

Reward Shaping Analysis in FrozenLake Environment

Kfir Amoyal 205710411 | Shlomi Friedman 318187002

1. Introduction and Environment

This project investigates how different reward shaping strategies affect reinforcement learning performance. We implemented Monte Carlo Control and SARSA across multiple reward shaping approaches to understand their impact on learning speed, agent behavior, and convergence.

1.1 FrozenLake Grid World

We used a 7×7 FrozenLake grid (49 states) with slippery dynamics (`success_rate=0.7`), approximately 20% hole density (randomly generated with fixed seed 41432), and sparse rewards (+1 for reaching goal, 0 otherwise).

The environment presents a challenging sparse reward problem where agents receive feedback only upon goal achievement, requiring extensive exploration before discovering successful behaviors.

FrozenLake Map



2. Algorithms

The algorithm used in the presented experiment is “Every-Visit Monte-Carlo” and SARSA(0) with the following Hyperparameters:

Learning rate (α): 0.1

Discount factor (γ): 0.95

Initial ϵ : { 0.05, 0.1, 0.2 }

Minimum ϵ : 0.01

ϵ decay: 0.0

Episodes: 10,000

Max steps per episode: 200

Environment: Frozen Lake 7x7

Slippery dynamics: True

Shaping decay: 0.95

Shaping strength: 0.1

3. Reward Shaping Strategies

We have defined four reward shaping strategies in the experiment, and the final reward is calculated as the following: $R'(s,a,s') = R(s,a,s') + h^{k-1} \cdot F(s,a,s')$, where F is the shaping function, h is the shaping decay, and k is the current episode number.

3.1 Baseline (No Shaping)

$R'(s,a,s') = R(s,a,s')$. Uses only original environment rewards as control condition.

3.2 Step-Cost Shaping

$R'(s,a,s') = R(s,a,s') - 0.01$. Adds a small penalty for each step taken, encouraging the agent to reach the goal in fewer steps. This creates a much denser reward signal than the baseline-every transition now provides feedback rather than only goal achievement.

3.3 Potential-Based Distance Shaping

Uses Manhattan distance to goal as potential function: $\Phi(s) = -d(s)$, where $d(s)$ is the Manhattan distance from the. The shaping is calculated as the following formula:

$$F(s,a,s') = \beta \cdot (\gamma \cdot \Phi(s') - \Phi(s)) , \text{ Where } \beta \text{ is the shaping strength.}$$

Movements that decrease distance to goal receive positive shaping rewards (since Φ increases toward zero), while movements that increase distance receive negative rewards. This shaping encourages the agent to moves towards the goal.

3.4 Custom: Safe Position Reward

Our first custom strategy encourages the agent to maintain distance from holes, safety(s'):

- +0.1 if distance to nearest hole > 1
- -0.1 if adjacent (Manhattan distance=1)
- -5.0 if on hole.

In stochastic environments where actions have uncertain outcomes, staying away from holes reduces the risk of accidentally falling in despite intending to move away. The slippery dynamics mean an agent one tile away from a hole has significant risk of falling in when it attempts any movement. This shaping penalizes such risky positions.

3.5 Custom: Safe Progress Reward

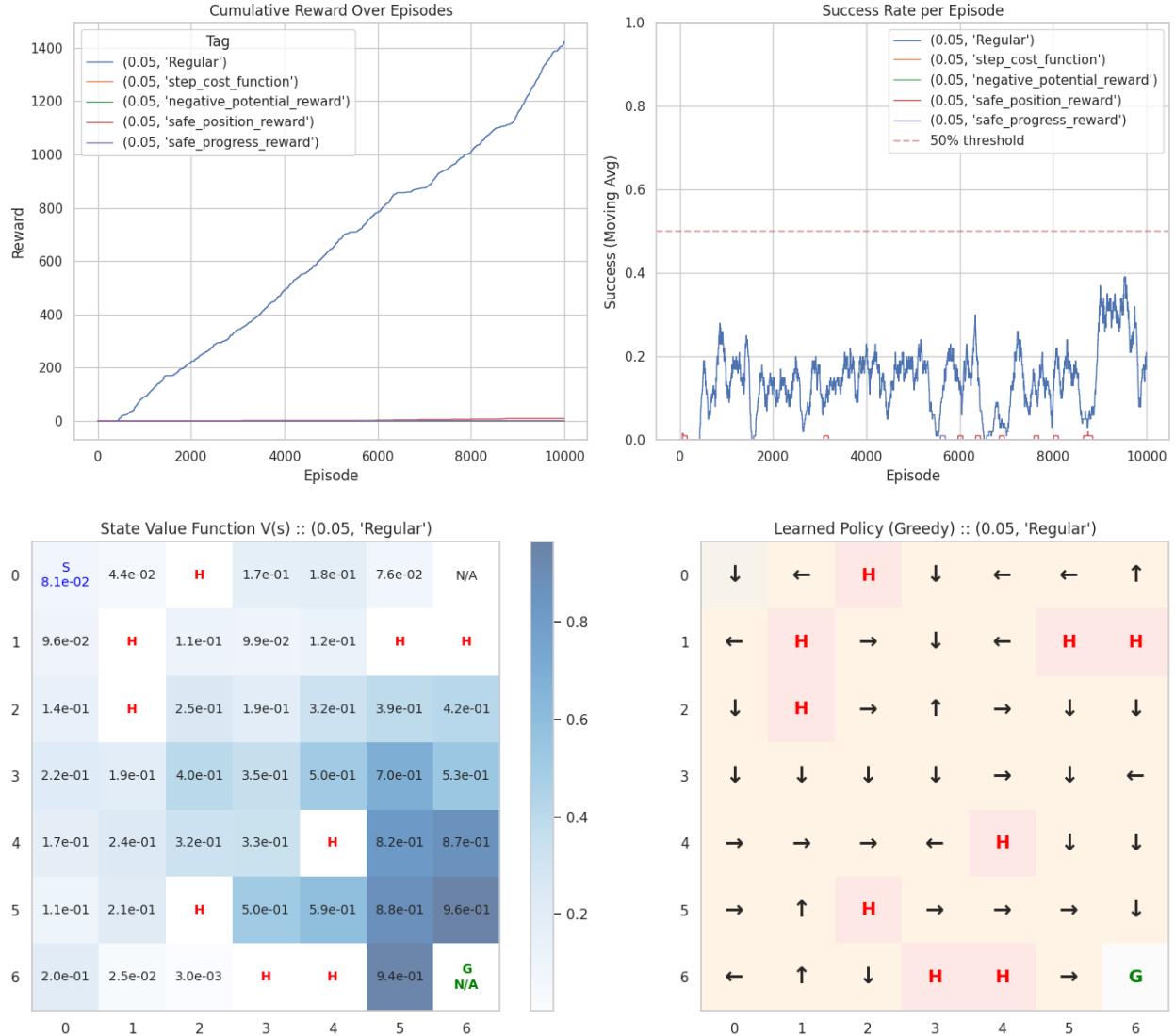
Combines progress toward goal with safety considerations: $R'(s,a,s') = R(s,a,s') + [d(s,goal) - d(s',goal)] + \text{safety}(s')$, where the first term rewards distance reduction and the second applies the same safety bonus/penalty as Safe Position Reward shown in 3.4 .

