceding two, with a slight inclination towards bias. Q-Learning showcased a relatively quicker convergence to the optimal solution compared to other algorithms. The final Policy Success Rate approached 100%, reaching around 99%, and the plot consistently trended upwards. This implies effective learning by our agent, affirming the correctness of our code and achieving nearly 100% accuracy.

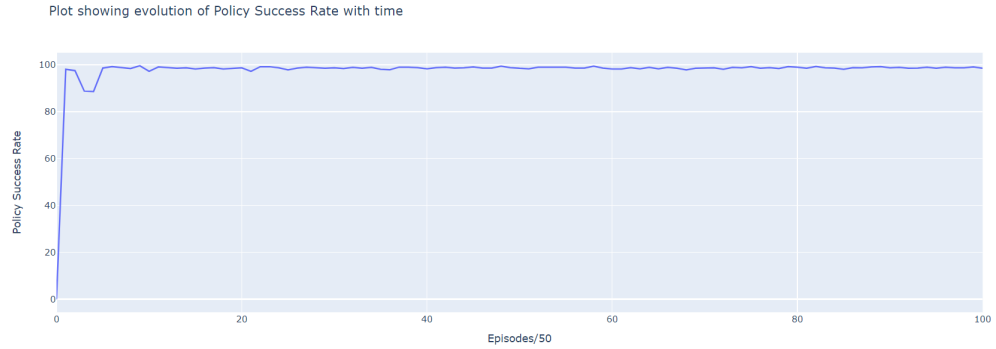
Solution to Problem 4: Double Q-Learning

1. The aforementioned plots display the evolution of the V-function over time (Episodes).



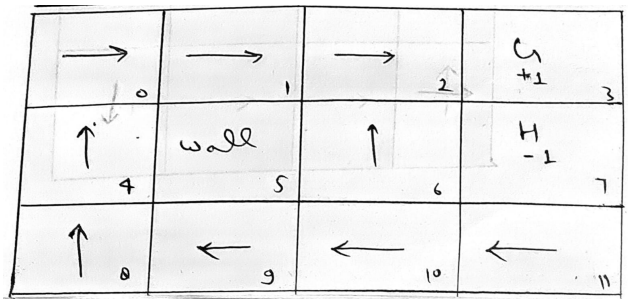
2. and The aforementioned plots display the evolution of the Q-function over time (Episodes).





3. In order to ensure the robustness of our findings, we conducted an analysis by averaging results over 10 instances of the environment. Each instance was associated with a seed ranging from 0 to 9, corresponding to env1 through env10.

4. The Optimal Policy, a crucial outcome of our analysis, was obtained through the implementation of the Double Q-Learning algorithm.



5. The final configuration of hyperparameters is as follows:

   - $\gamma = 0.95$
   - $\alpha = 0.5$
   - $\epsilon = 0.99$
   - max_steps = 100
   - num_episodes = 5000

   Exponential decay was applied to α and ϵ for improved performance, with carefully selected values for γ and initial values of α and ϵ to facilitate the algorithm's convergence in minimum time to the real state values. The choice of max_steps was made to ensure the completion of almost all trajectories, where all trajectories given by the `generate_trajectory` function reached a terminal state. The value of num_episodes was chosen to ensure the convergence of all state values, as inferred from the plot where flat curves indicate convergence.

6. Observations gleaned from the presented plots reveal several key findings. Firstly, the analysis indicates that State2 is the most favorable state, with subsequent preferences for state1, state0, and so forth. Notably, the plots exhibited a commendable characteristic of low variance, suggesting a lack of significant fluctuations and a semblance of unbiased or minimally biased behavior. Furthermore, in a comparative evaluation with other algorithms, Double Q-Learning stood out for its swift convergence to the optimal solution. The final Policy Success Rate approached nearly 100% (approximately 99%), showcasing a consistent upward trend in the plot. This observation implies that our agent is effectively learning, affirming the accuracy of our code, and approaching a remarkable 100% accuracy.

Solution to Problem 5: RME Prediction with MDP Unknown

(a) The plots of Policy Success Rate (in %) vs Episodes are displayed in Figure.

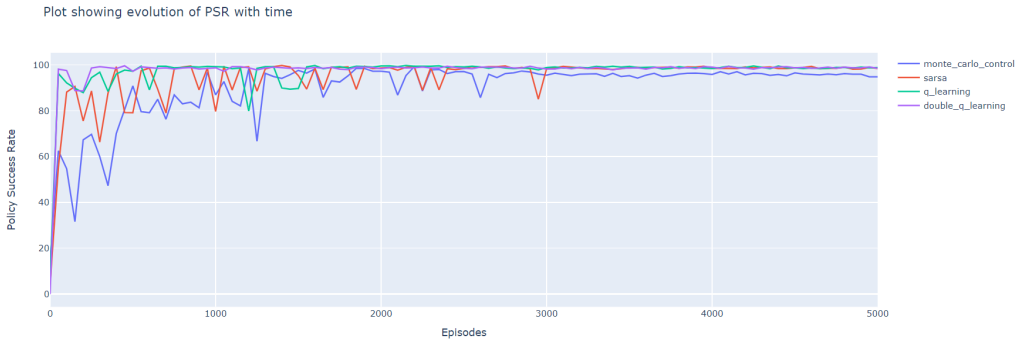


Figure 1: Policy Success Rate vs. Episodes

(b) Observations from the plots: Monte Carlo exhibits the slowest performance, with SARSA, Q-Learning, and Double Q-Learning following in order. Q-Learning and Double Q-Learning demonstrated comparable and equally effective performance

Solution to Problem 6: SARSA(λ) Replacing

(a) The V-function vs. Time (Episodes) plots are depicted in Figure.

