Markov Decision Process

Wan, Yuet Wing Frank  
ywan33@gatech.edu

***Abstract—***To understand Markov Decision process reinforcement learning, two problems were set up, including a frozen lake of 16 states and gambler’s problem of 1001 states via python’s Open AI’s gym module. Value iteration, policy iteration, and Q-learning was then performed on the two problem sets to find the optimal policies. The policies from those two methods were then plotted and compared based on results, iterations to complete, and time to complete.

## Framework Setup

The testing framework of this problem was constructed through a learner class and the environment problem through the OpenAI’s gym module. The class contains all the necessary training and testing methods in addition to the learning environment. The framework of this class was inspired by other similar projects and modified for further investigation of the project requirements, including adjusting gamma values and convergence boundaries for learning iterations.

Due to some version conflicts with the Gambler’s problem, a specific version of GYM was installed and stored on the poetry lock file. However, this did not resolve the problem with Q-learning, and was not implemented in the final iteration.

## Frozen Lake Environment

Frozen Lake was set up for an environment of the reinforcement learning. The Frozen Lake question was selected due to the ease of implementation due to the limits of the four actions, is a simple way of testing a real-world scenario of a person navigating a through thin frozen lake, ease in maze simplicity, and ability to approach the solution using the Q learning reinforcement learning approach through iterations. This frozen lake problem was selected as it while being small enough for simplification purposes.[[1]](#footnote-1)

The 4x4 grid of the environment was initiated within the MDP class for the reinforcement learner to iteration to find the optimal policy.

## Gambler’s Problem Environment

A gambler is making bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of $1000 or loses by running out of money. On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler’s capital, the actions are stakes, and the reward is zero on all transitions except those on which the gambler reaches his goal, when it is +1. The state-value function then gives the probability of winning from each state while a policy is a mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.[[2]](#footnote-2)

This problem was selected as it contains 1001 states to satisfy the large number of states in the problem and is not a grid problem. This problem was reproduced from the Gambler’s problem as described in Chapter 4 of Sutton’s Reinforcement Learning book and is undiscounted, finite, episodic MDP so gamma is 1, and the terminal state is 1000[[3]](#footnote-3) which was specified via Gambler.py.

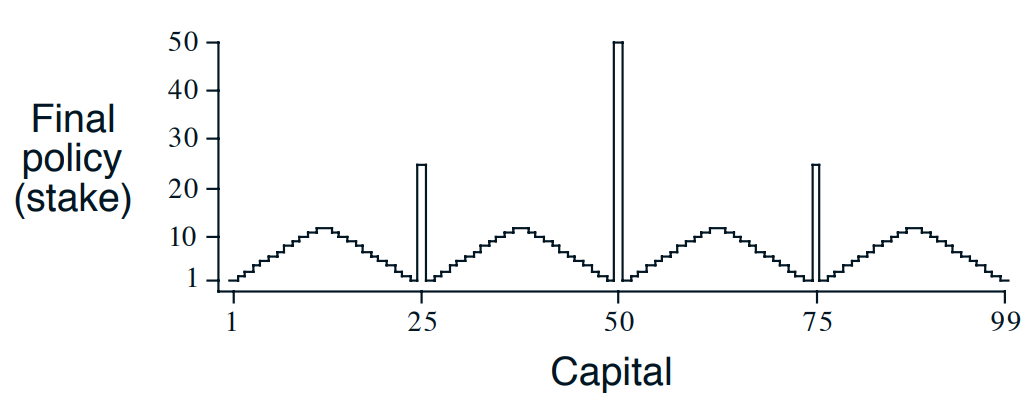


Figure – Gambler’s problem from Reinforcement Learning (Sutton)

Copied from figure 4.6 of the textbook, this figure illustrates how to maximize the chance of hitting the reward of reaching state 100 (1000 in our scenario) by betting different amounts at different states, where the user should bet all their holdings at state 50 and so forth.

## Value Iteration

The value function is a measure on how rewarding it is for an agent to be in each state, or how good it is for it to perform a given action at each state and is calculated by the value iteration method.[[4]](#footnote-4) The iterations and deltas were then calculated through finding the number of iterations needed for the solution to converge, where the result is said to have converged when delta is below 0.00001. To find the delta for each iteration, all four possible actions for each state is examined through policy evaluation.

Policy evaluation leverages the environment p method, which stores for each state (16 in total) and action (4 directions in total) a list (3 in total) which further contains a list that stores information for that action was performed at that state. The list of actions is 3 components as the actual direction taken is stochastic and has a chance of going to the adjacent direction and contains information on the probability of transitioning to that state, the next state, corresponding reward, and if that action terminates the game.

The first three variables were stored and summed up through the value iteration formula derived from the Bellman Optimality Equation:

Prob \* (reward + discount\_factor \* state\_value of next state) or

The Bellman Optimality Equation tells how expected return from original state s plus action a, with gamma set to 0.9 as suggested by documentation to balance between learning quickly and retaining old values.[[5]](#footnote-5) This value was then stored for each of the possible transition states of each action-state then repeated for each possible action of that state, before finally stored for the Q table in object V through the Bellman Optimality Equation, updating the value function of the state with the highest state-action value. This is performed for all 16 states of the environment until the difference between the old and new state values after each iteration is smaller than the convergence threshold of 0.00001.[[6]](#footnote-6)

After the value iteration, convergence occurred on iteration 59 in 0.0663 seconds. Below shows the change in values as iterations increase.