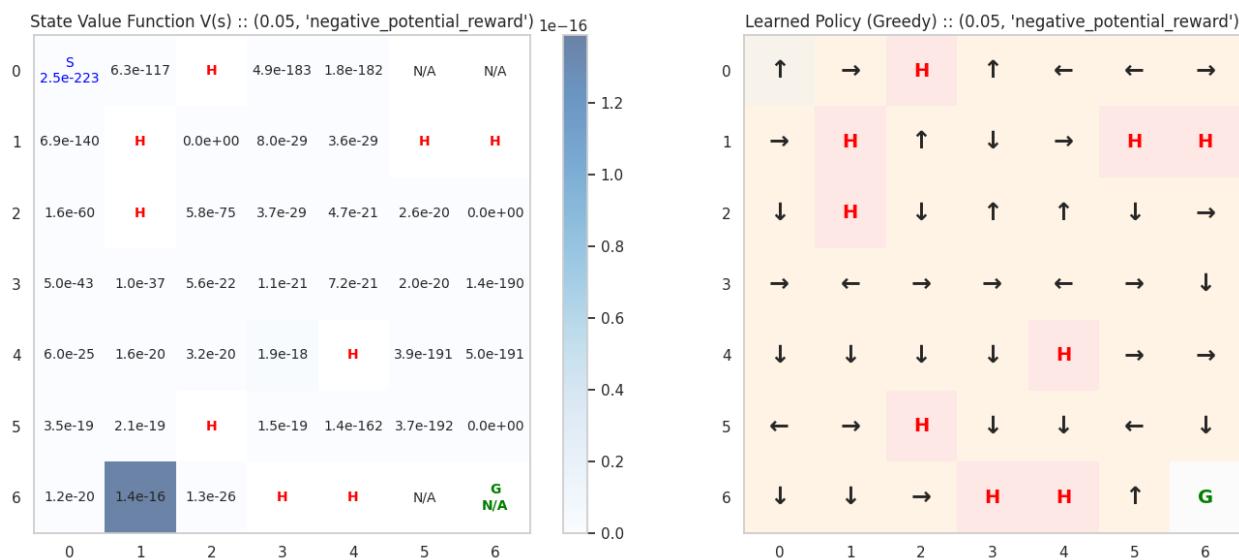
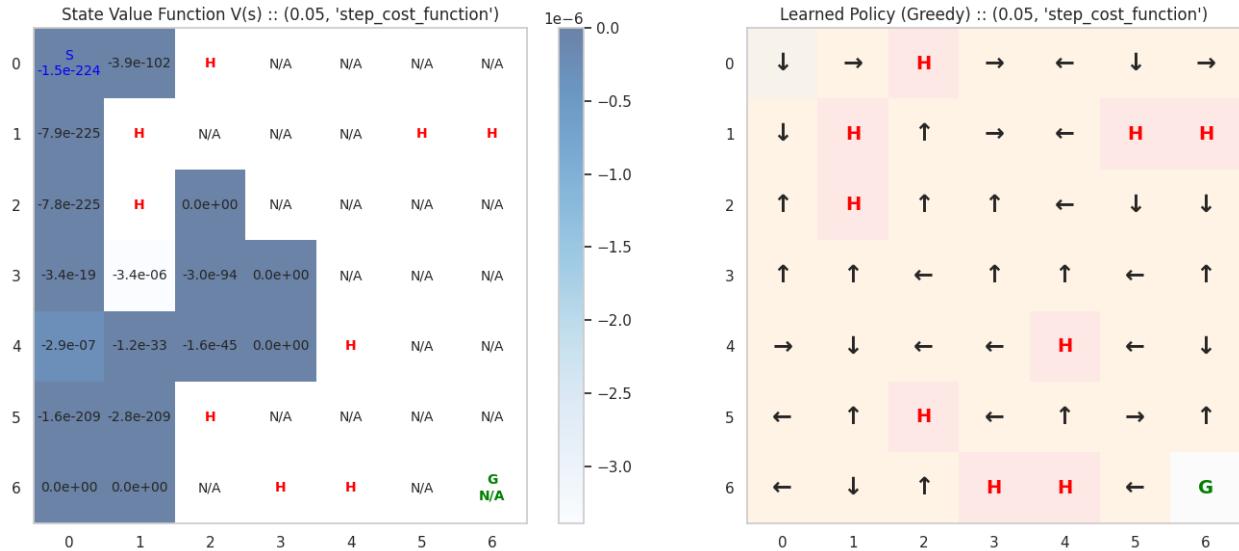
This shaping recognizes that in stochastic environments, an agent optimizing only for shortest path might take routes adjacent to multiple holes, risking failure from slippery dynamics. Safe Progress rewards finding paths that balance efficiency (getting to goal quickly) with safety (avoiding dangerous areas). This domain-specific design demonstrates how understanding the environment's characteristics can inform effective shaping strategies.