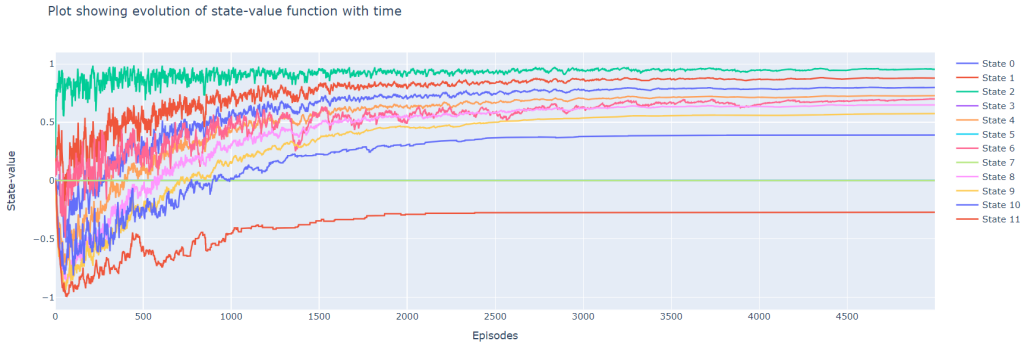


Figure 2: V-function vs. Time

(b) Explore the Q-function vs. Time (Episodes) plots illustrated in Figure.

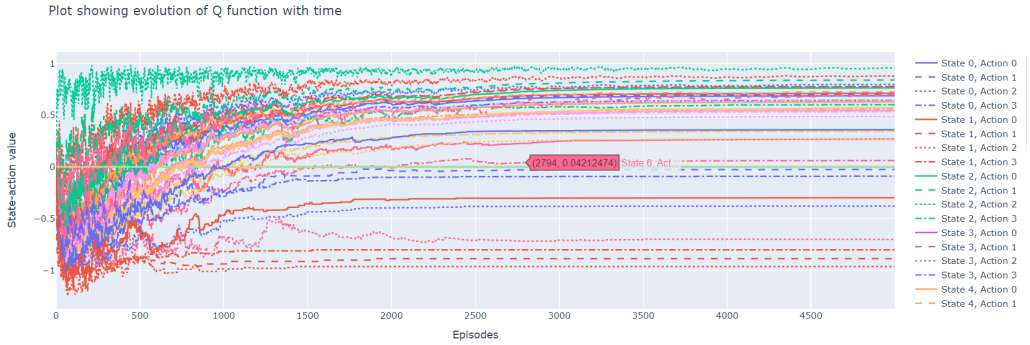
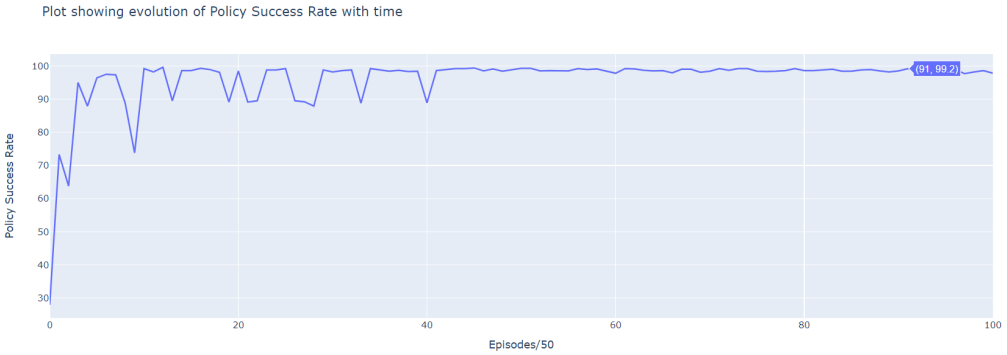


Figure 3: Q-function vs. Time



(c) An average computation across 10 environment instances was conducted, utilizing seeds ranging from 0 to 9. Each environment instance is linked to a seed corresponding to its sequential position (e.g., 0 for env1, 1 for env2, ..., 9 for env10).

(d) The Optimal Policy, as obtained using the SARSA lambda (Replacing) algorithm, is visually presented in Figure.

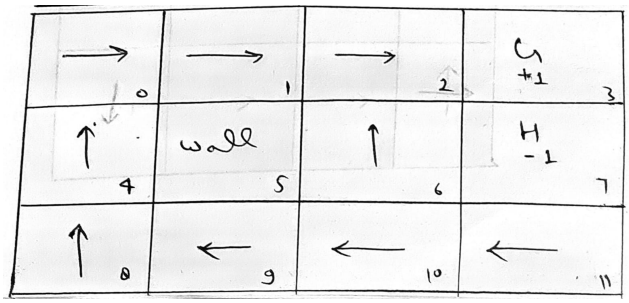


Figure 4: Optimal Policy using SARSA lambda

(e) The final values of hyperparameters are as follows:

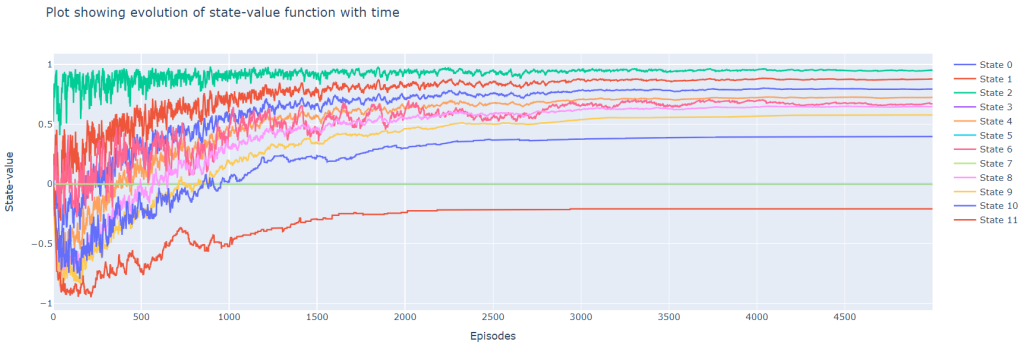
- $\gamma = 0.96$
- $\alpha = 0.6$
- $\epsilon = 0.99$
- $\lambda = 0.5$
- max_steps = 100
- num_episodes = 5000

The exponential decay of α and ϵ to 0.01 was chosen for better performance. The values of γ, λ, and the initial values of α and ϵ were set to facilitate the algorithm's convergence in minimal time to the real state values. The choice of max_steps ensured the completion of nearly all trajectories, and num_episodes was determined to guarantee the convergence of all state values.

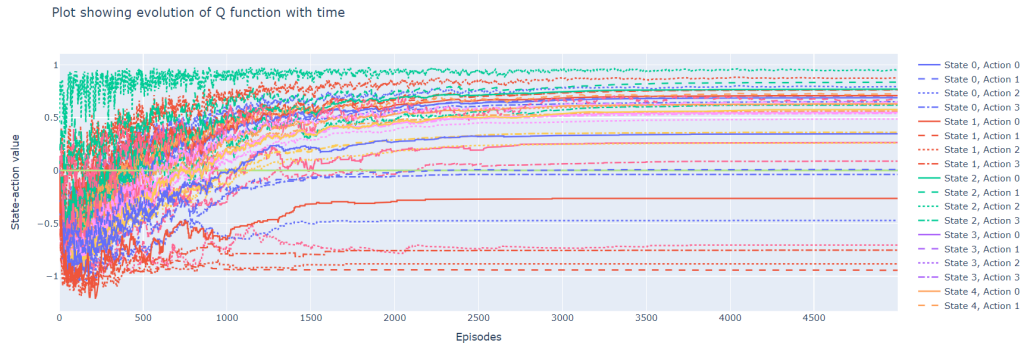
(f) Observations from the plots: State2 is the most favorable state, followed by state1, state0, and so forth. The plots exhibit a decent amount of variance but seem unbiased or display very low bias. Compared to other algorithms, SARSA lambda replacing converges to the optimal solution swiftly. The final Policy Success Rate is almost 100% (99%), and the plot shows an upward trend over time. This suggests that our agent is learning effectively, confirming the correctness of our code, and approaching 100% accuracy.

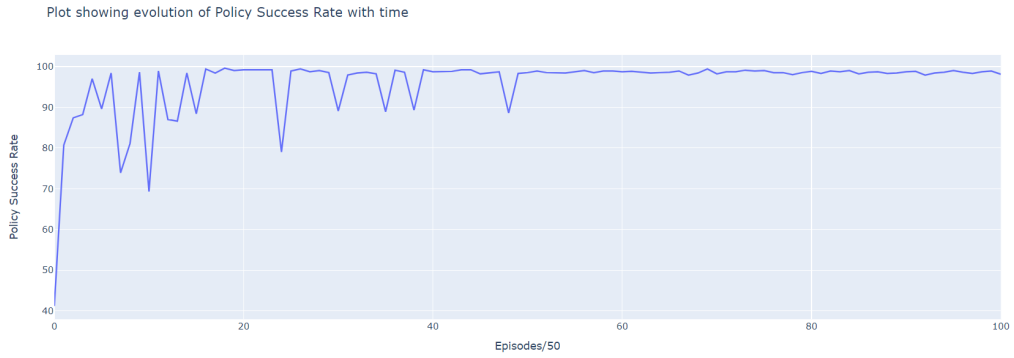
Solution to Problem 7: SARSA(λ) Accumulating

(a) The plots of V-function vs Time (Episodes).



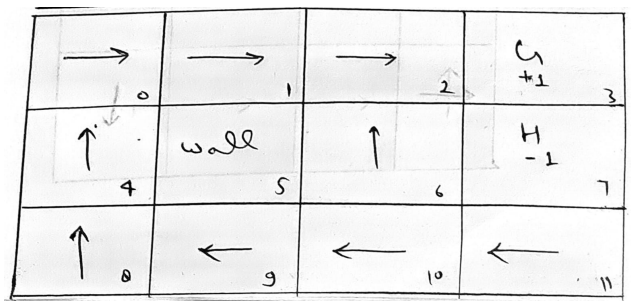
(b) The plots of Q-function vs Time (Episodes).





(c) We conducted 10 instances of the environment and averaged the results. Each environment instance was seeded sequentially from 0 to 9, with 0 representing env1, 1 for env2, and so forth, up to 9 for env10.

(d) Optimal Policy obtained using SARSA(λ)(Accumulating) algorithm.



