José Tristani

Charles Isbell

CS-7641

November 28, 2021

Markov Decision Processes: Analysis of Block Dude & Maze MDPs

To discuss and analyze the strengths and weaknesses of MDPs, two different MDP problems have been selected for analysis due to their common use to showcase MDPs and highlight their potential for surprising users and use in the video games industry among others. The problems selected are the traditional Block Dude problem from ____ in both small and large state space variants and a custom grid world MDP configured as a Maze to show how the different methods of Value Iteration, Policy Iteration and Q-Learning solve them.

The Block Dude problem configurations are one of 500 states and one of 15000 states, a 30x increase of the small version, and as shown in the figures, the environment from small to large, while larger, is not that much more complex than the initial one.

The Maze problem was selected to showcase the potential pathfinding capabilities of the different solvers, from VI to Q-Learning, the most interesting of which is Q-Learning, which as will be shown, manages to find the optimal path to the goal consistently after a few iterations.

In both problems, a custom Decaying Epsilon Greedy policy was used, this was to get the correct amount of exploration in the initial episodes of training the Q-Learners and to achieve consistent results when plotting the results of the learners in a steady state.

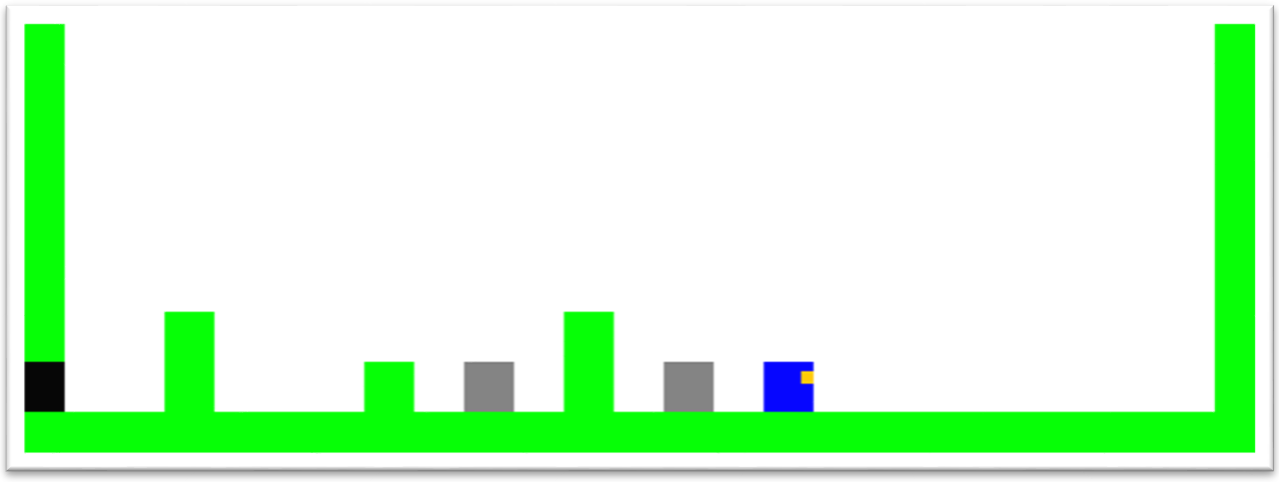

Figure 1: Small Block Dude Level (~500 states)

