Cairo University

Faculty of Engineering

Computer Engineering Department

CMPS458 - Reinforcement Learning

Assignment 1

Team Name/Number:

Abdelrahman Tarek

Ziad Hesham

Supervisor: Ayman AboElhassan

October 22, 2025

Deliverables

Repository Link: GitHub Repository
Video Record Link: Google Drive - Videos

Experiment Summary

During this experiment we learned a lot about the policy iteration method for reinforcement learning, we got introduced to a lot of its flaws and limitations. Another side we had to face was the tuning of parameters in order to achieve acceptable agent behaviour, down below we will be noting each of the tunable parameters we found and the effects we observed on the agent's behaviour. We will also highlight the limitations mainly in form of computational expenses as well as memory constraints.

Discount Factor Observations

During all the previous testing we had the discount factor set at 0.99, our original hypothesis had it that the discount factor when increased would prefer long term goals while when decreased would prefer instant rewards, what we found during testing was that the higher the value we have the more optimal the outcome will be, while when we lower it it tends to deviate into loops as it prefers delaying any possible loss as far as possible while future rewards no longer mean much.

Table 1: *Reward Parameter Tuning and Behavioral Observations*

lightgray **Step Cost (R)**	**Bad Cell Penalty (R_X)**	**Goal Reward (R_G)**	**Key Behavioral Observations / Convergence Notes**	**Figure Ref.**
-1.0	-100.0	$+100.0$	We noticed that the agent is cautious and avoids the bad cells as much as possible since the agent prioritizes arriving at the goal safely.	Fig. 1
-1.0	-10.0	$+10.0$	When the step cost is near to the goal reward and the bad cell cost, it tries its best to rush to the goal, no matter the risk.	Fig. 2
-1.0	-500.0	$+100.0$	When the penalty for the bad cell is extremely large, the agent avoids the bad cells as much as possible even if it means taking longer routes.	Fig. 3
-50.0	-100.0	$+100.0$	Since the step cost is high, it rushes to anything to decrease the penalty it gets (goal and bad cells).	Fig. 4