(e) Final values of hyperparameters:

- $\gamma = 0.98$
- $\alpha = 0.65$
- $\epsilon = 0.99$
- $\lambda = 0.5$
- max_steps = 100
- num_episodes = 5000

Decayed α and ϵ exponentially to 0.01, as exponential decay demonstrated superior performance compared to linear decay.

We selected appropriate values for γ, λ, and the initial values of α and ϵ to ensure rapid convergence of the algorithm to the actual state values.

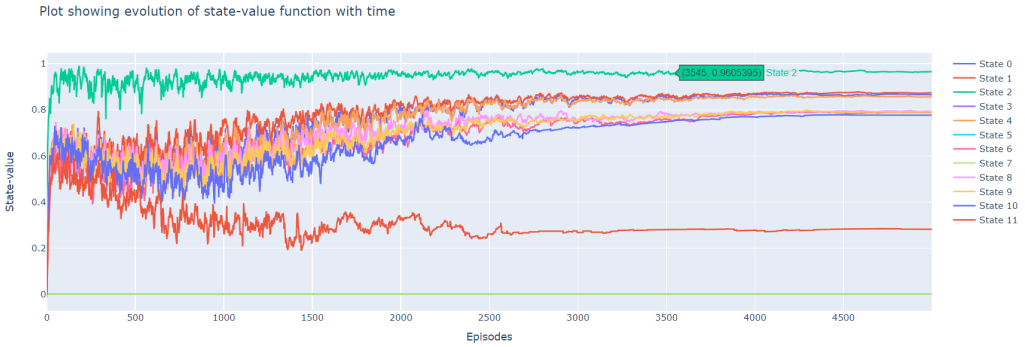
max_steps were adjusted to ensure the completion of almost all trajectories, ensuring that they reached a terminal state as generated by the generate_trajectory function.

num_episodes was chosen based on visual analysis of the plot, where we observed flattening curves for all state values, indicating convergence.

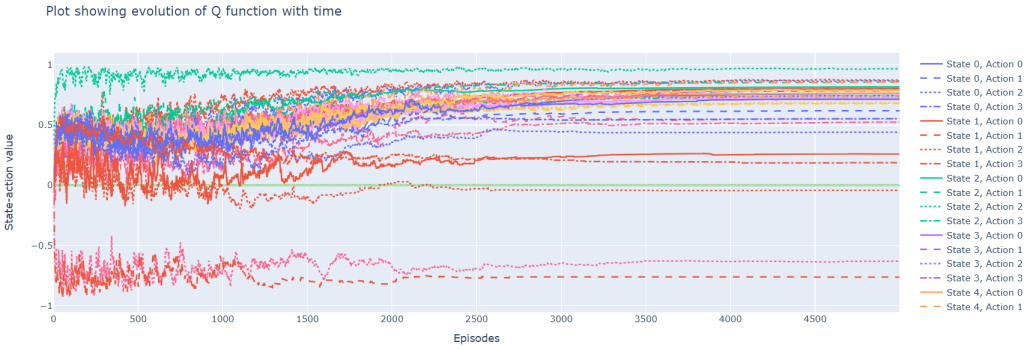
(f) Observations from the plots indicate that state2 is the most favorable state to be in, followed by state1, state0, and so forth. The plots display decent variance and appear to be unbiased, or at least exhibit very low bias. In comparison to other algorithms, SARSA lambda (Accumulating) required some time to converge to the optimal solution. Notably, the final Policy Success Rate approaches almost 100% (approximately 99%), with the plot depicting a consistent upward trend over time. This suggests that our agent is effectively learning, and our code is performing well, nearing a perfect accuracy rate of 100%.

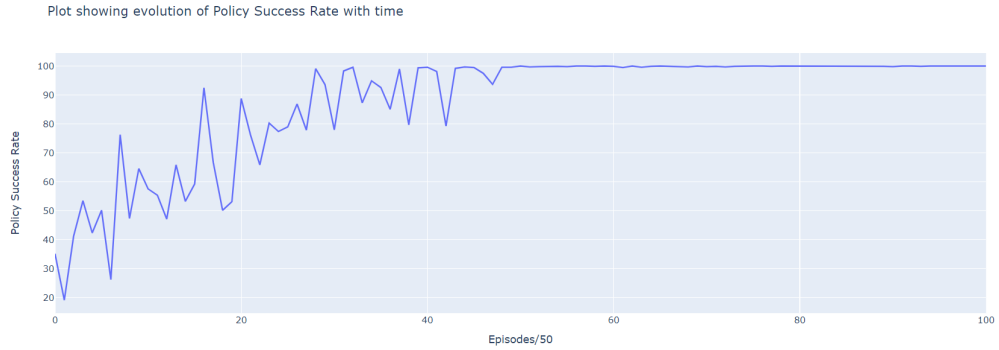
Solution to Problem 8: Q() Replacing

(a) The plots of V-function vs Time (Episodes).