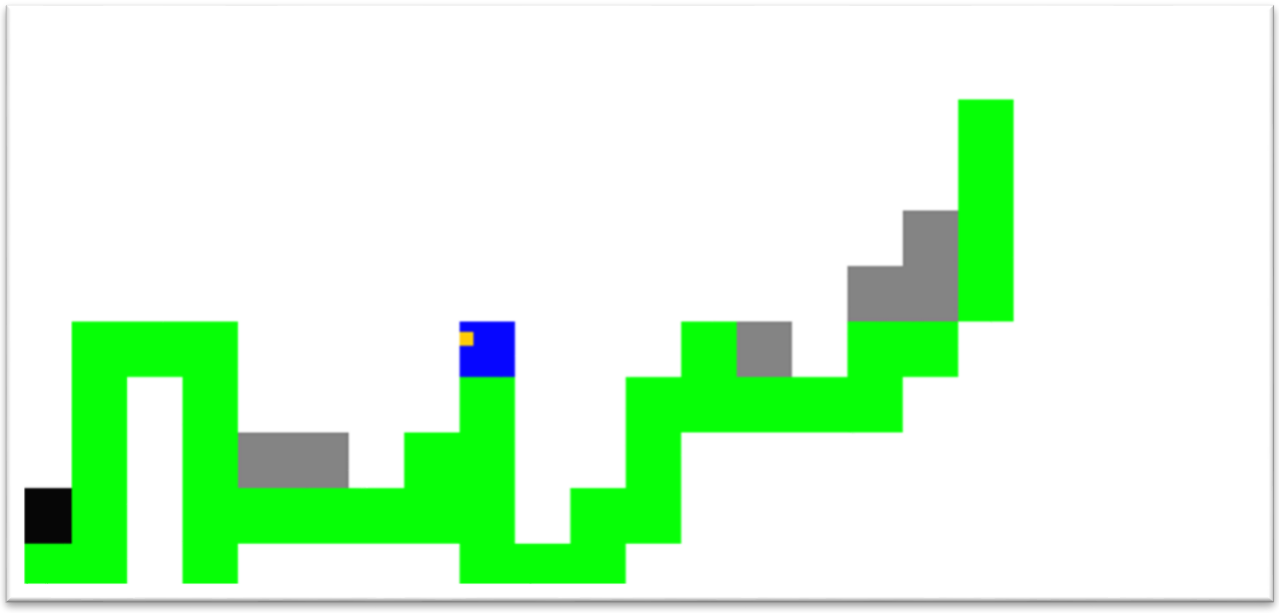

Figure 2: Large Block Dude Level (~15,000 states)

As shown in Figure 1 and 2 above, the different Block Dude levels are not that much different and yet the number of states in the MDP grow substantially, getting the policy with Value Iteration for instance takes 418 iterations for both variants, a very curious result which must be due to the convergence criteria, which is when the policy has a net change of less than 0.00015.

For Policy Iteration, it takes in all cases twice as long to compute the policy and it completes in 16 iterations for the small variant and 47 iterations for the large variant. In both cases the resulting policies are the same, but values in some states could vary by as little as ±0.01.

