

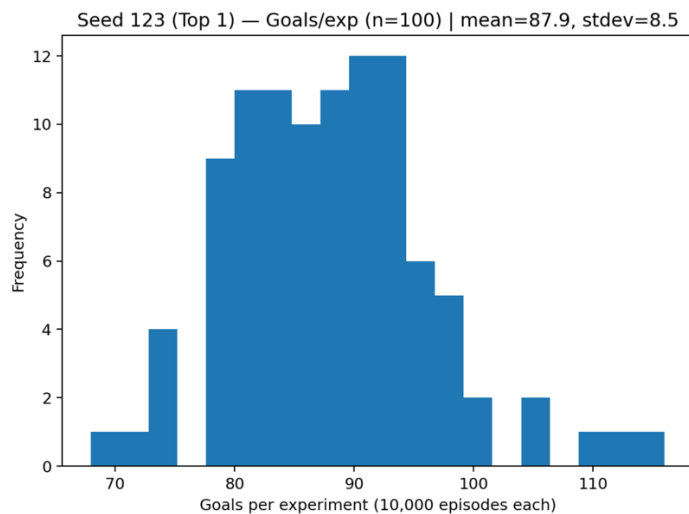
## CSC 380 - ASSIGNMENT 2

GABRIELLE BOYER-BAKER

[file:///Users/gabrielleboyerbaker/Desktop/CSC380\\_HW2/HW2\\_FrozenLake\\_Gabrielle.html](file:///Users/gabrielleboyerbaker/Desktop/CSC380_HW2/HW2_FrozenLake_Gabrielle.html)

### **PART 1 – random fixed policies**

#### **TOP 1**



#### Policy grid (8x8)

```
0 2 2 0 3 0 1 0
1 0 1 3 1 3 1 1
3 3 3 3 0 2 1 0
0 3 3 0 1 2 0 2
1 3 2 0 3 3 0 2
3 0 0 0 1 0 1 2
1 0 1 0 1 0 2 2
1 3 0 2 2 3 2 3
```

#### **Performance of top 1 policy:**

Mean goals = 87.90 stdev 8.5

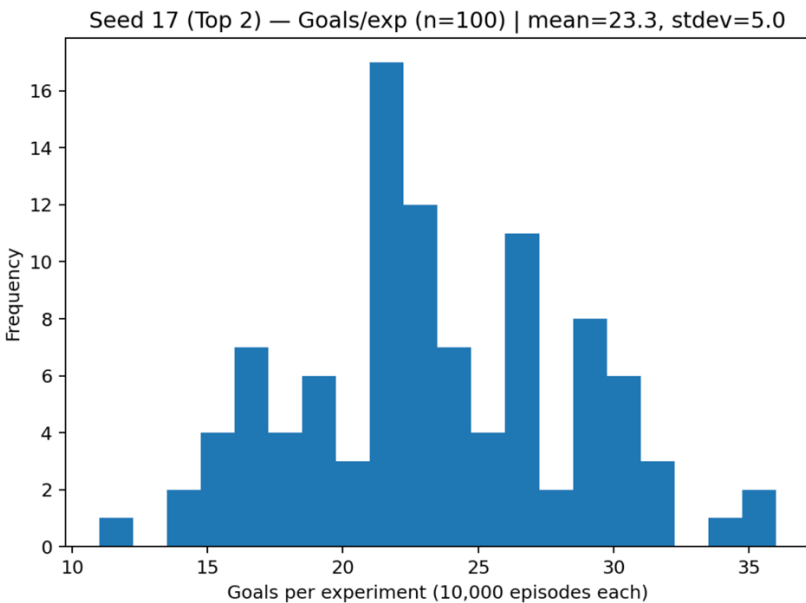
#### **Histogram summary:**

The goals in each experiment are distributed with a moderate spread, centered around 88 goals (out of 10,000 episodes per trial). Despite the stochastic "slippery" surface, the majority of runs exhibit consistent performance, clustering tightly between 80 and 95 goals. Most of the time, this approach appears to steer the agent toward the objective in a sufficiently safe manner while avoiding obstacles.