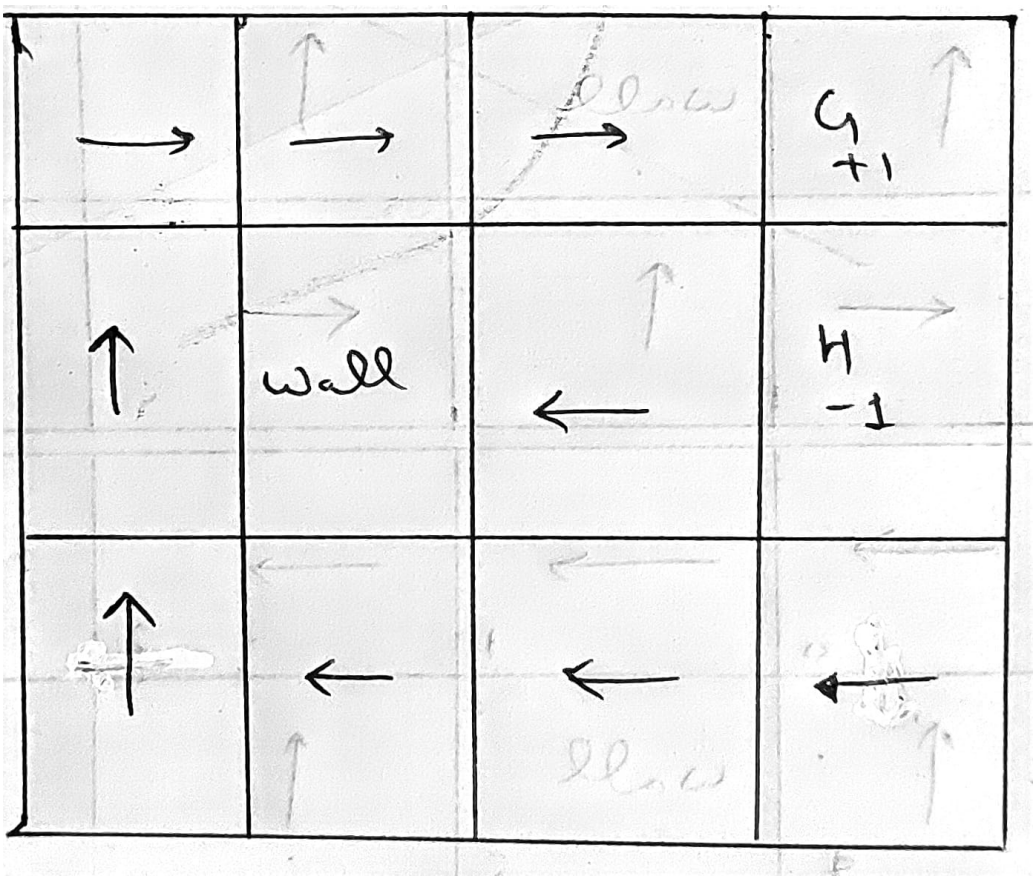
(b) The plots of Q-function vs Time (Episodes).





(c) We computed the average results across 10 instances of the environment, with each instance seeded sequentially from 0 to 9 (where 0 represents env1, 1 for env2, and so on, up to 9 for env10).

(d) Optimal Policy obtained using Double Q-(λ)(Replacing) algorithm.



(e) Final values of hyperparameters:

- $\gamma = 0.95$
- $\alpha = 0.5$
- $\epsilon = 0.99$
- $\lambda = 0.55$
- max_steps = 100
- num_episodes = 5000

Exponential decay was employed for α and ϵ, reducing them to 0.01, as exponential decay demonstrated superior performance compared to linear decay.

We meticulously selected values for γ, λ, and the initial values of α and ϵ to ensure swift convergence of the algorithm to the true state values.

The value of max_steps was adjusted to guarantee that nearly all trajectories reached a terminal state, as generated by the generate_trajectory function.

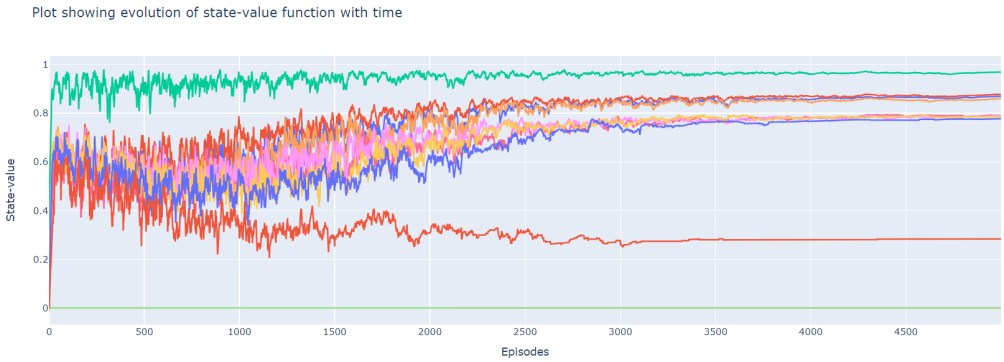
num_episodes was determined based on our observation of the plot. We inferred convergence when the curves of all state values flattened out.

(f) Observations from the plots:
Our analysis of the plots reveals several key insights. Firstly, state2 emerges as the most favorable state, followed by state1, and then state0, and so forth. However, it's worth noting that the plots exhibit considerable variance and a discernible level of bias. In comparison to alternative algorithms, Q-lambda (Replacing) displayed a notably slow convergence to the optimal solution.