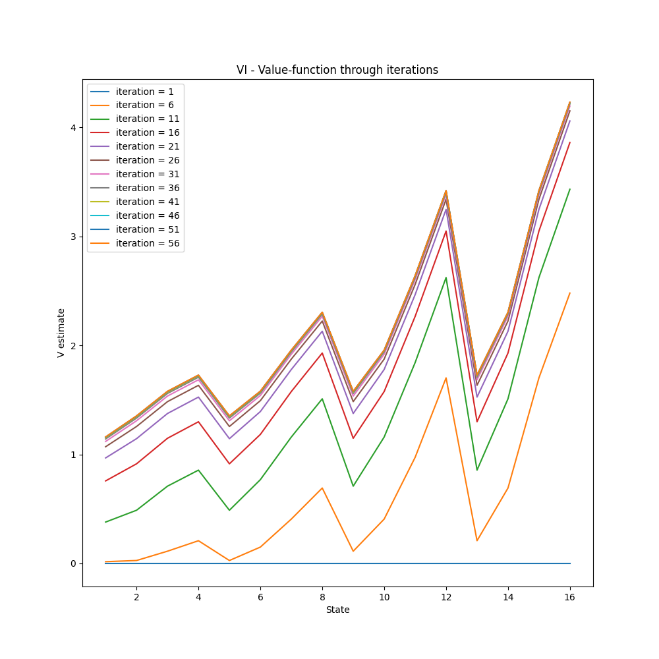
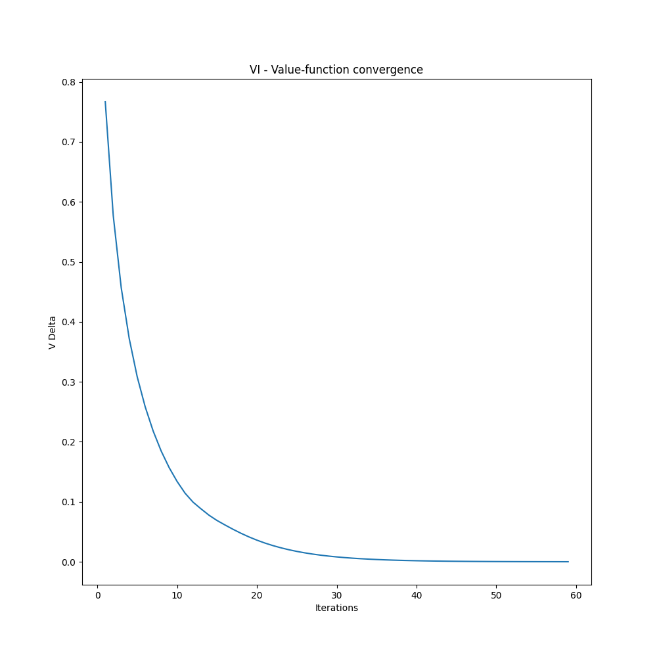


Figure - Delta (left) and Value function (right) of the value iteration over iterations

One the left is the graph of the deltas of policies over iterations, while the right figure shows the value of each state over iterations using the Bellman optimality equation. The value of each state increases logarithmically slowly due to the discount factor, while the delta value function decreases logarithmically as iterations’ state-action value become more similar over iterations. Interestingly, the value function never dips below 0, as would be expected when the next action leads to the hole in the ice.

The value iteration for the gambler problem had some difficulty converging, as when gamma was set to 1, the solution never converged despite increasing the convergence threshold. This may be contributed As such, gamma was later adjusted to 0.9 with delta maintained as 0.0001 like the frozen lake problem. After this change, the problem converges at iteration 17 in 36.0423 seconds.

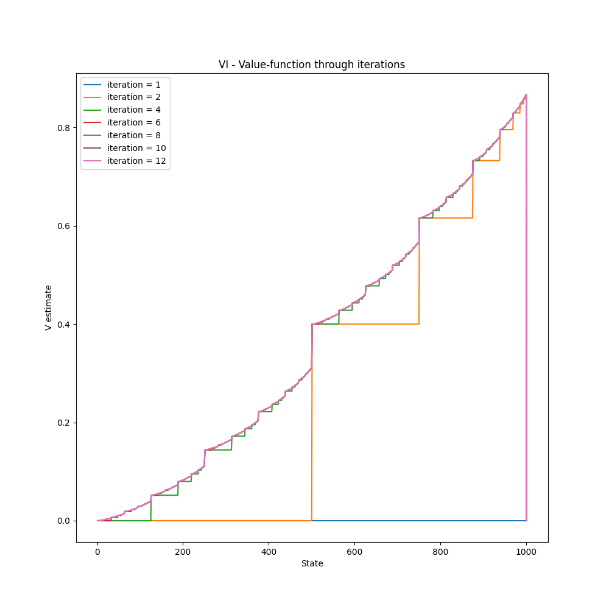


Figure - Value function of states over iterations of the Gambler’s problem

The number of iterations it takes to reach convergence was surprising, as it took approximately 4 times less iterations for the results to converge despite having more than 20 times the number of states and actions. This makes sense as the probability of state 500 reaching a reward of state 1000 will not change over iterations, as the probability of the gamble will always remain 50% and the state-value of that state remains constant. Rather, the iterations only help with populating the value function of the different states as the reward is only reached by one state only, before propagating down towards other states. As the value-function does not change over time, the V difference between iterations would quickly converge as a result.