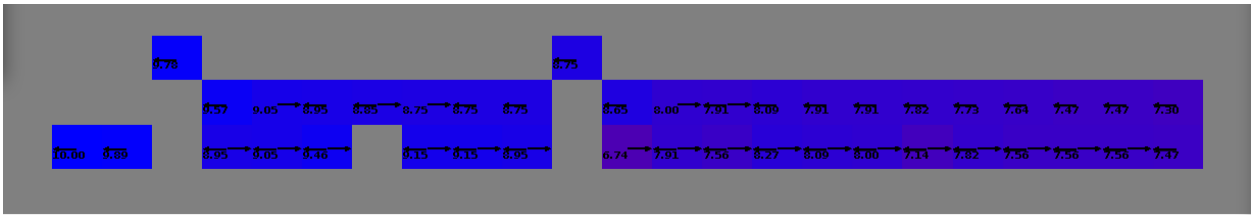

Figure 3: Block Dude Policy Iteration for Small Variant

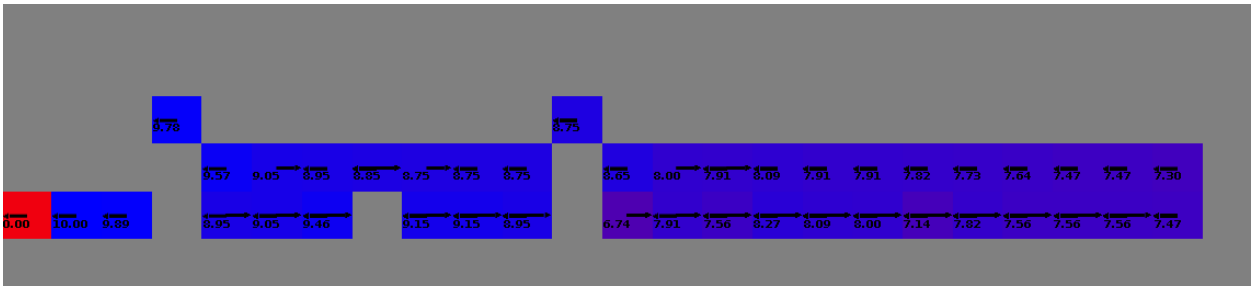

Figure 4: Block Dude Value Iteration for Small Variant

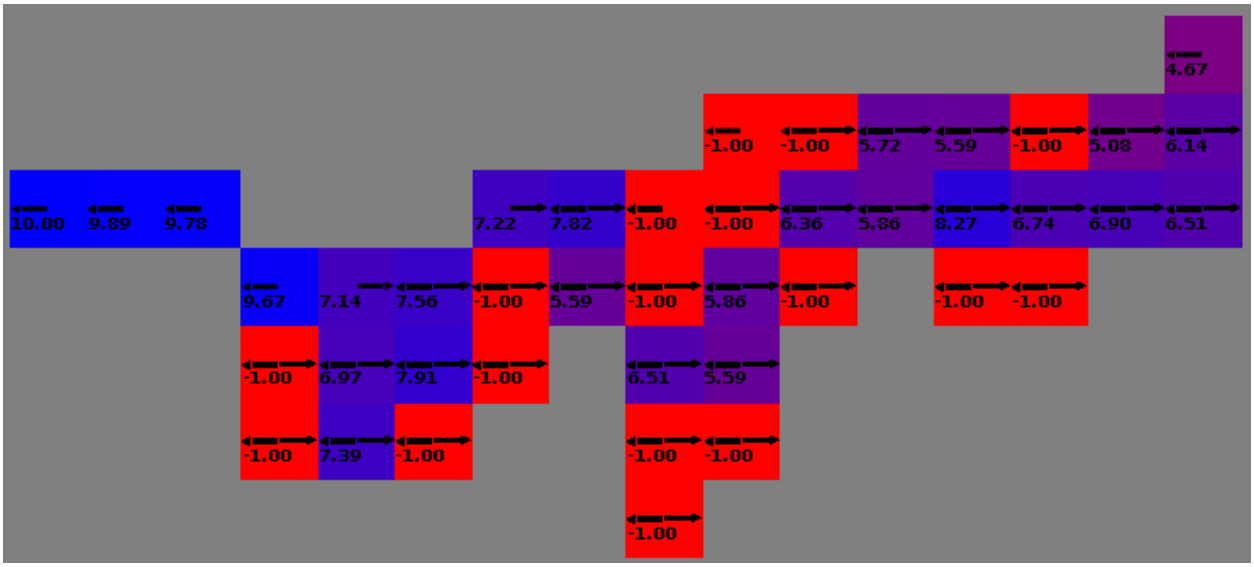

Figure 5: Block Dude Policy Iteration Large Variant

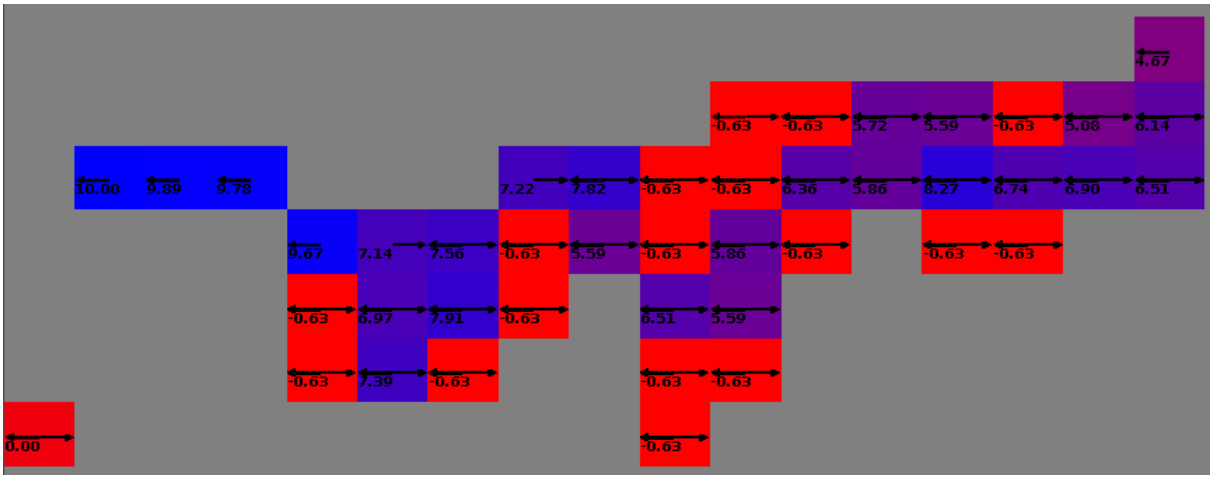

Figure 6: Block Dude Value Iteration Large Variant

As shown in the figures for the different Block Dude runs for Value Iteration/Policy Iteration, values very rarely varied and the resulting policy was the same in all scenarios using the same convergence criteria, assuming no iteration limit other than the convergence criteria.

For the Reinforcement learning algorithm, as mentioned above, Q-Learning was used with a Decaying Greedy Epsilon policy, this ensured good exploration in the initial episodes and a focus on greedy on the later episodes to get consistent results.