#### **Interpretation:**

Because its action pattern strikes a balance between exploration and safety, Seed 123's policy worked best. Its frequent "Down (1)" and "Right (2)" motions keep it moving in the direction of the goal while preventing lateral slips. Its somewhat longer but safer path still results in a high number of successes overall, as evidenced by its higher mean steps-to-goal ( $\approx 62$ ).

## TOP 2



### Policy grid 8x8

2 3 0 0 1 2 3 1

0 0 1 1 3 1 2 2

0 2 0 0 2 1 1 2

0 0 2 3 2 3 1 0

2 2 2 2 1 0 2 0

1 0 2 1 1 2 0 1

2 3 0 3 1 1 3 0

0 1 2 3 1 1 3 3

### Performance of top 2 policy:

Mean goals = 23.29 stdev = 5.05

### Histogram summary:

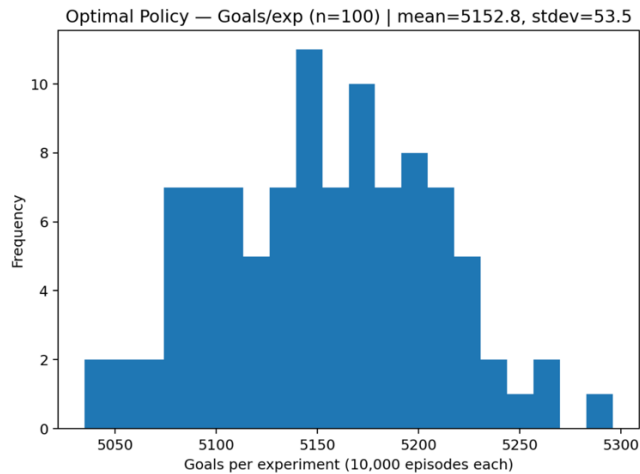
With a mean of roughly 23 goals and a smaller spread (stdev = 5), Seed 17 appears to be performing steadily poorly. Even after 100 experiments, the agent's goal count remains low because it regularly selects actions that result in holes or ineffective loops.

### Interpretation

This policy does not avoid danger states and appears to be less in line with the goal's location. When it does succeed, it reaches the target fast, as seen by the shorter mean steps-to-goal ( $\approx 53$ ); nonetheless, most episodes end prematurely because of gaps. All things considered, it shows how the predicted returns of random deterministic policies can vary greatly under slippery dynamics.

The difference between Seeds 123 and 17 demonstrates how random initial rules affect problems involving stochastic control. A policy's directional bias can result in significant fluctuations in average reward and stability even in the absence of learning. Policies that choose safe paths or implicitly "hug the edges" produce greater expected returns than ones that charge directly toward the goal because the surface of FrozenLake presents uncontrollable slippage. These outcomes establish a starting point for the Value Iteration algorithm in Part 2, which will methodically discover the actually best course of action.

## PART 2



### Optimal policy grid (8x8)

```
3 2 2 2 2 2 2 2
3 3 3 3 3 3 3 2
0 0 0 0 2 3 3 2
0 0 0 1 0 0 2 2
0 3 0 0 2 1 3 2
0 0 0 1 3 0 0 2
0 0 2 0 0 0 0 2
0 1 0 0 1 2 1 0
```

## Method Overview

I used the Value Iteration algorithm to determine the optimal course of action for Frozen Lake. Each state's value is updated periodically by this procedure until the values cease to fluctuate, indicating convergence.

As needed, I set the convergence threshold  $\theta = 0.0001$  and the discount factor  $\gamma = 1.0$ . With the exception of terminal states, which were set to 0, I also initialized the value of each state with a little random number. `env.unwrapped` was the source of the `env.unwrapped.P`, which lists every potential following condition and reward.

I determined the best course of action for each state by selecting the course of action that produced the highest expected value once the values had converged. In order to increase the agent's chances of achieving the objective, this policy instructs it on which way to go in each circumstance.

