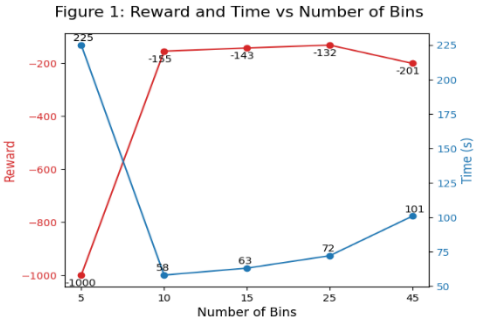
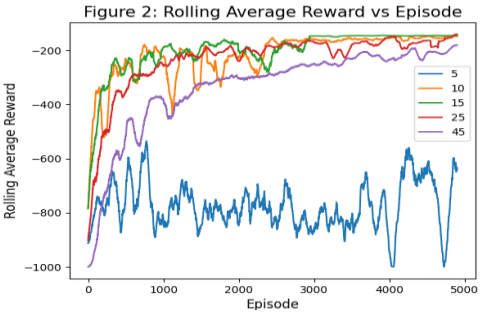
INF368 – P4 – Ex. 4 – Feature Engineering

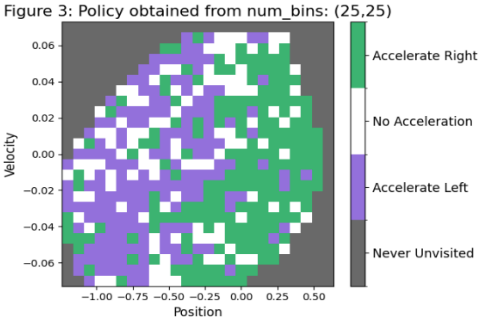
In our exploration of Gymnasium's Mountain Car environment, we focused on feature engineering to transform continuous raw inputs into discrete states. Opting for simplicity, we utilized a fixed number of bins for both the position and velocity of the agent, simplifying interpretation and handling of the engineered data. For control, we employed Q-learning with specific parameters: learning rate α=0.1, discount factor γ=0.95, epsilon ϵ=0.5, epsilon decay constant ϵ_k=0.995, and ran the algorithm for 5000 episodes. We chose a slightly higher epsilon initially to encourage exploration, gradually decaying it to favour exploitation later in the training process.





Task 2

We initially examined various bin counts, ranging from 3 to 55, before narrowing down our selection to 5, 10, 15, 25, and 45 bins. Lower bin counts offer faster training due to reduced resolution, while higher counts provide finer state and action space resolution at the expense of longer training times. Our results, depicted in Figure 1, generally align with this expectation, except for the case of 5 bins. Figure 2 demonstrates that 5 bins result in consistently lower rolling average rewards, leading to more episodes reaching the maximum iteration limit compared to other bin counts and will thus use more time. Notably, 45 bins display slower but steadier reward growth with less noise, suggesting potential for improved policy with additional episodes. However, with 5000 episodes, 25 bins appear to yield the best outcome.



Looking at Figure 3 we see the best policy. We see the some of the states are never explored, at the edges, but most of the state space is assigned an action. The ones on the left side seem to be mostly purple to signify accelerating left and those on the right side seems to mostly be green to accelerate right and with a few states the other colour is marked. These seems to be states where it has learned that just accelerating in the same direction won't make it go any further up the mountain, rather it must stop or go back down to build up momentum.

Task 3

We considered employing tile encoding, a form of coarse coding, to create binary features. Tile encoding partitions the state space into overlapping tiles or grids, each associated with a feature vector. While offering flexibility in capturing state space structure, tile encoding expands the state space significantly.

Another option explored was direct function approximation using neural networks to approximate Q or π, bypassing discretization before Q-learning. This approach enables better generalization across similar states and immediate handling of continuous state space but may be computationally expensive and prone to overfitting without proper regularization.

Additionally, we explored dynamic features incorporating information from previous steps or episodes. Utilizing velocity and position data from previous states could approximate momentum or acceleration, aiding the agent in recognizing the need for momentum to ascend. Incorporating the distance to the goal after initial reach could guide the agent's direction, though it risks the agent fixating on that direction without building momentum. To mitigate issues like maxing out step counts without reaching the goal, we considered reward shaping, adding small rewards for exploration and penalizing repetitive behaviour that didn't yield any results.

While these approaches offer potential benefits in accelerating learning, they also introduce bias and complexity, impacting the true dynamics of the environment and expanding the state space. Careful consideration would be necessary if we implemented any of these forms of feature engineering.