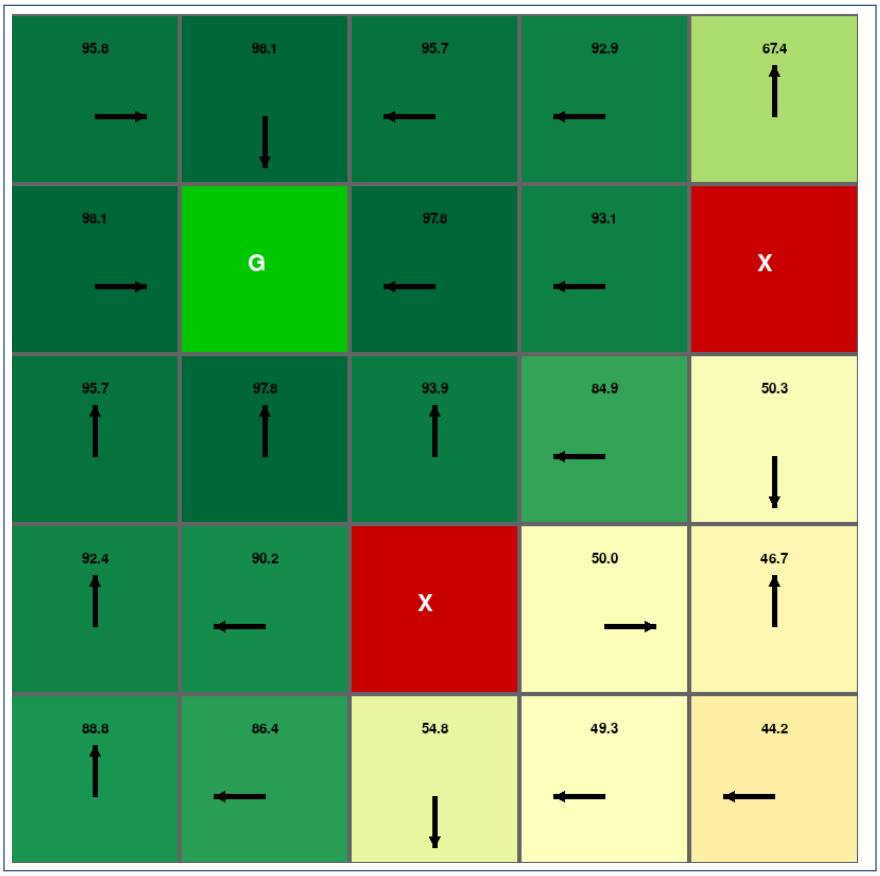

Figure 1: *Final Policy Map for Baseline Parameters (*$\mathrm{R} = -1.0, \mathrm{R}_X = -100.0, \mathrm{R}_G = +100.0$*). This corresponds to the first row of Table 1.*

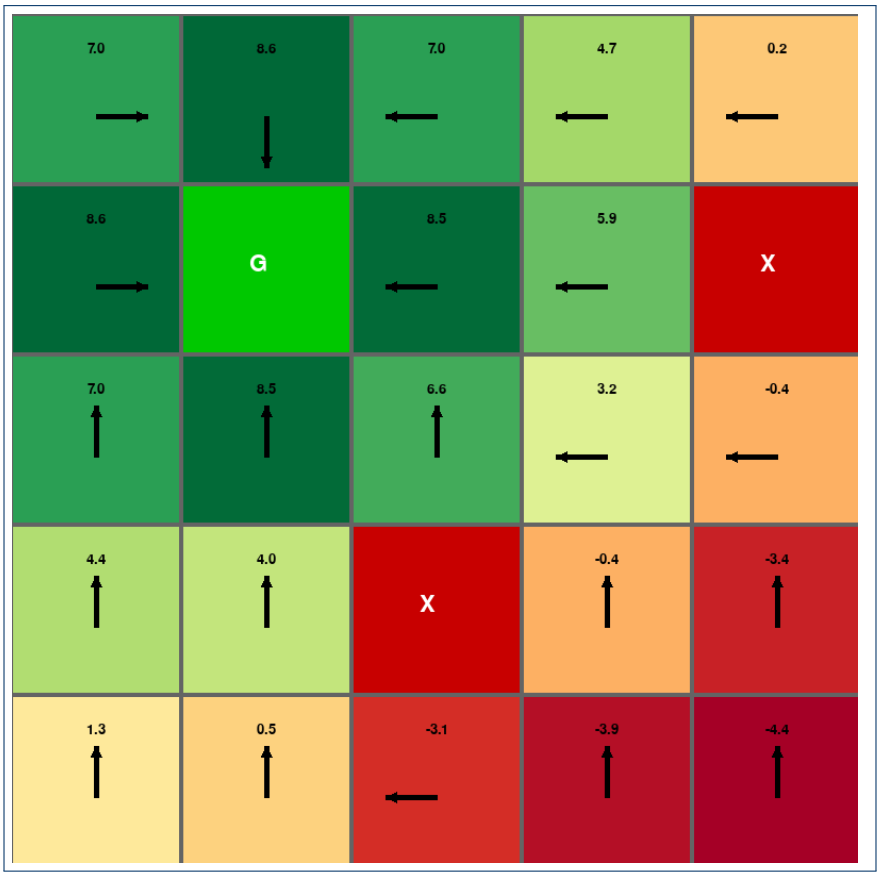

Figure 2: *Final Policy Map for Balanced Parameters* ($R = -1.0, R_X = -10.0, R_G = +10.0$).