The time for convergence was nearly 100 times longer than the frozen lake environment, which was mostly caused by the sheer number of states and values each iteration had to calculate through. This result was therefore expected.

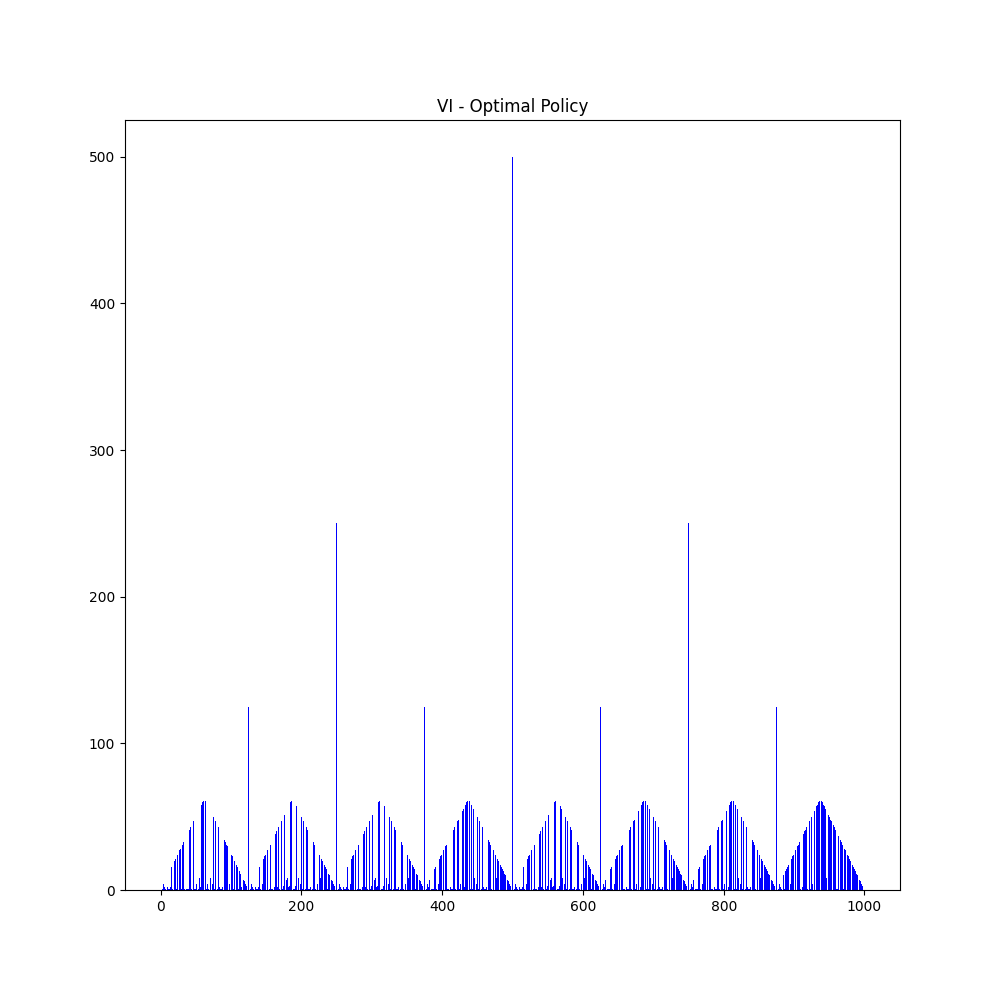
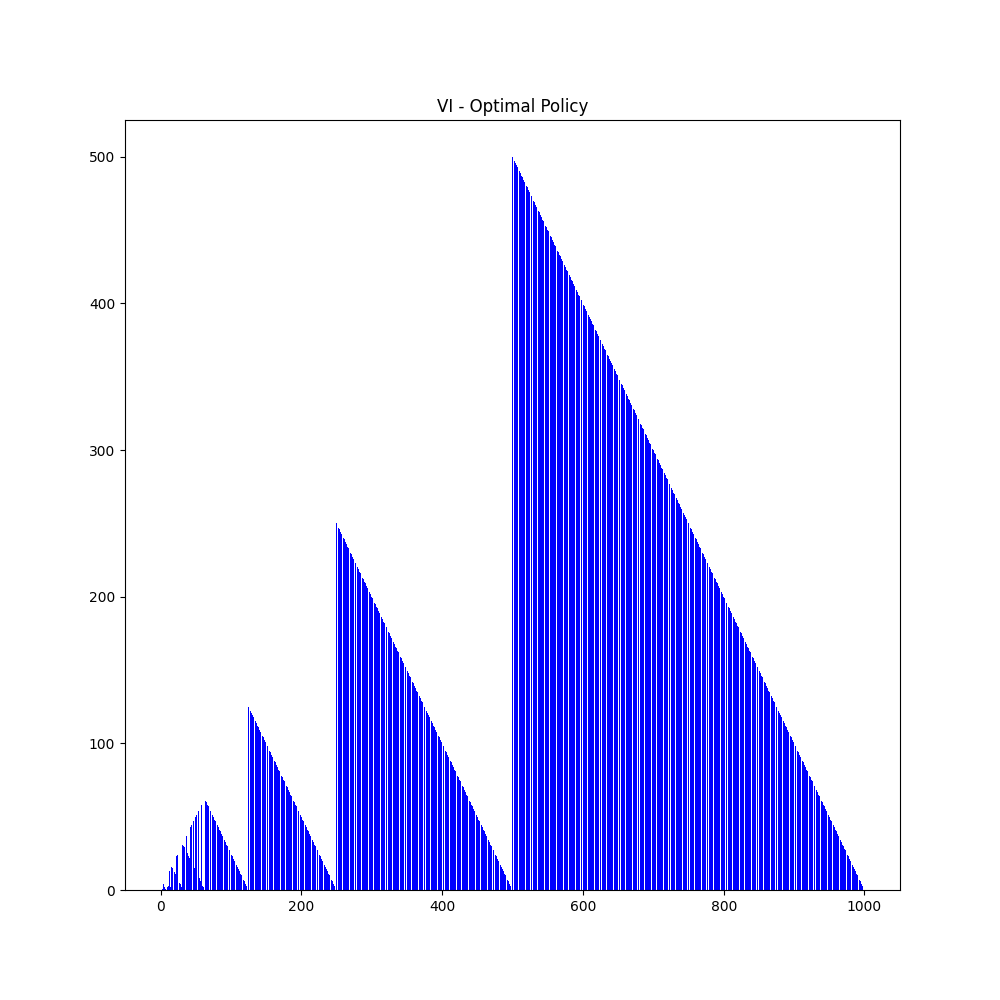
 