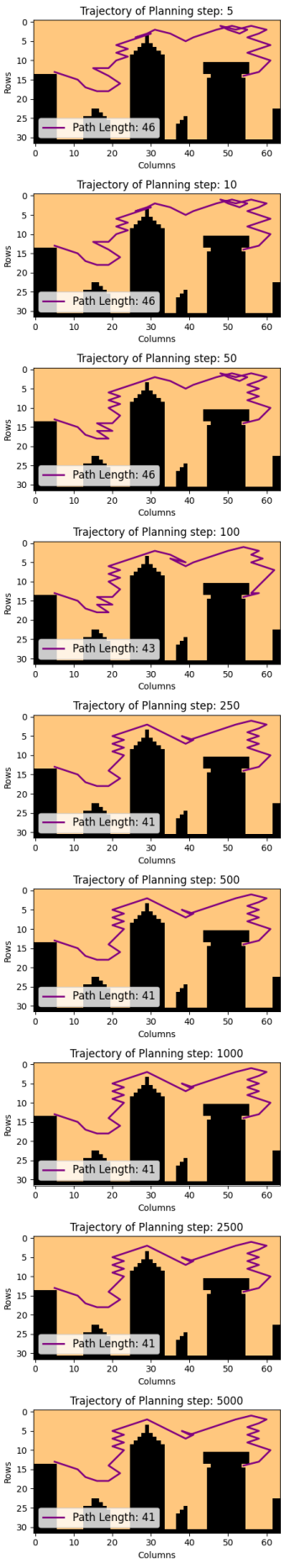
INF368 – P4 – Ex. 5 – Planning

In our project on the Gymnasium's 'GridWorld-v0' environment, we have engaged in the development of a model of the environment, applied the Dyna-Q learning architecture, and visualized the optimal trajectory as a result of our algorithm's training process. To develop a model $\widehat{M}$ of the environment, we constructed transition and reward functions based on the empirical data collected from 'powered_flight.txt'. We subtracted 1 from the first three columns and columns four to six to adjust the data indices, adhering to Python's zero-indexing. The transition function maps each state-action pair to a consequent state, while the reward function associates each state-action pair with a numeric reward, forming the basis for the model estimated M.

We implemented the Dyna-Q architecture by integrating a model-based approach into the reinforcement learning paradigm. Using Q-learning, we adjusted the parameters: learning rate $\alpha = 0.4$, discount factor $\gamma = 0.99$, exploration rate $\epsilon = 0.3$ and we used 2500 number of episodes to ensure the agent found the goal. The planning portion allows the algorithm to simulate experiences and update the Q-values extensively. The optimal policy, derived from the Q-table, was obtained by selecting the action with the highest Q-value for each state.

The optimal trajectory learned by the algorithm is visualized in a plot Figure 4 overlaying the 'GridWorld-v0' environment. The trajectory, marked in purple, indicates the path that the agent is to take from its starting position to the goal, given the learned policy. It represents the sequence of states that the agent traverses, optimized for total reward.

Figure 4: Trajectories of the agent with different planning steps



Task 4

The inclusion of different numbers of planning steps within the Dyna-Q algorithm has led to varying trajectories and path lengths for the agent. Initially, at lower planning steps—5, 10, and 50—the agent's paths are noticeably inefficient, indicated by a longer path length of 46. These initial paths likely represent a phase where the agent's knowledge of the environment is still incomplete, and the value function is not yet well estimated.As the planning steps are moderately increased to 100, the path length decreases to 43, showing initial improvements in the policy as the agent begins to utilize the simulated experiences effectively.

However, when the planning steps are further increased to 250 and 500, the path length shortens to 41, suggesting a significant clarification of the policy. It appears that at these levels, the agent is able to strike a balance between exploring the environment and exploiting its growing knowledge base. Suggesting that the agent was better able to apply the principle of temporal difference learning. By simulating experiences and updating its Q-values accordingly, the agent enhanced its policy, demonstrating an improved understanding of the environment and a closer approximation to an optimal policy.

Contrary to expectations, at 1000 planning steps, the path length does not improve but remains at 41. This plateau suggests that the agent has potentially reached a near-optimal policy where additional planning steps do not yield further significant improvements. This is in line where after a certain point, the agent's policy converges, and additional updates to the Q-table have a minimal impact on the agent's performance. Further increases in the number of planning steps yield the same result, reinforcing the idea of diminishing returns in policy enhancement. This consistency at high planning levels indicates that while the agent continues to learn from the simulated experiences, there is no additional benefit in terms of the trajectory's efficiency. This highlights that beyond a certain threshold, the learning curve flattens out, and the agent's policy becomes stable, showing no significant improvement despite the increased computational efforts.

Overall, the observation suggests that there is an optimal range of planning steps where the agent can effectively learn and improve its policy. Beyond this range, additional planning does not necessarily translate into better decision-making. This finding is crucial as it points to the importance of balancing computational effort against the practical gains in policy performance, highlighting the efficiency of the learning process.