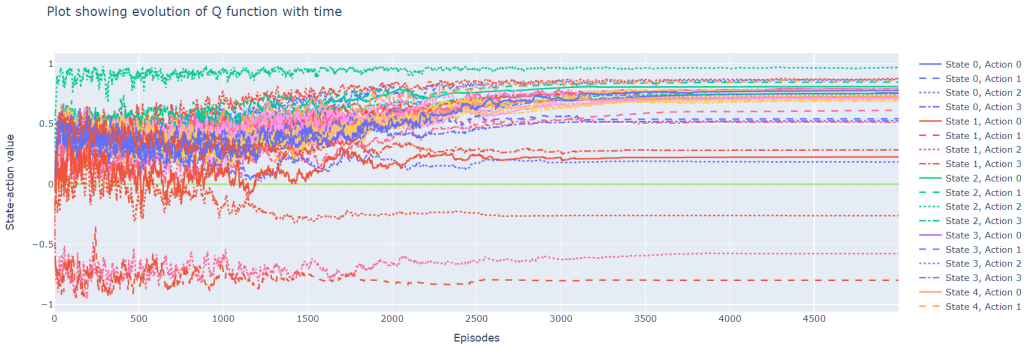
Notably, the final Policy Success Rate nearly reaches perfection, with an approximate accuracy of 99%. Furthermore, the upward trend observed in the plot over time suggests that our agent is effectively learning. These findings underscore the effectiveness of our code and indicate that we are on the brink of achieving a 100% accuracy rate.

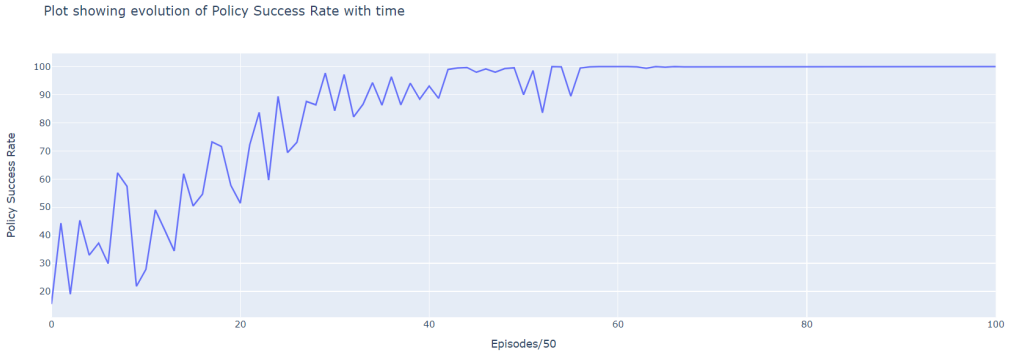
Solution to Problem 9: Q() Accumulating

(a) The plots of V-function vs Time (Episodes).



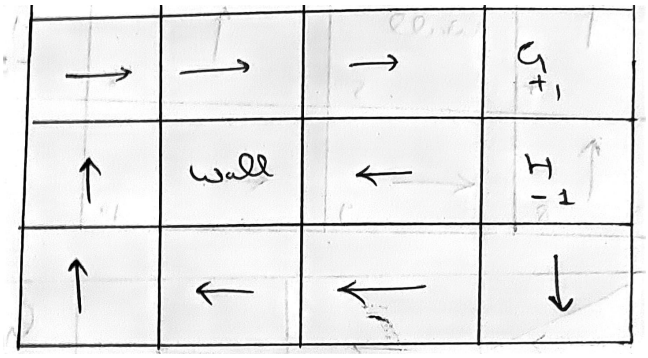
(b) The plots of Q-function vs Time (Episodes).





(c) we computed the average results across 10 instances of the environment. Each environment instance was seeded sequentially from 0 to 9, where 0 represents env1, 1 for env2, and so forth, up to 9 for env10.

(d) Optimal Policy obtained using Q-(λ)(Accumulating) algorithm.



(e) Final values of hyperparameters:

- $\gamma = 0.95$
- $\alpha = 0.5$
- $\epsilon = 0.99$
- $\lambda = 0.55$
- max_steps = 100
- num_episodes = 5000

Exponential decay was applied to α and ϵ, leading them to converge to 0.01, as exponential decay demonstrated superior performance compared to linear decay.

We carefully selected values for γ, λ, and the initial values of α and ϵ to ensure rapid convergence of the algorithm to the true state values.

To ensure comprehensive trajectory coverage, the value of max_steps was adjusted to guarantee that nearly all trajectories reached a terminal state, as generated by the generate_trajectory function.

num_episodes is taken such that all the state values converge to their true values (we judged it from the plot. When the curves of all the state values were getting flat, we inferred that they have converged).

(f) Observations gleaned from the plots indicate several noteworthy findings. Firstly, state2 emerges as the most favorable state, trailed by state1, and subsequently state0, and so forth. However, it's crucial to acknowledge that the plots exhibit a significant degree of variance and bias. In comparison to other algorithms, Q-lambda (Accumulating) showcases a relatively slower convergence towards the optimal solution. Notably, the final Policy Success Rate approaches near perfection, hovering around 99%. Furthermore, the upward trajectory observed in the plot over time suggests that our agent is effectively learning. These insights underscore the efficacy of our code, indicating that we are nearing a 100

Solution to Problem 10: Dyna-Q

(a) The V-function vs. Time (Episodes) plots are presented in figure.