Figure - An initial policy when gamma is 1 (left) and when gamma 0.9 (right)

When attempting to train the policy with gamma equal to 1 as the problem is undiscounted and finite, the delta of the iterations never converged likely due to too many fluctuations between the iterations from each action-state value summation. Additionally, the utility of sequences assumes how the gamma should be below 1 which may impact the simplification of calculating of the reward and subsequent delta.[[7]](#footnote-7) Without the gamma to dampen the Q value in the Bellman Optimality Equation, the deltas are never lowered enough to reach the convergence threshold.

However interestingly, left portion of figure 4 mirrors in figure 1 from the textbook of the gambler’s problem, where the policy attempts to reach states of intervals 50 to reach states of 500, whereas in the problem where gamma is 0.9, the policy attempts to reach states 150, 250, 500, and 1000 and linearly decreasing afterwards to reach the next state of importance. The large difference in policies were solely affected by the gamma discount factor, where the reward of the iteration is affected by the previous iteration. This suggests how over time, states that are not between intervals of 50 have a higher likelihood of reaching a reward by going as soon as possible to state 1000 rather than jumping to the interval states. This is exemplified by the states from 501-999 that prefer to frog leap to the closest intervals of 50 when gamma is 1 but prefer to jump straight to state 1000 when gamma is 0.9.

## Policy Extraction

A Reinforcement Learning policy determines the probability of the action that the agent will take in each state, resulting in the optimal policy for that robot.[[8]](#footnote-8)

With the value function of all states is stored after convergence, the policy can then be extracted by policy iteration method which calculations the state-action values of all possible actions from that state and choose the action with the highest state-action value as shown in the equation above. The optimal action-value function is as follows:[[9]](#footnote-9)

This is evaluated by policy\_improvement method and checks to see if any of the most likely action to perform at a state differed from the policy from the previous iteration as improved by policy\_evaluation method.

The following plots show the initial and final states of the policy:

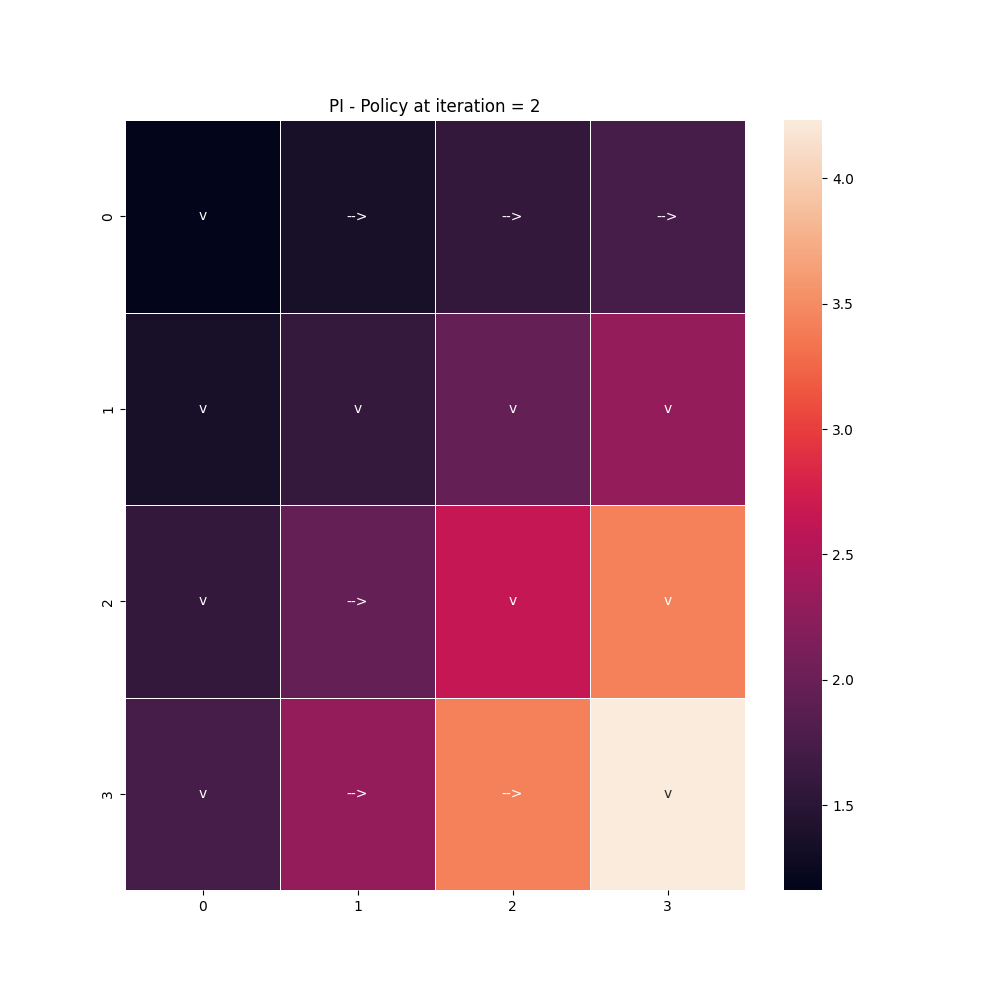
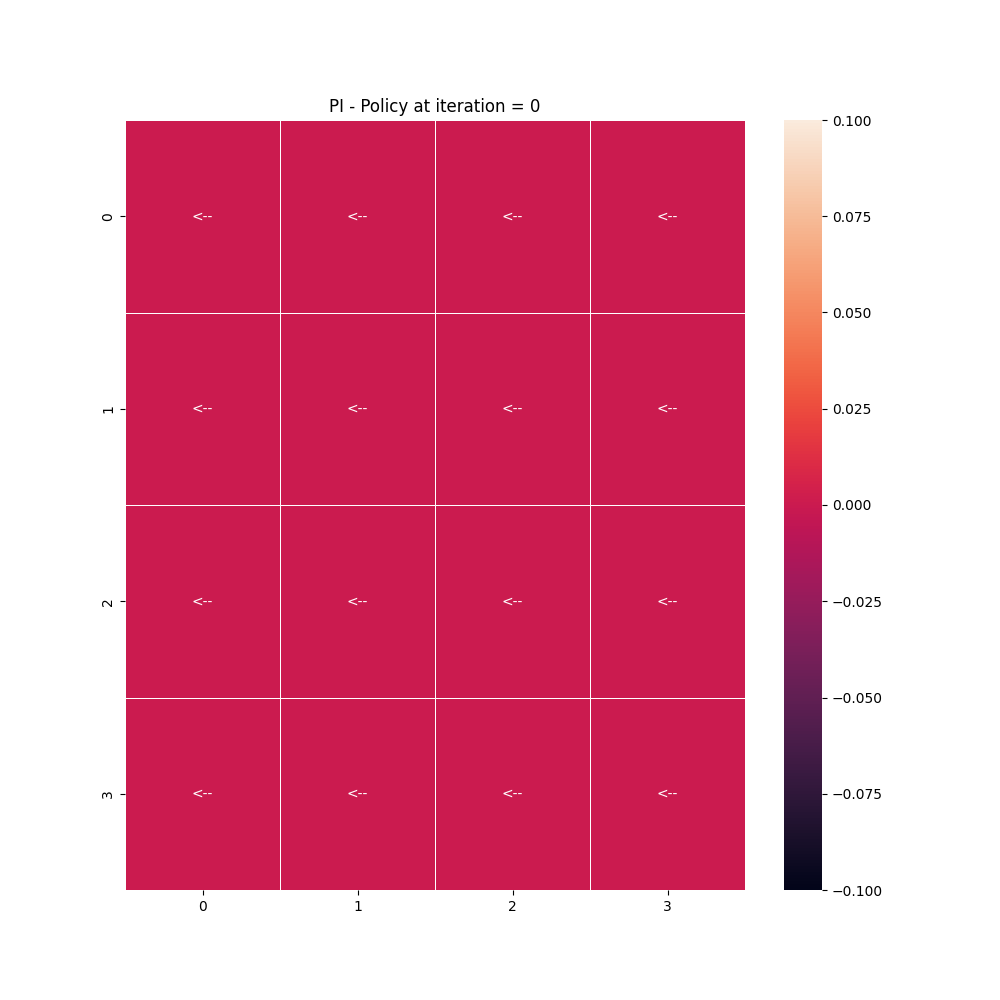


Figure – Policies of the Frozen Lake problem at first and last iteration

The difference between the initial and final policy is pictured in the grid in figure 4, where the rewards are distinguished with the highest reward at the bottom right grid. It’s also interesting to note how the reward scale does not dip below 0 in the final policy iteration, which is surprising as the location of the holes should reap a negative reward, although this may be circumvented as the positive actions can balance out the state-value value and figure 2 also suggests some states are close the holes in the ice. However, the final policy never appears to actively point towards a spot with a potential hole in the ice, but rather steadily heads towards the bottom right goal and all the scores gradually increase in value. This suggests that the environment was not implemented correctly, but due to time constraints this was never fully debugged.