## OPTIMAL VALUE TABLE

0.998234	0.998311	0.998423	0.998544	0.998667	0.998784	0.998889	0.998965
0.998215	0.998274	0.998370	0.998485	0.998608	0.998732	0.998861	0.999011
0.996833	0.975258	0.923893	0.000000	0.855320	0.944988	0.980966	0.999099
0.995586	0.930791	0.798160	0.473507	0.622421	0.000000	0.943786	0.999227
0.994522	0.821689	0.539917	0.000000	0.538488	0.610467	0.851203	0.999388
0.993684	0.000000	0.000000	0.167631	0.382611	0.441741	0.000000	0.999577
0.993107	0.000000	0.193290	0.120299	0.000000	0.332168	0.000000	0.999784
0.992812	0.726108	0.459628	0.000000	0.277385	0.554773	0.777385	0.000000

It is confirmed that the agent's expected reward rises as it gets closer to the goal and falls near holes (represented by 0.000000 entries) as the values progressively rise toward the target state (bottom-right corner).

## **Histogram summary:**

Mean = 5152.8, Stdev = 53.5 over 100 experiments (each 10 000 episodes).

The optimal policy achieves over 5 000 successful episodes per experiment, a massive improvement compared to even the best random fixed policy (mean  $\approx 88$  goals).

## **Do the actions make sense?**

The optimal policy's actions make perfect sense because they obviously demonstrate prudent, risk-aware behavior. going up or right is preferred in the majority of top-row states, which keeps the agent focused on safer tiles and going toward the objective. The low V-values (around 0) surrounding dangerous regions demonstrate how Value Iteration naturally avoids edges and holes, while the highest values in the bottom-right corner verify that the agent is rewarded for achieving the objective. The agent's willingness to take little diversions to escape danger rather than going directly into dangerous areas is demonstrated by their occasional "up" and "down" movements. Overall, the policy strikes a balance between efficiency and safety, producing a dependable approach that works effectively despite the randomness of the slick surface.

## **Interpretation**

Value Iteration demonstrates the effectiveness of dynamic programming in reinforcement learning in contrast to the random rules in Part 1. Every state's expected return is maximized by the algorithm's deterministic, optimum policy.

Value Iteration averaged  $\approx 5153$  goals per trial, which is around a  $58\times$  improvement over the best random policy, which only managed  $\approx 88$  goals each trial.

This illustrates how uninformed random tactics are routinely outperformed by an explicit evaluation of future returns (via  $V(s)$ ).

## **Reflection**

I was able to make the connection between reinforcement learning theory and practice by working on this assignment. Since even little directional biases can drastically alter how frequently an agent reaches the target on a slippery surface like Frozen Lake, I initially didn't think random deterministic policies would perform so significantly from one another, but after running Part 1, it made sense. I was able to see how much randomness influences results in stochastic contexts by seeing how the averages and histograms differed among seeds.

Although it required some debugging to get correct, I think my favorite element was the Value Iteration part. It was rather satisfying to watch the values progressively increase

toward the objective when I finished printing the final V table and setting up the two-table update. The algorithm seems to be picking up "common sense" on its own, traveling more frequently to the right and down but still taking safer detours when necessary. Realizing how effective dynamic programming may be was made clear by witnessing the ideal policy obtain thousands of successes as opposed to just dozens previously.

It took a lot of time to run the large-scale experiments. It took roughly 21 minutes to finish each whole Part 1 run with 10,000 episodes per experiment because I'm using a MacBook Pro with an M1 chip. It was intriguing to observe how rapidly the performance increased after the code was optimized and operating natively on Apple Silicon; despite the simulations' high workload, it managed it effectively.

Managing the runtime and ensuring that the environment calls were accurate with the new Gymnasium API was the most challenging aspect overall. Determining when to reset, how to accurately track awards, and how to prevent it from printing excessively required several trial runs. But when everything went smoothly at last, the outcomes were quite evident and satisfying.

If I were to redo this project, I would definitely use vectorized simulations or progress bars to speed it up. I might also try varying the discount factor ( $\gamma$ ) to see how it affects the behavior of the policy. Additionally, I believe it would be entertaining to use arrows on the grid to represent the policy or to animate an agent moving in response to the actions; this would further simplify the interpretation of the outcomes.

Overall, I gained knowledge about how transition probabilities, convergence thresholds, and iterative updates function in real-world code rather than simply on paper. I now have a far better understanding of what "optimal" in reinforcement learning actually means and how value-based algorithms may transform haphazard wandering into wise decision-making.