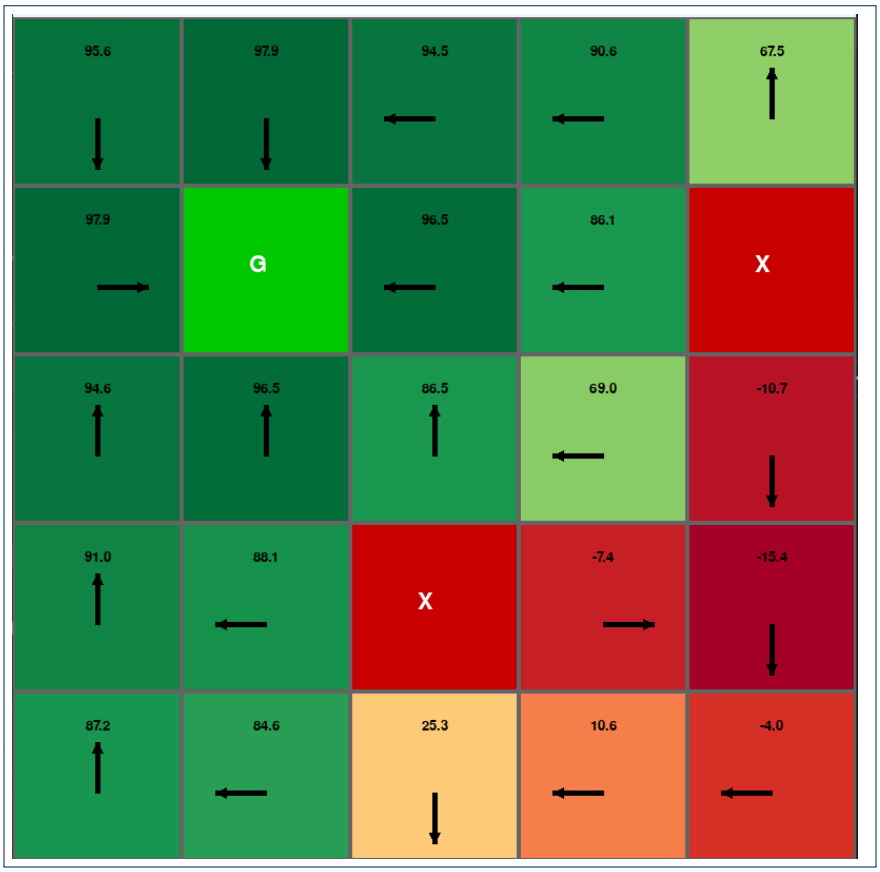

Figure 3: *Final Policy Map for High Bad Cell Penalty* ($R = -1.0, R_X = -500.0, R_G = +100.0$).