As for the gambler’s problem, the 1st iteration and optimal policy for the action at a state is shown below:

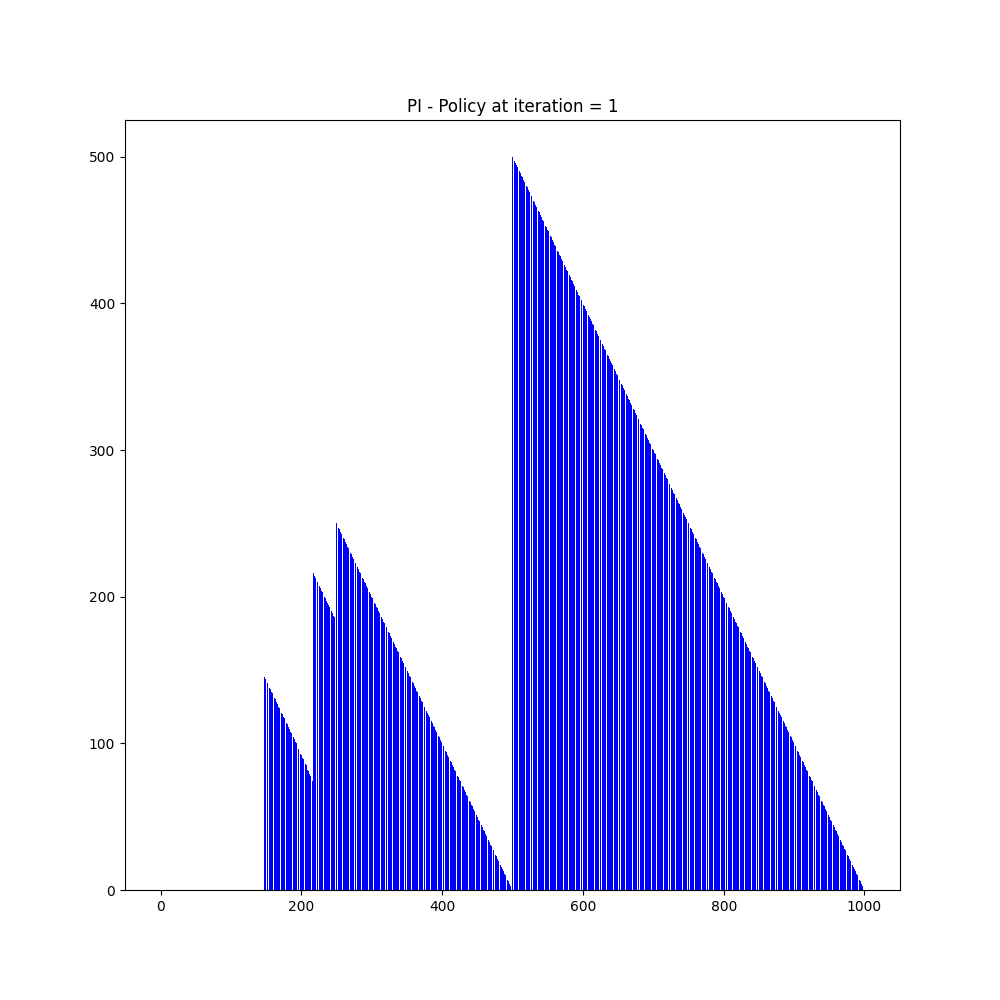
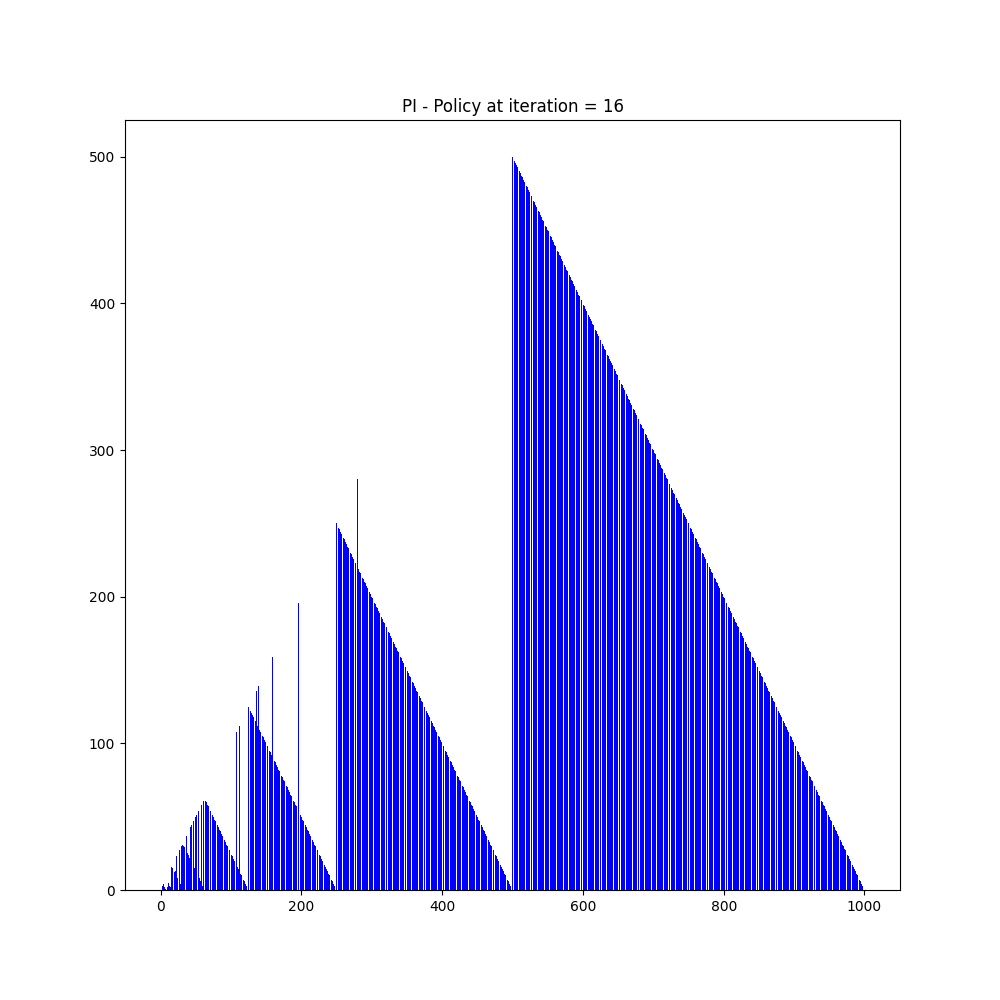
 