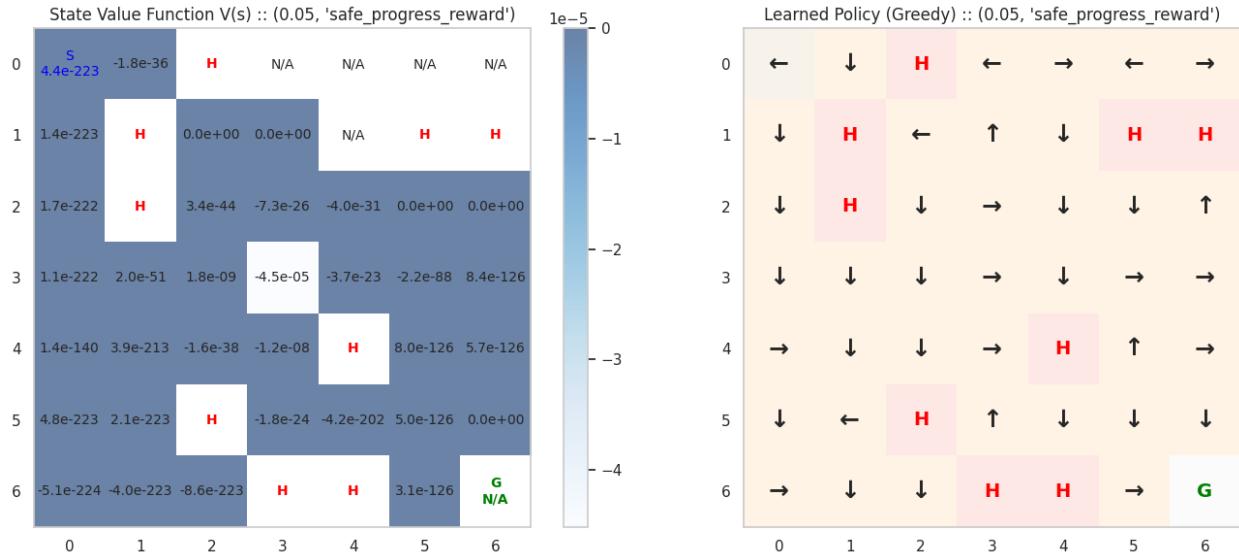
4. Experimental Results

All results represent aggregated statistics from 5 runs per epsilon, each run with a different reward shaping function, using MC algorithm and another 5 runs with the SARSA(0) algorithm. Total of 20 independent runs, of the every Visit Monte Carlo agent

4.1 $\epsilon = 0.05$ (MC)