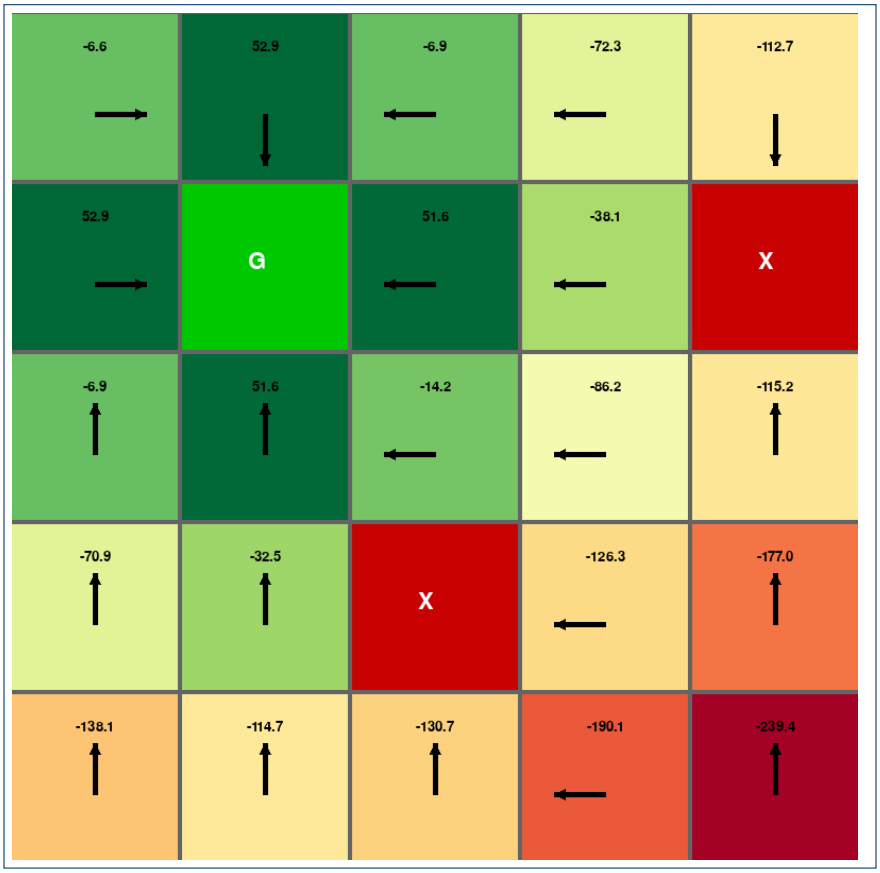

Figure 4: *Final Policy Map for High Step Cost* ($\mathrm{R} = -50.0, \mathrm{R}_X = -100.0, \mathrm{R}_G = +100.0$).

Question Answers

Question 1: What is the state-space size of the 5x5 Grid Maze problem?

It depends on the definition of the state-space used. It can be considered episodic which means that when the goal cells and the bad cells are finalized (locked in place), it would mean that we have 25 possible states only.

If we consider that we are training a more general case agent, one that can be put in any possible combination of the 5x5 grid and still have optimal policy, we would have to take into consideration the goal and bad cells positions in each state, expanding from our 5x5 $\rightarrow$ 25 state space to a much bigger one, $25 \times 24 \times \mathbf{C}_2^{23} \rightarrow 151800$. This takes into consideration 25 possible starting/current positions with 24 remaining goal positions, and 2 out of 23 remaining positions for the bad cells. This holds under 2 assumptions, that both bad cells are indistinguishable such that their order does not matter and under the assumption that there can not be any 2 cells in the same position (no starting on a bad cell or on a goal cell).

Question 2: How to optimize the policy iteration for the Grid Maze problem?

We set delta threshold during the policy evaluation phase to make it converge to a value that is close enough, this could be further optimised if we remove completely the iterative property of it and stick to one time evaluation per iteration. Another optimisation we have is that we set an iteration limit so as to not keep it running forever. While the previous decision does not ensure optimal policy reaching it ensures that we can find some policy that could be satisfactory within a realistic timeframe, this would help a lot if we increase our state space or grid size.

Question 3: How many iterations did it take to converge on a stable policy for 5x5 maze?

Due to having an initially random policy, converging speed depends heavily on our starting state, it usually ranges between 4 and 8 iterations.

Question 4: Explain, with an example, how policy iteration behaves with multiple goal cells.

In this case, it will all come down to our reward function. In our case with a reward function that penalizes extra steps taken and also has high reward for the goals, the policy usually points towards the closer goals, they are also affected heavily by surroundings since the movement model is stochastic, the agent will also pick goals with movements that have no chance of falling in a bad cell, so it will work but there will always be biases towards one goal.

Question 5: Can policy iteration work on a 10x10 maze? Explain why?

Implementation 1:

If we train the agent for the current episode, it's computationally feasible as it would only calculate for the current grid which isn't demanding at all.

Implementation 2:

If we train all the possible states beforehand, it would be very computationally hard as if we consider our initial calculation, there would be $100 \times 99 \times \mathbf{C}_2^{98}$ possible states which is equal to 47,054,700 which is very expensive compared to the 151,800 of the 5x5 grid maze.

Question 6: Can policy iteration work on a continuous-space maze? Explain why?

No, as policy iteration requires a discrete space to be able to calculate. We can approximate the continuous-space to a discrete space which would make it work.

Question 7: Can policy iteration work with moving bad cells (like Packman moving ghosts)? Explain why?

Implementation 1:

In this implementation, this would not work as the policy would continue to change and would not converge.

Implementation 2:

Since we calculated everything before hand and the agent knows where everything is, it can simply switch to the policy of the grid which has these bad cells' locations.