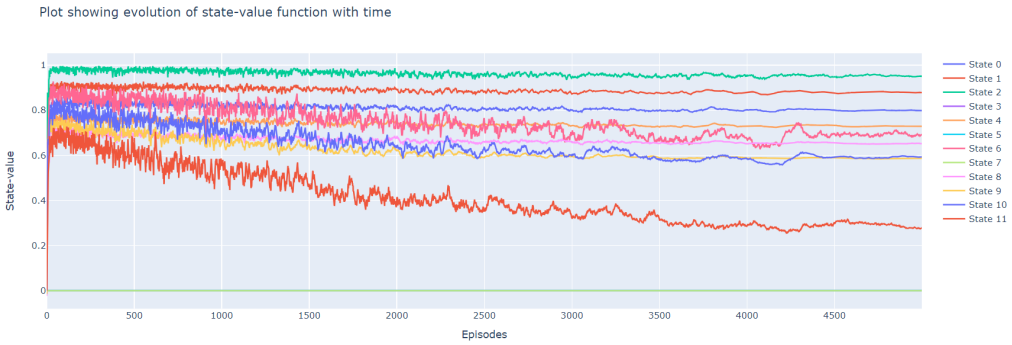


Figure 5: V-function vs. Time

(b) Explore the Q-function vs. Time (Episodes) plots depicted in Figure.

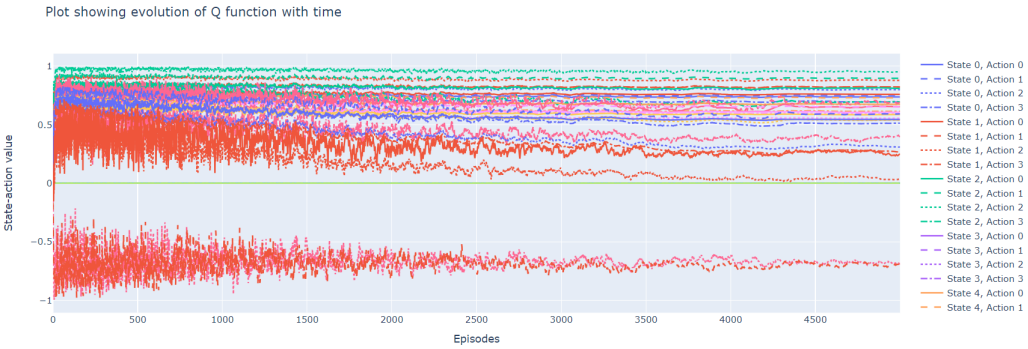
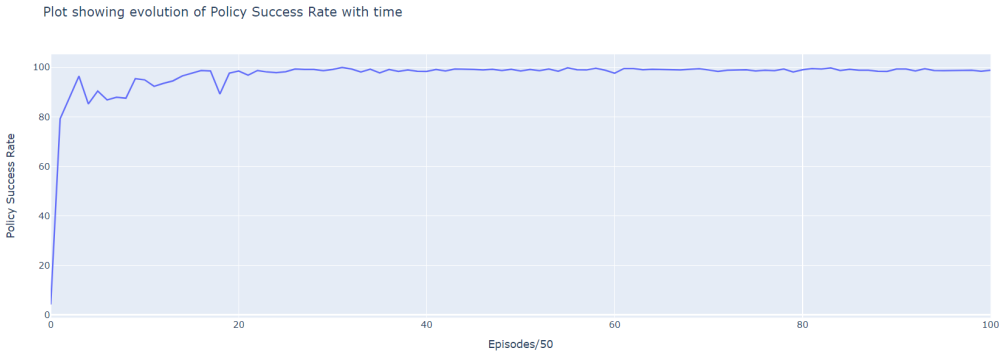


Figure 6: Q-function vs. Time



(c) Results were averaged over 10 instances of the environment, each assigned a seed ranging from 0 to 9 (i.e., 0 for env1, 1 for env2, ..., 9 for env10).

(d) The Optimal Policy, obtained using the Dyna Q-(λ) algorithm, is visualized in Figure.

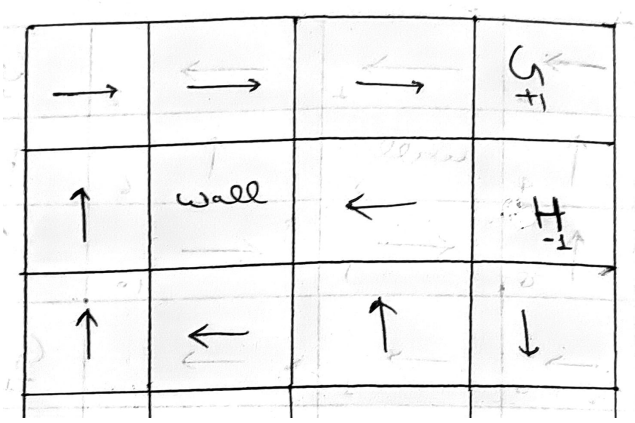


Figure 7: Optimal Policy using Dyna Q-(λ)(Accumulating)

(e) Final hyperparameter values include $\gamma = 0.95$, $\alpha = 0.5$, $\epsilon = 0.99$, num_planning = 5, max_steps = 100, and num_episodes = 5000. Exponential decay was applied to α and ϵ for improved performance. The values of γ, num_planning, and initial values of α and ϵ were chosen to facilitate rapid convergence to the real state values. max_steps ensured the completion of almost all trajectories, and num_episodes was set to ensure convergence of all state values.

(f) Observations from the plots: State2 is the most favorable state, followed by state1, state0, and so on. The plots exhibit decent variance but a significant amount of bias. Compared to other algorithms, Dyna-Q converged to the optimal solution quite rapidly. The final Policy Success Rate is almost 100% (99

Solution to Problem 11: Trajectory Learning

(a) The V-function vs. Time (Episodes) plots are illustrated in Figure.