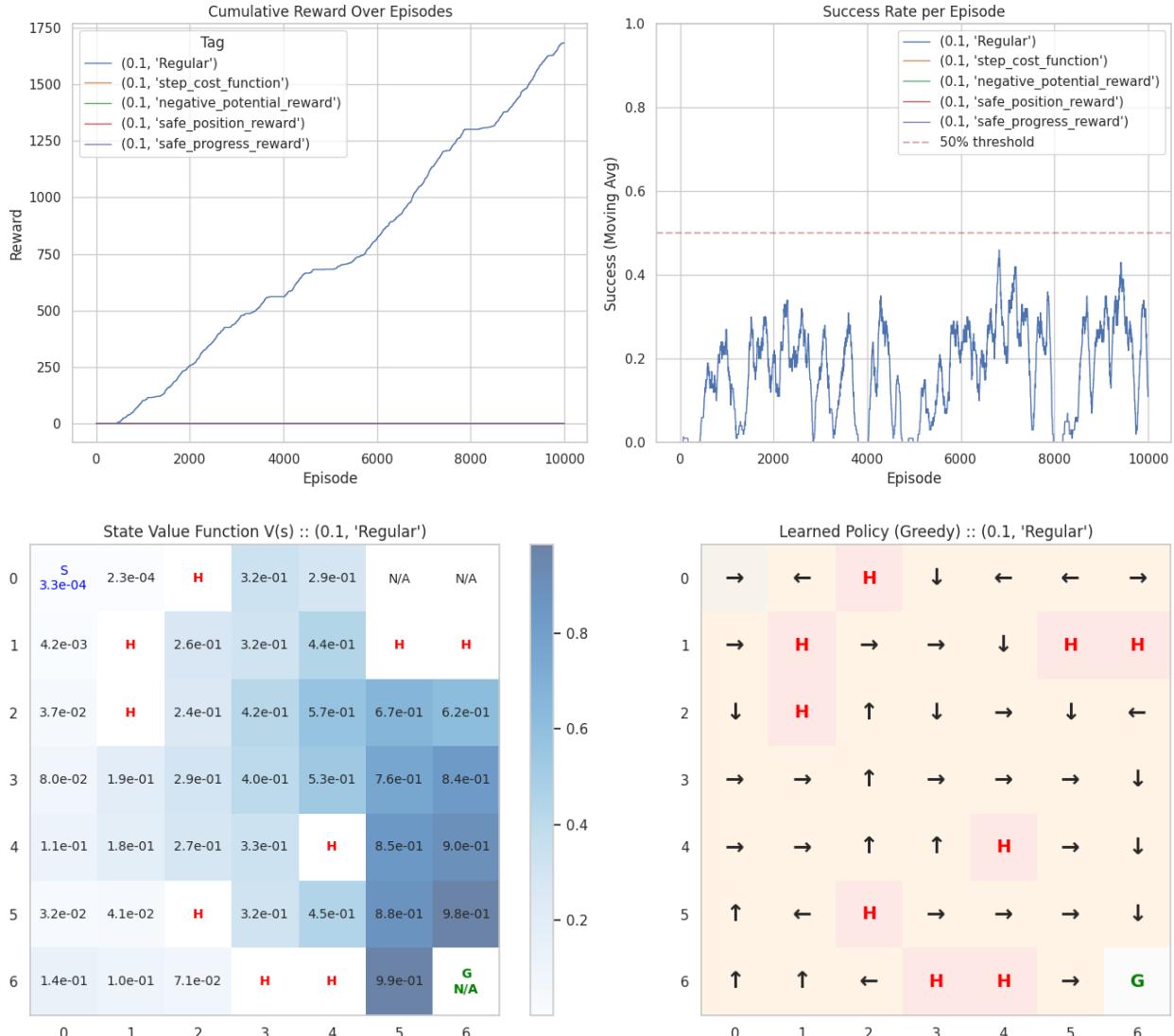


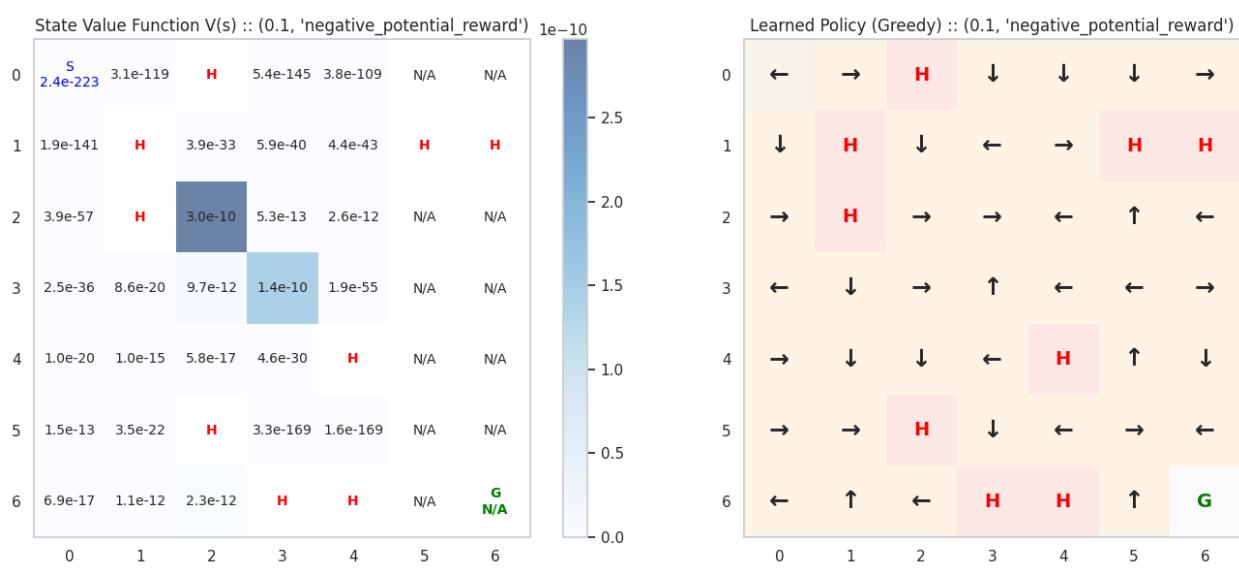
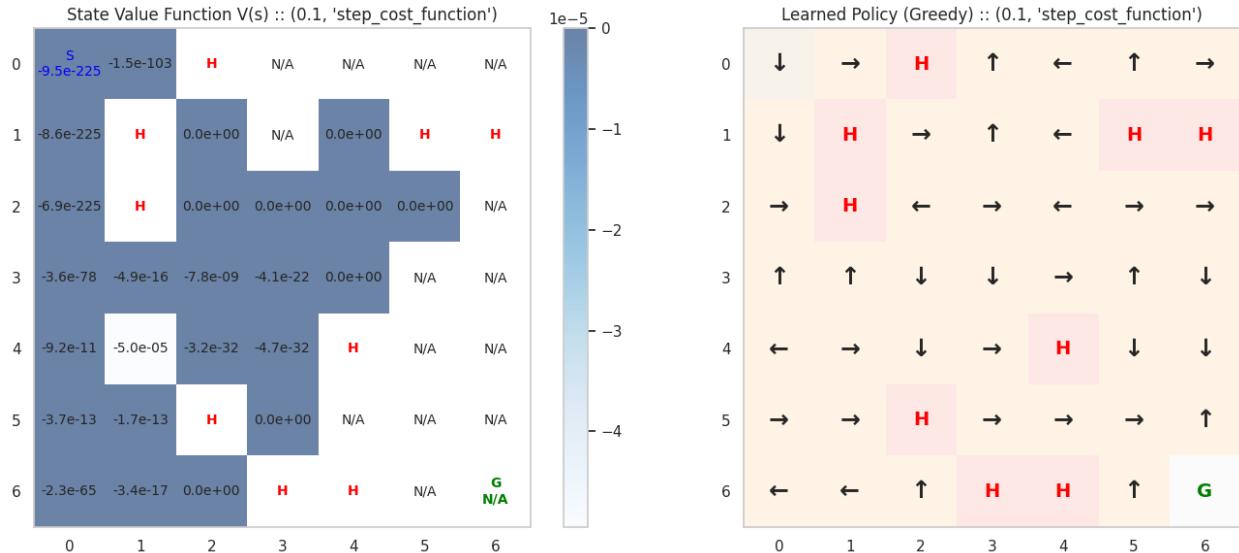


Conclusion:

It seems the regular method (no shaping) is the most optimized by its throughput., It seems that the step cost function agent and the negative potential function agent haven't reached the goal in any of the episodes.

