Final Project: Q-Learning

As a final project, I implementing reinforcement learning as an extension to PA3. More specifically, I am implementing Q-Learning, which can be used to find an optimal action-selection policy for any given Markov decision process. It uses an action-value function which gives the expected utility of taking a given action in a given state and following the optimal policy as a result. In this case, the probabilistic models used to solve the Markov Decision Process problem are not known and have not been learned.

# Implementation

To implement Q-Learning, I started with the straightforward approach and assumed that the robot only moved in the correct direction. I assumed this because the robot can only move in a single direction. If it moves in a different direction than specified, then it would give a wrong Q-value for that particular action of a single state. For each direction that the robot could head in, it would calculate the maximum Q-value of heading into that particular direction. This was calculated using the formula 

Afterward, I multiplied that with alpha and added it with (1-)\*. This gives the correct Q-value for heading in that direction. I would then replace the current Q-value with the one I just calculated and move the robot in the direction it stated it was going in.

After implementing this functionality, I also added the rest of the movements (Backward, Right, Left) with their respective probabilities in a separate q-learning file so that I could account for those moves if the robot decided to move in that direction. In this scenario, I am assuming that the robot has a scanner and can scan neighboring nodes after deciding on a move.

# Observations and Analysis

## Wall Rewards

Having 0 Reward for bumping into the walls can lead to an extremely slow convergence because if our exploration rate isn’t set too high, we will always prefer bumping into the wall over a negative-rewarded step. The solution to this would be to either give a negative reward to bumping into walls, or give a high exploration rate. However, the latter means that you are leaving it up to chance for which direction you want the robot to go in. In that case, it will be more likely for the robot to disregard the policy you’ve calculated.

By giving a negative reward for hitting walls, we also come to the possibility that the convergence will not look like what value iteration looks like if walls have a 0 reward. We can see this because the output of PA3 uses 0 wall reward and the optimal policy uses this to an advantage by telling the robot that walls are helpful. If we did this in Q-Learning, we would be ‘head-butting’ the wall continuously because our policy says that this is the correct way to go. There is also a chance we’ll explore other nodes, but that needs to have a higher probability.

## Robot Uncertainty

By adding uncertainty to the robot in which we give the probabilities of it going in other directions than the one specified, the outcome looks more similar to that of what we did in PA3 for value iteration. However, by giving a negative wall reward, it will definitely not be the exact same output.

## Comparison to Value Iteration and Policy Iteration

Q-Learning is a great way to represent reinforcement learning because we walk a robot through an unknown map, and as it gathers data, it can slowly start charting out the optimal policy. However, compared to value iteration and policy iteration, Q-Learning converges at a much slower rate. This is obvious since we do not have as many resources available to us as in value iteration and policy iteration.

# Links

## Repository

https://github.com/chewmeister/CSE190-QLearning

## YouTube video link

https://www.youtube.com/watch?v=sYG9vSTtcRA

# Conclusion

As a conclusion, Q-Learning is a great way to represent reinforcement learning because we walk a robot through an unknown map, and as it gathers data, it can slowly start charting out the optimal policy. However, compared to value iteration and policy iteration, Q-Learning converges at a much slower rate. This is obvious since we do not have as many resources available to us as in value iteration and policy iteration.