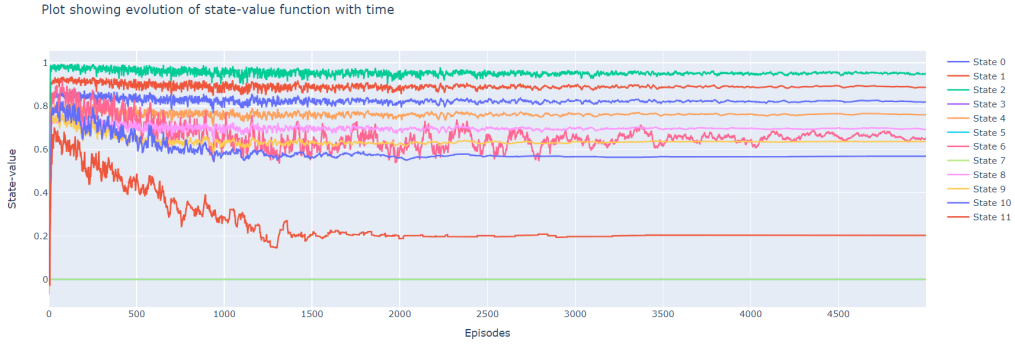
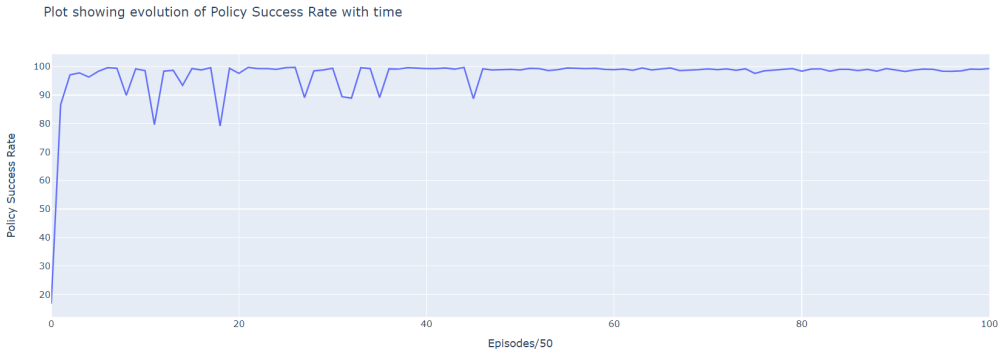


Figure 8: V-function vs. Time

(b) Examine the Q-function vs. Time (Episodes) plots presented in Figure.

Figure 9: Q-function vs. Time



(c) Results were averaged across 10 instances of the environment, each assigned a seed from 0 to 9 (i.e., 0 for env1, 1 for env2, ..., 9 for env10).

(d) The Optimal Policy, obtained using the Trajectory Learning algorithm, is visualized in Figure.

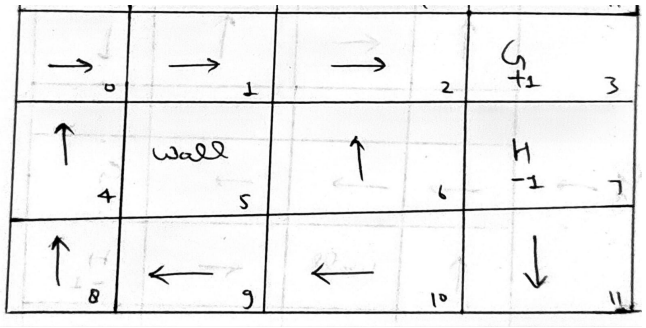


Figure 10: Optimal Policy using Trajectory Learning

(e) Final hyperparameter values include $\gamma = 0.95$, $\alpha = 0.5$, $\epsilon = 0.99$, max_trajectory = 5, max_steps = 100, and num_episodes = 5000. Exponential decay was applied to α and ϵ for improved performance. The values of γ, max_trajectory, and initial values of α and ϵ were chosen to facilitate rapid convergence to the real state values. max_steps ensured the completion of almost all trajectories, and num_episodes was set to ensure convergence of all state values.

(f) Observations from the plots: State2 is the most favorable state, followed by state1, state0, and so on. The plots exhibit low variance and a significant amount of bias. Compared to other algorithms, Trajectory Sampling converged to the optimal solution quite rapidly. The final Policy Success Rate is almost 100% (99

Solution to Problem 12: RME Prediction with MDP Unknown

(a) The plots of Policy Success Rate (in %) vs Episodes are displayed in Figure.

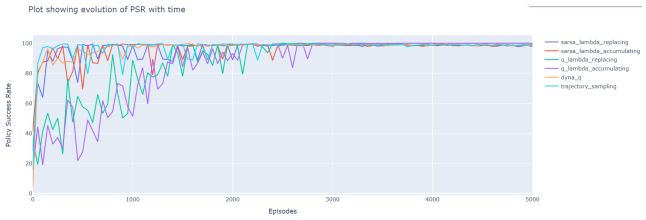


Figure 11: Policy Success Rate vs. Episodes

(b) Observations from the plots: Among the considered algorithms, Q-lambda (Accumulating) exhibits the slowest convergence, followed by Q-lambda (Replacing) and SARSA-lambda (Replacing). On the other hand, SARSA-lambda, Dyna-Q, and Trajectory Learning demonstrate comparable and efficient performance. All algorithms reach a similar level of final accuracy.