4.2 $\epsilon = 0.1$ (MC)



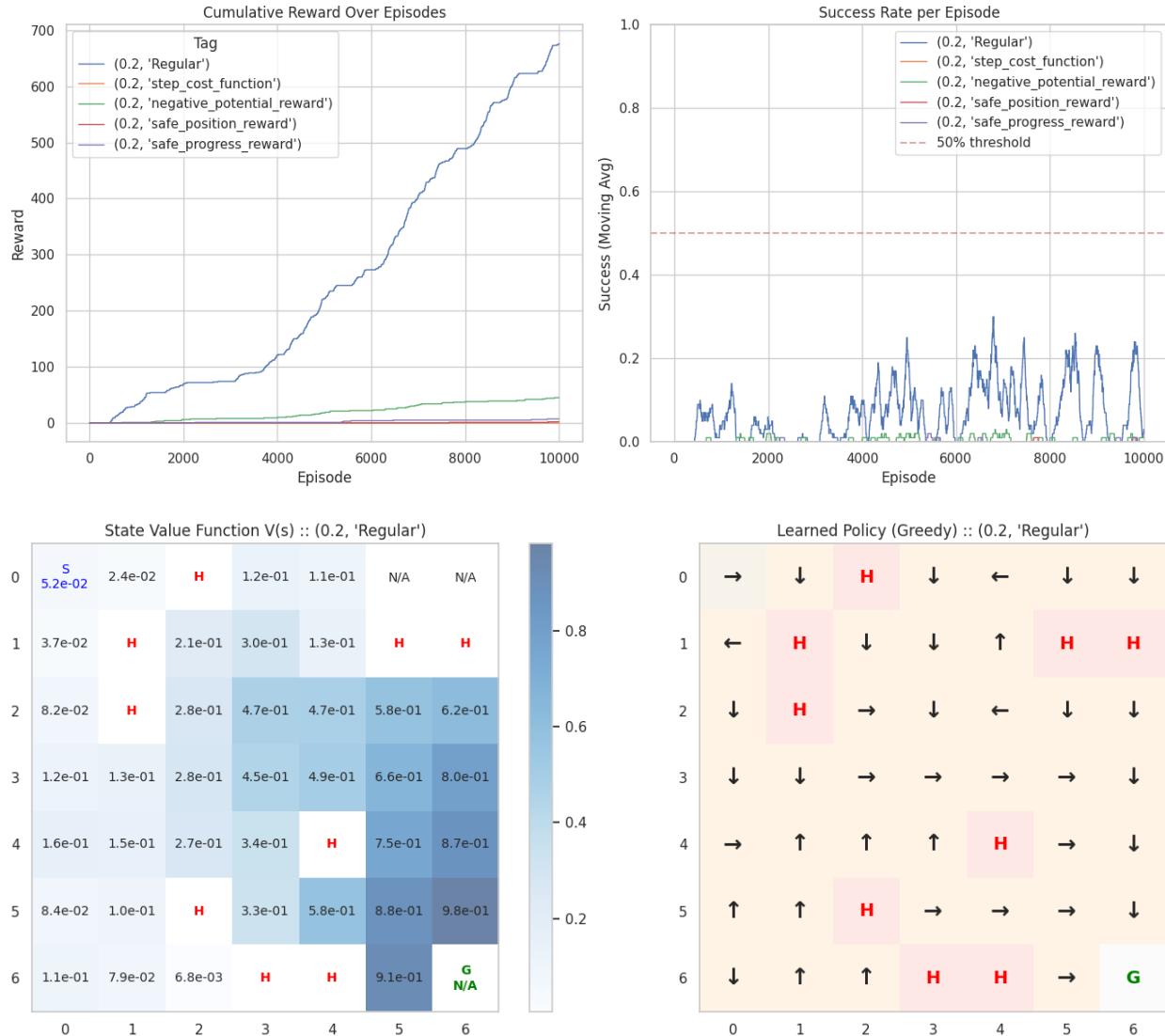


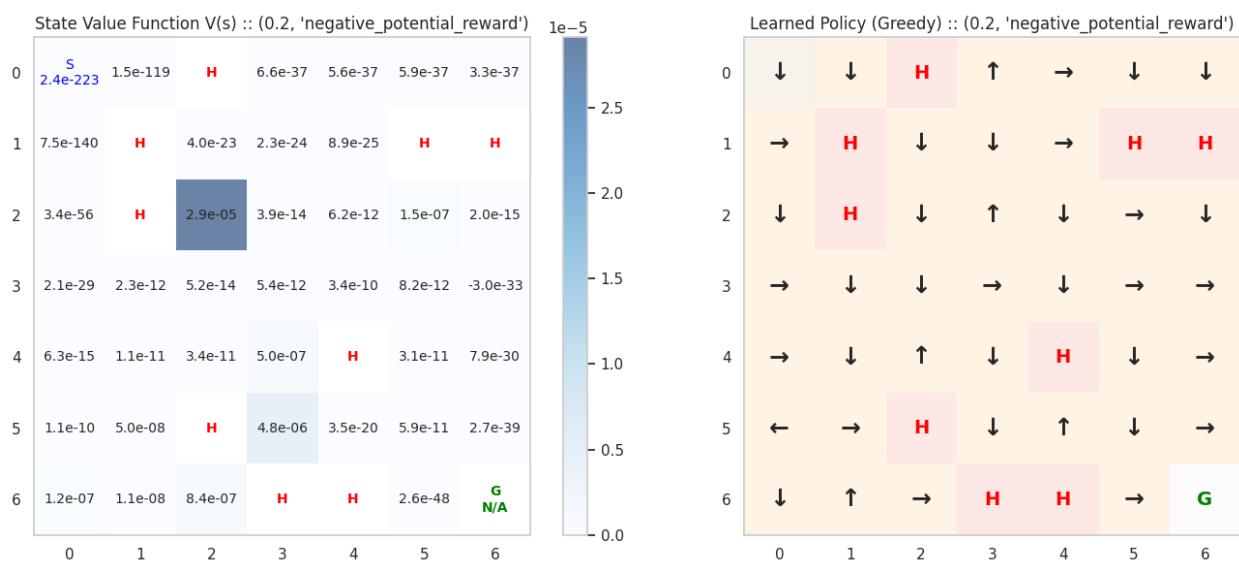
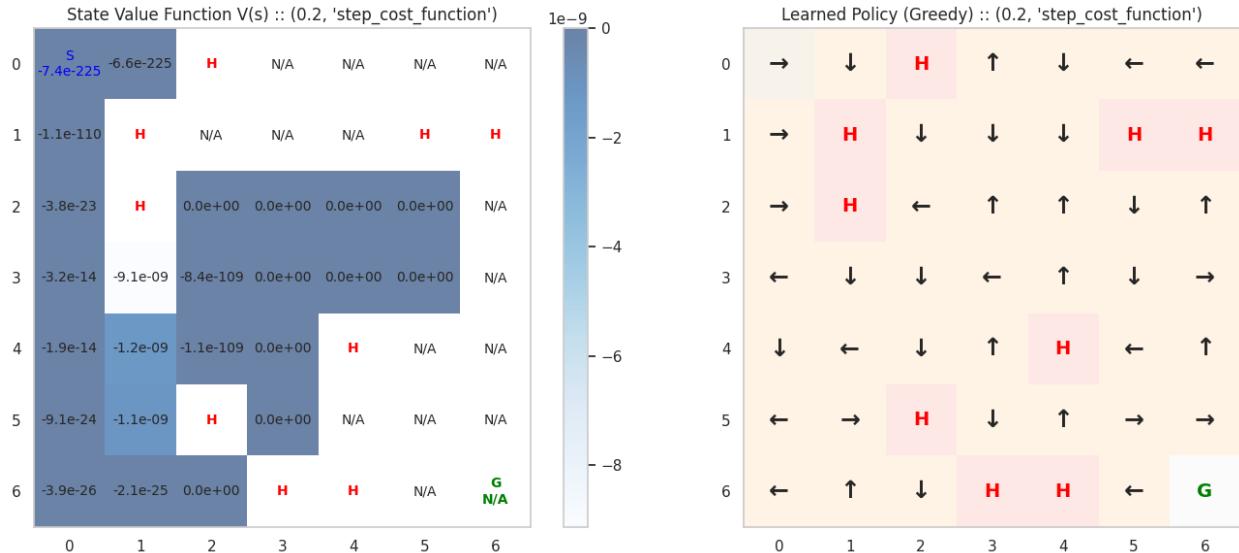