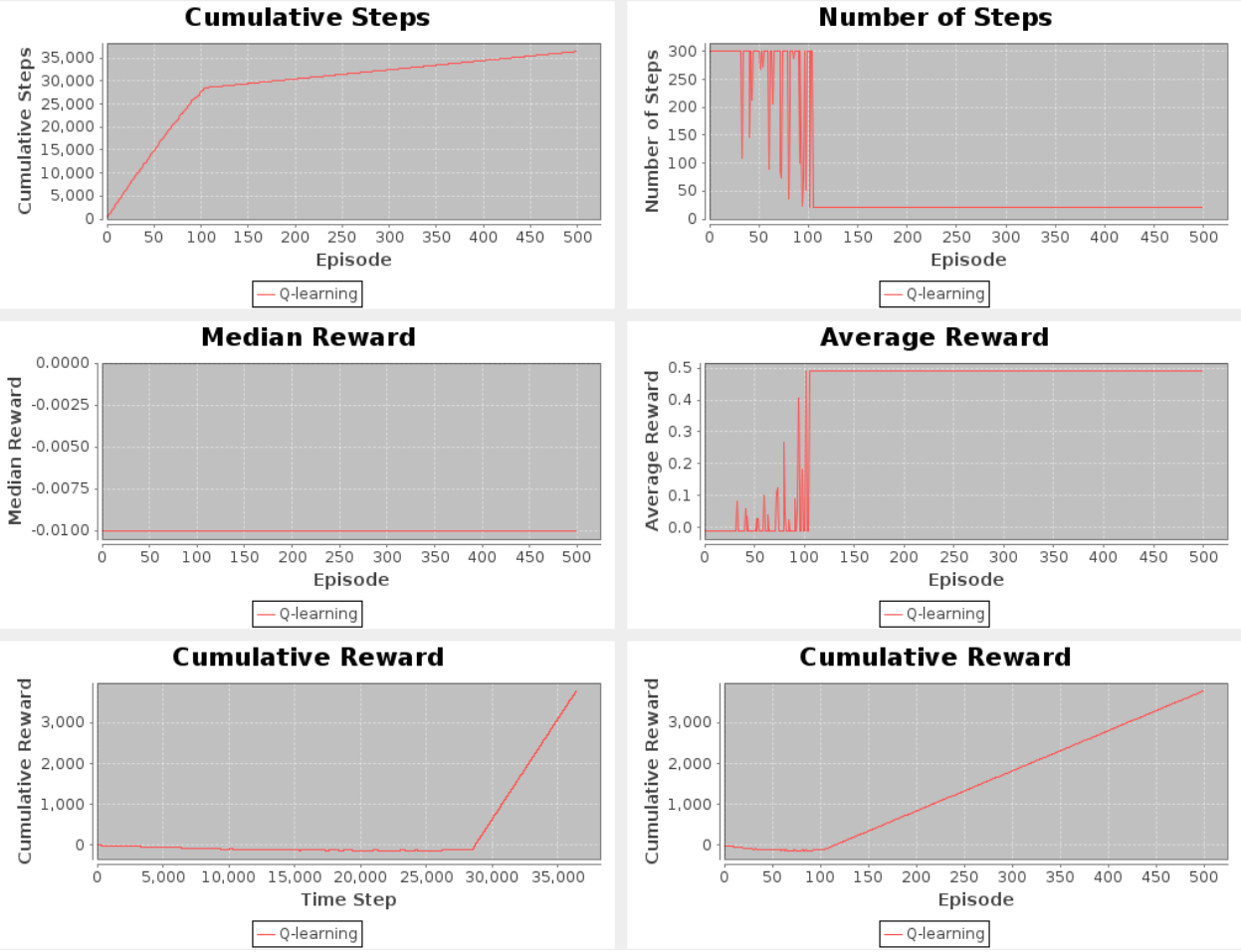

Figure 7: Block Dude Q-Learning Result for Small variant, 500 episodes

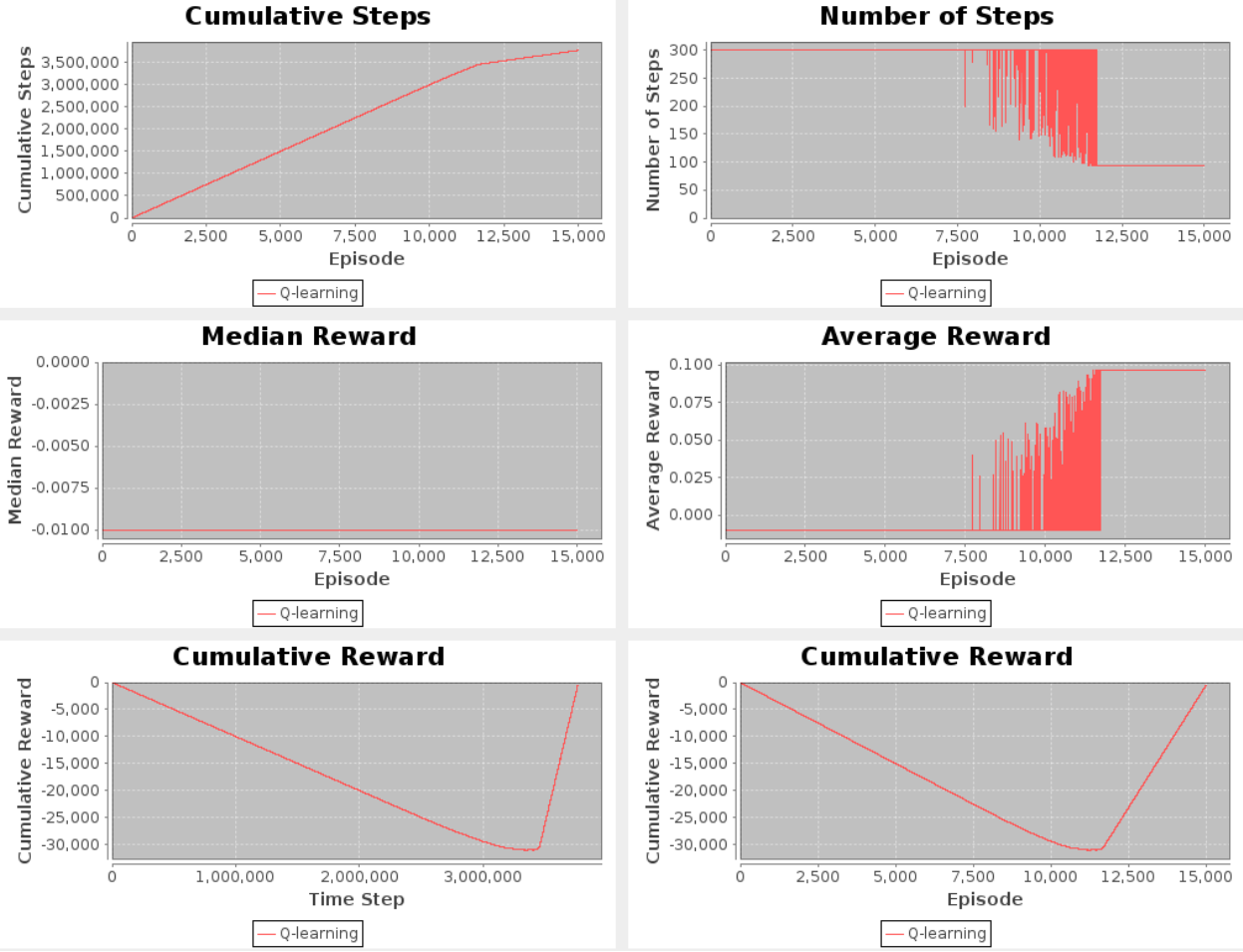

Figure 8: Block Dude Q-Learning Result for Large variant, 15000 episodes

In Figures 7 & 8, it is shown how the Q-Learner learns by building its Q-table, in the initial states, the learner explores the state space, in both cases, there is a threshold episode where the learner solves the problem, where the average reward jumps, from this point on the learner will explore that path, still with some exploration along the way to ensure if a better path