Figure – Initiation (left) and final (right) Policy for the Gambler’s problem

As seen in the left graph of figure 5, it is expected how the policy suggests an action of betting 500 to 1 linearly as the state moves from 500 to 999 to reach the goal of state 1000 quickly, while the states prior to that attempt to reach state 500 as soon as possible. The portion from state 250 and above does not change from the first policy to the last optimal one, but the policy does change below that due to the insufficient funds to perform larger actions.

There also appear to be some out-of-line peaks on the right graph of figure 6, which is confusing as those trends did not exist on the first iteration, and don’t adhere to the interval pattern seen throughout. The action remains within the reasonable bounds of its state and may likely be attempting to save the possibility of losing the game by going over the threshold state of 500 as much as possible. However, this trend is not prevalent.

## Q Learning

Q-learning was used to traverse through the frozen lake environment to establish a similar policy and state-value function to that of value and policy iteration. Iterations are performed until the number of max iterations is reached. The Q learning iterates from the start state until it either reaches the goal or falls into a hole and iterations are limited to 10000 for the frozen lake.

The Q learner finished training in 11.2260 seconds, taking much more time to finish than that of the value iteration. However, this may be skewed as implementation of the Q learner was not set to stop iterations after policy convergence due to missing delta calculations of the policies.

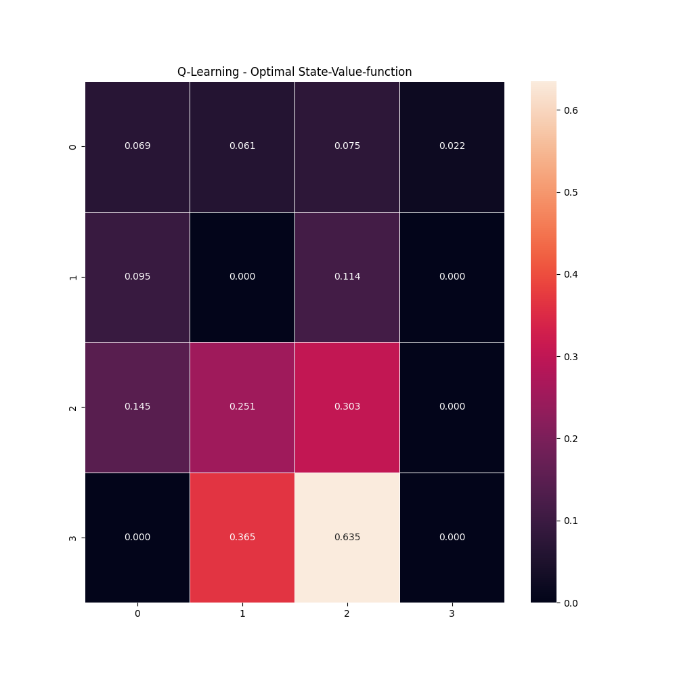
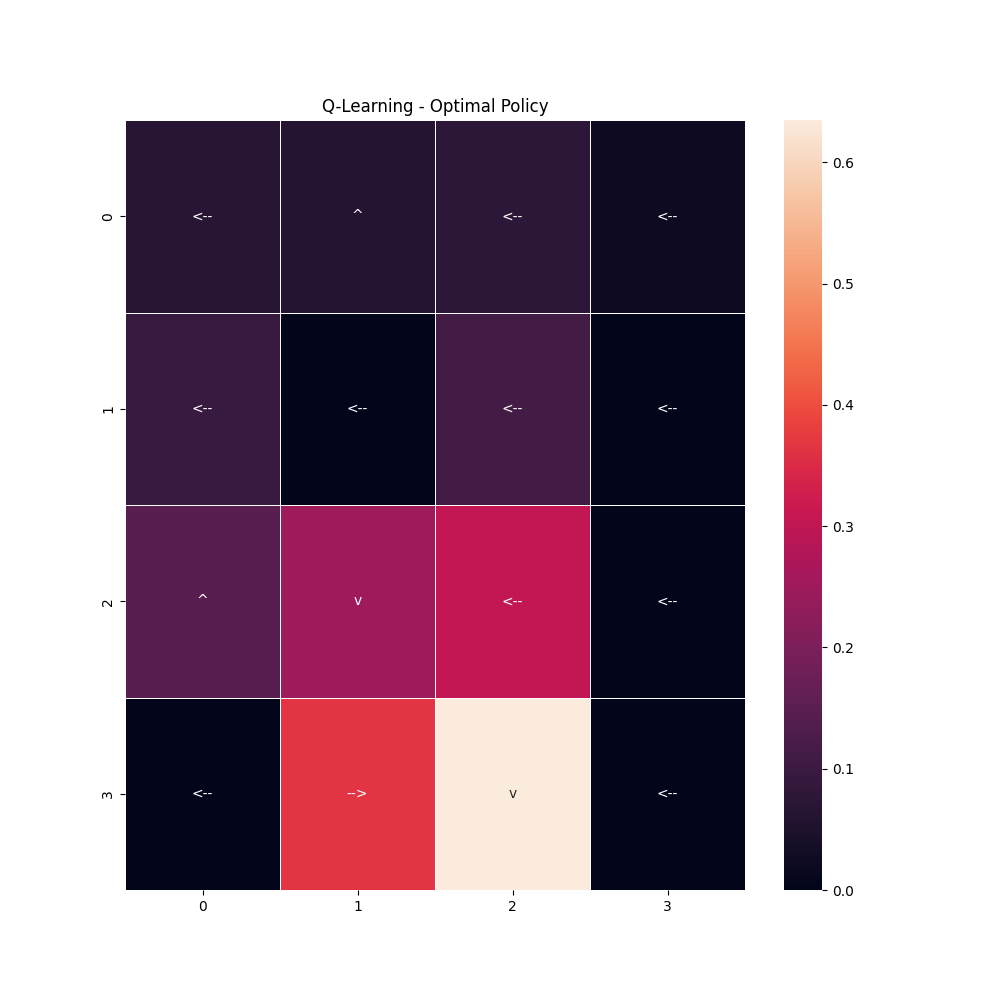
 

Figure – Q-Learner’s policy for frozen lake

Comparing figure 5 with figure 7, the policies between the two approaches are very different, with the policies and state value functions being much more pronounced from the Q learner. The state-value functions signal the locations of ice holes much better, with the 0.000 values suggesting their locations. Many of the recommended actions make sense, attempting to point away from those holes like the first value of the second row. Since the process is stochastic with a chance of going to the adjacent left or right of the intended direction, that grid point points towards the wall to avoid the hole at all costs. However, there are also some points of confusion where some states like the third value of the second row points towards left into a hole rather than downwards towards the goal. Additionally, state 16 does not appear to be the goal, while state 15 does, suggesting that more investigation needs to be done on the correct implementation of the frozen lake environment.