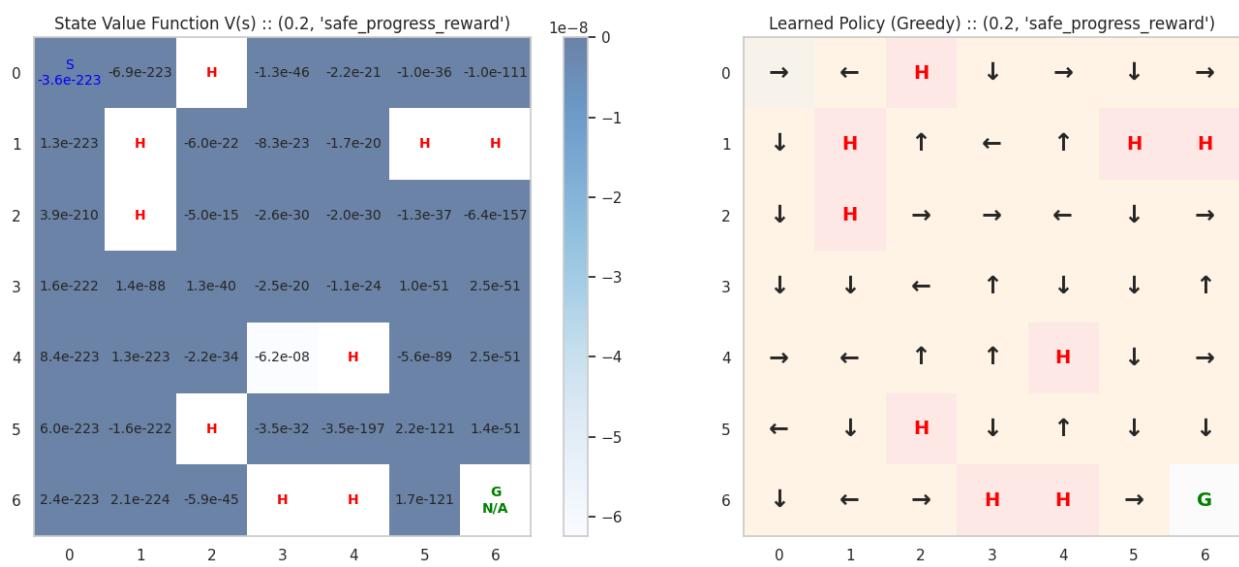
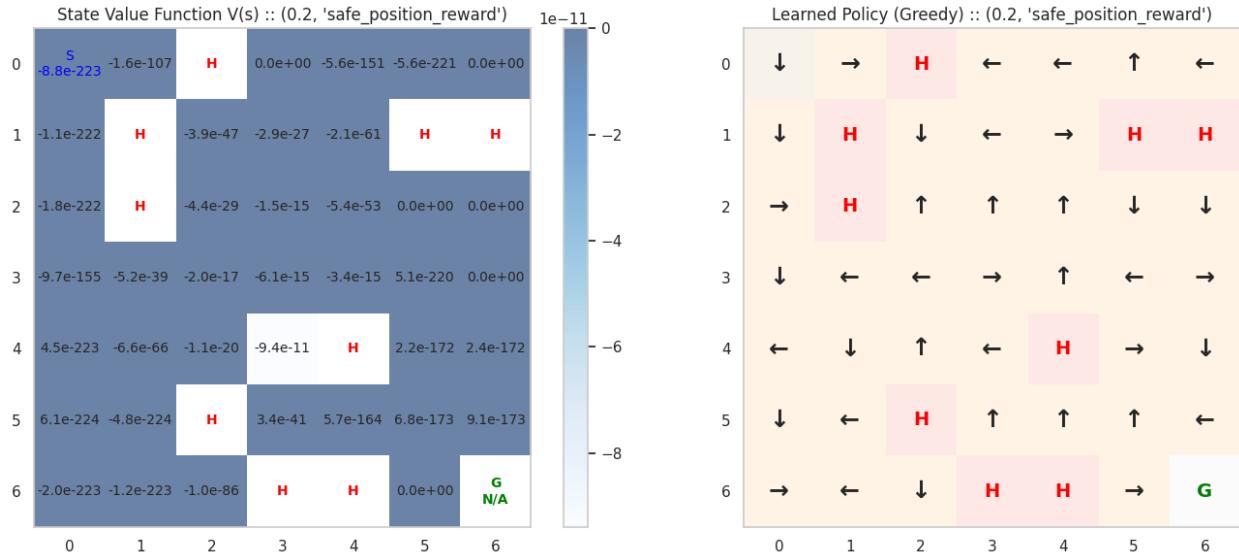
Conclusion:

It seems the regular method (no shaping) is the most optimized by its throughput. And all beside it and the safe-progress agent haven't reached the goal in any of their episodes. But we can also note the safe progress reward agent haven't reached the goal enough times to build a safe policy as we can see in states 5.3, 5.4 of its policy.