exists, it will be found. In the small variant, the learner solves the problem around episode 25, and gains a steady state by episode 100, where it consistently solves the problem and achieves it in less than 25 steps. In the large variant, the learner solves the problem around episode 7,500 and reaches a steady state around episode 11,500. In both cases the jagged lines both in the Number of Steps &

Average Reward show us that the learner is still finding better paths that minimize its reward and number of steps to complete before reaching an optimal solution.

The second problem that this analysis will look at is the Maze problem, this problem is to showcase how the different methods discussed in this analysis can be used to solve problems like a maze. In the following figure, we can observe the Value Iteration & Policy Iteration runs for the maze problem, both yielding the same policy, but same as the Block Dude problem, the policy iteration too longer to complete and still yields the same results.

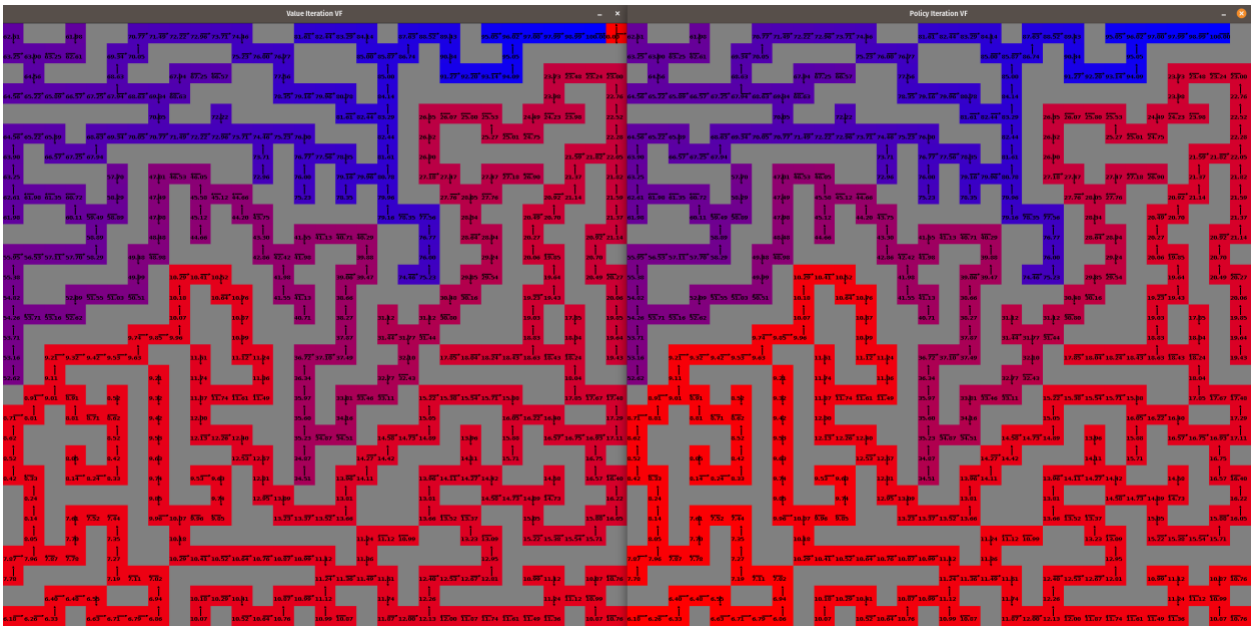

Figure 9: Value Iteration vs Policy Iteration

Value iteration completed in 129 iterations, versus Policy Iteration which took 149 iterations to complete.