Q learning was unsuccessfully implemented for the gambling solution due to a bug in the .env.step() method of the gambling environment. Different gym versions were tested but causes environment issues in the frozen lake problem[[10]](#footnote-10), and thus was not fixed.

## Conclusion

Some problems remain in the implementation of this experiment which needs to be address in future iterations, including the reward allocations of the frozen lake problem and the Q-learning implementation for the gambler’s problem. The former issue was highlighted through the grid policies from Q learning and policy iterations, where the rewards and recommended actions were not consistent, and the function values were not expected as the final goal was expected to be on the bottom right grid square. However, figure 7 suggested otherwise, instead with grid 15 having the highest value function.

Additionally, the Q-learner should have a stop check to prevent iterations after policy convergence by checking to see if any recommended actions change after iterations. This would limit unnecessary iterations and shorten run time to compare with the policy iteration method. However due to time constraints this was not implemented.

1. https://gym.openai.com/envs/FrozenLake-v0/ [↑](#footnote-ref-1)
2. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf [↑](#footnote-ref-2)
3. https://medium.com/@jaems33/gamblers-problem-b4e91040e58a [↑](#footnote-ref-3)
4. https://deeplizard.com/learn/video/eMxOGwbdqKY [↑](#footnote-ref-4)
5. https://zsalloum.medium.com/basics-of-reinforcement-learning-the-easy-way-fb3a0a44f30e [↑](#footnote-ref-5)
6. https://medium.com/analytics-vidhya/solving-the-frozenlake-environment-from-openai-gym-using-value-iteration-5a078dffe438 [↑](#footnote-ref-6)
7. https://edstem.org/us/courses/17142/lessons/27909/slides/160662 [↑](#footnote-ref-7)
8. https://deeplizard.com/learn/video/rP4oEpQbDm4 [↑](#footnote-ref-8)
9. https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/ [↑](#footnote-ref-9)
10. https://github.com/mimoralea/gym-walk/issues/3 [↑](#footnote-ref-10)