4.3.1 $\epsilon = 0.2$ (MC)



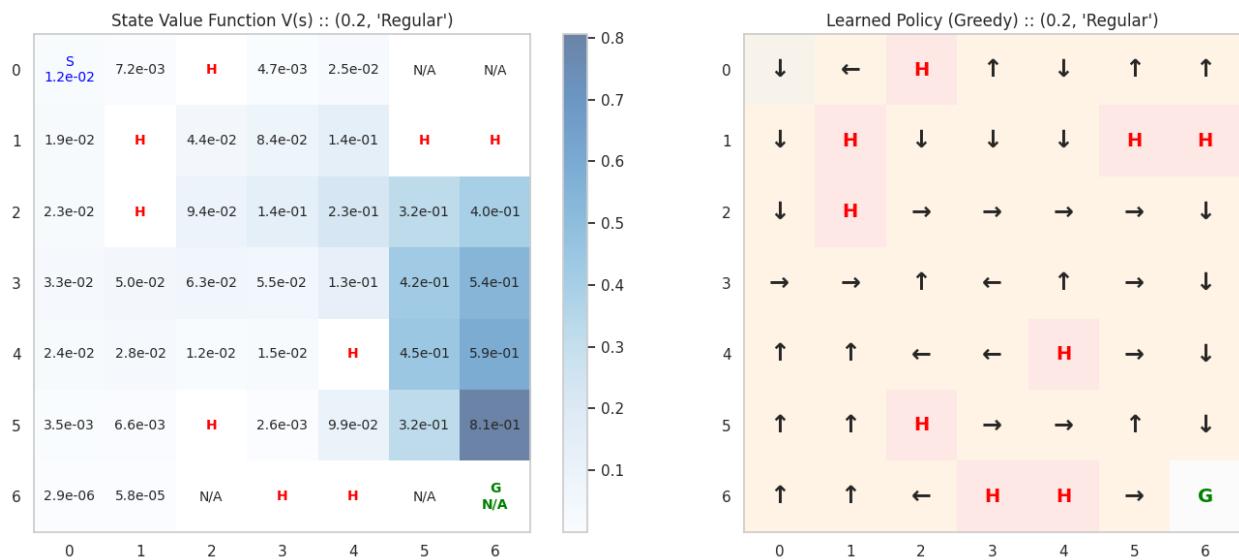
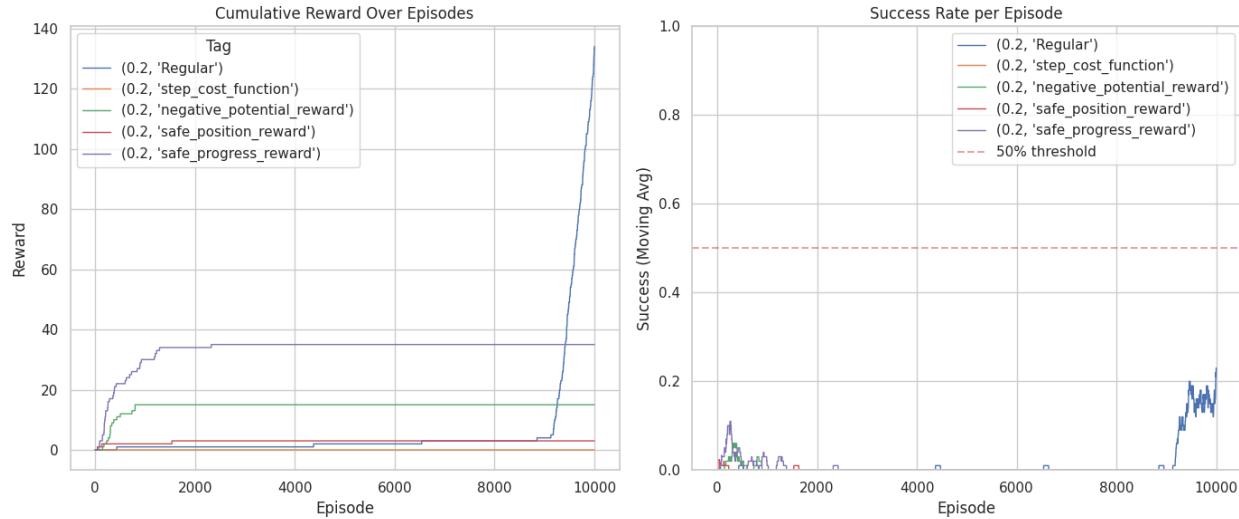