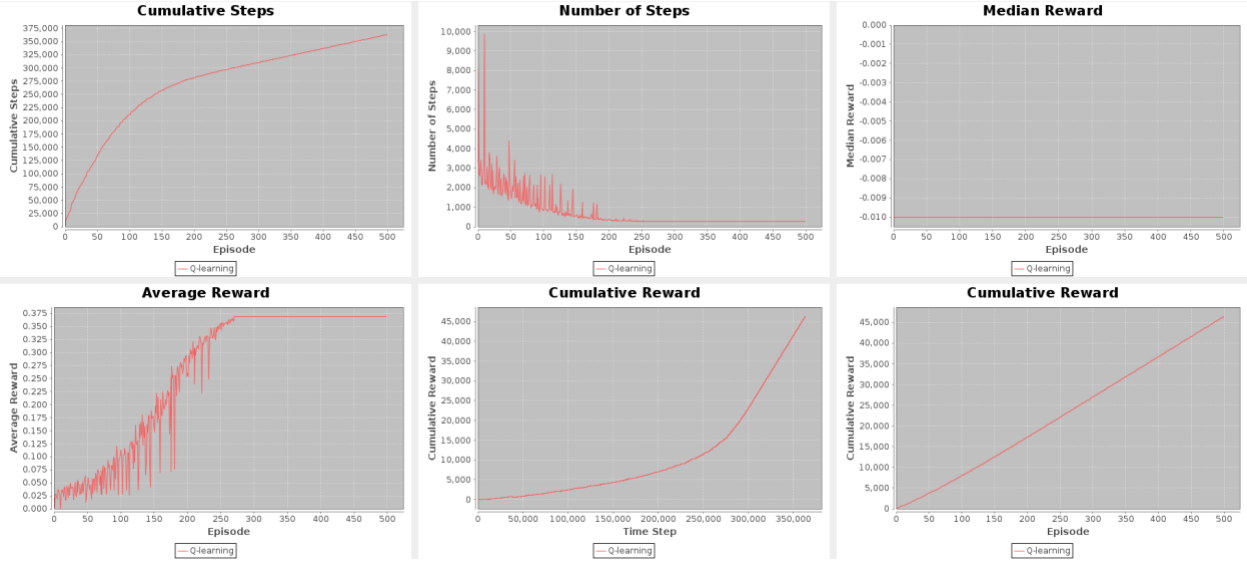

Figure 10: Maze Q-Learner results

In the Q-Learner results observed above, the learner achieves steady state in about 200 episodes, yielding a steady increase of the average reward, consistent with the exploration in the selected decaying greedy policy which correctly balances exploration in initial episodes. This greedy policy as well as a very high (0.99) discount rate, let the learner appropriately learn the environment, like Value Iteration/Policy Iteration, the resulting policy found by the learner is the same, although it may be the case, depending on the parameters, the learner could get stuck in a less optimal path in the maze if not enough exploration during initial episodes is had.

In conclusion, the best strategies for both problems were achieved by having a very high discount rate for all scenarios as well as extending BURLAP's GreedyPolicy class to a DiscountedEpsilonGreedy class, if more time was had and better domain customization in BURLAP, exploring more complex routes and tile types with negative rewards would have made for good policy visualization as well as surprising results from Q-Learners, which in interesting environments usually find solutions that no one would have thought of.

Works Cited

MacGlashan, James. *BURLAP*. n.d. <http://burlap.cs.brown.edu/>.

Openheim, Jonas. *Maze Generator*. 2017. November 2021.

<https://github.com/oppenheimj/maze-generator>.