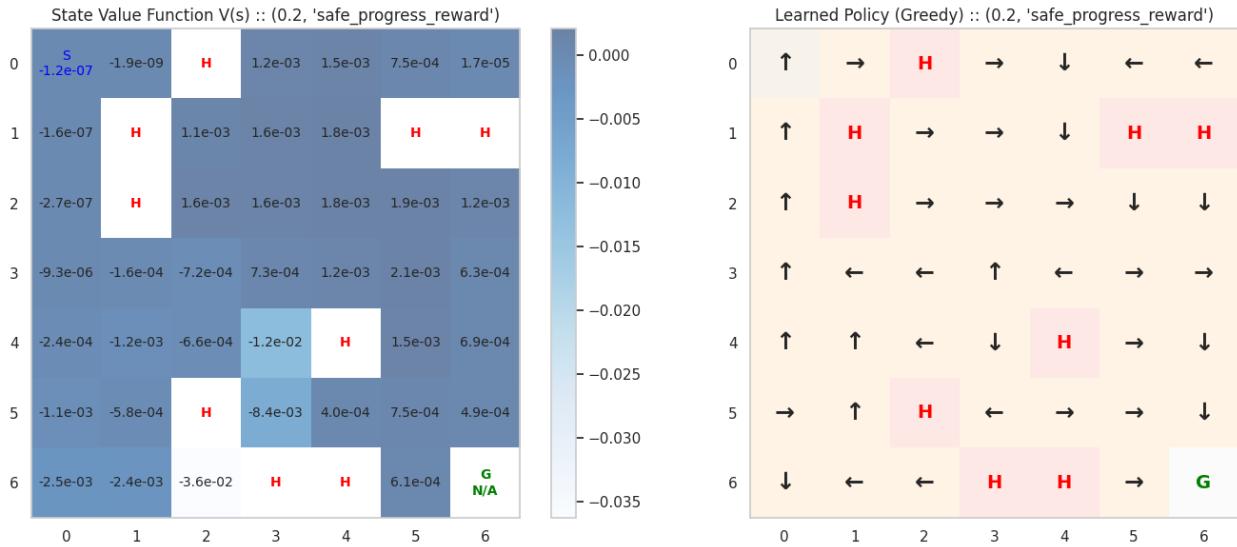
Conclusion:

It seems the regular method (no shaping) is the most optimized by its throughput. Compared to the previous epsilon-values, this one seems to be the most beneficial to the different reward-shaping functions, as we can see by the two graphs. but we can also note that their throughput was no where near to that of the regular's and it reflects in their policies.

4.3.2 $\epsilon = 0.2$ (SARSA)







Conclusion:

In the SARSA agent we see a better performance of the reward shaping progress (especially in Safe Progress), but still the regular has the best performance with the safest policy, unlike the other that don't have a safe policy that leads towards the goal.

5. Findings/Insights

5.1 Reward-shaping functions

5.1.1 Regular (no shaping)

Seems to be the most optimized in the Monte-Carlo agent with the hyperparameters used in the runs.

5.1.2 Step cost function

Seems to be ineffective based on the results of the runs, we theorize that the if the route to the goal is long, the agent gets unmotivated which causes it to not reach the goal.

5.1.3 Negative Potential

Seems to be ineffective based on the results of the runs, We theorise that the reason for that is if there is an available option, that leads to a closer state to the goal, based o Manhattan distance, even that state does not lead to the goal. The agent would still take it based on the shape reward, and we can see it in the results where epsilon was low (0.1, 0.05) and the agent didn't explore, it chose to take the action with the most reward and haven't reached the goal in any of the episodes.

5.1.3 Safe Position

Seems to be ineffective based on the results of the runs. We came up with the function hoping that it would encourage the agent to take safer routes towards the goals. In the runs we saw that agent, because of the safe position function, chose to not take step near holes , as we hoped, but as the routes to the goal had holes near them and near the start, the agent did not make progress and haven't reached it in any of the episodes.

5.1.4 Safe Progress

Seems to be the second most effective shaping method (with the first being no shaping at all), based on the results of the runs, we came up with the functions in hope that even if the route to the goal has holes near it, the Manhattan distance would still encourage the agent to take it. In the results we saw that it did not work, because along with the problem it also had the problem of the negative potential shaping function and thus the agent hasn't reached the goal in the most of its episodes.

5.2 epsilon parameters

For no shaping we conclude that a lower epsilon is optimized, as shown in the throughput graphs. For the shaping functions we have presented , we see that an higher epsilon increase the chance to reach the goal, causing the agent to explore more. Having said that it still does not make those functions optimal compared to no shaping.

5.3 MC vs SARSA

From the compression of 4.3.1 and 4.3.2 (runs with $\epsilon = 0.2$) we gather that while MC yields better results for the no shaping, SARSA(0) provides a tad of improvement for the custom Safe-Progress, we have defined but not enough to make it a valid option for a functioning agent, thus we conclude that the custom reward shaping functions we have defined, do not improve the performance of the agent, both in MC and in SARSA(0).

6. Conclusions

Our experimental results across 20 independent runs revealed several counterintuitive findings that challenge conventional assumptions about reward shaping in reinforcement learning.

We've identified specific failure modes for each shaping strategy. The step-cost shaping demotivated the agent on longer paths, creating a competing objective that conflicted with goal achievement. The potential-based distance shaping led to local optima where the agent gravitated toward states with lower Manhattan distance to the goal, even when these states didn't lead to viable paths. The safe position reward proved overly conservative, causing the agent to avoid necessary risks near holes, while the safe progress reward inherited both the conservatism of safety shaping and the local optima issues of distance-based shaping.

These results highlight an important consideration for practitioners: reward shaping is not universally beneficial and requires careful domain analysis. In environments with stochastic dynamics and sparse rewards, simpler approaches may outperform sophisticated shaping strategies. The effectiveness of reward shaping depends critically on alignment between the shaped rewards and the true objective, and misalignment can create competing objectives that degrade performance.

Quick Start Guide

Project Structure: Single Jupyter notebook (RL_Project_Reward_Shaping.ipynb) containing all implementations. Key components:

- MetricsWrapper: tracks episode metrics
- RewardShapingWrapper: implements all shaping strategies
- TrainableAgent: unified MC/SARSA implementation
- visualization functions: of the policy, throughput , state value function, sum of rewards.
- run_experiment: main execution function.

Installation: pip install gymnasium numpy pandas matplotlib seaborn tqdm

Library Versions: gymnasium==0.29.1, numpy==1.24.3, pandas==2.0.3, matplotlib==3.7.2, seaborn==0.12.2

Running the Code: Open notebook in Google Colab, run all cells in order to load functions and classes, configure hyperparameters in the main section (eps_arr, shaping_reward_func_arr, base_hyperparams), execute main() to run all experiments. Visualizations generate automatically after training. Fixed random seed (RAND_SEED=41432) ensures reproducibility, same configuration should produce identical results.