Alex Navarro ajn853 Machine Learning Assignment 5  
  
**Reinforcement Learning with Q-Learning**

Introduction

Reinforcement learning is the algorithmic process developing an optimal solution to a problem based on some user defined reward or cost function through iterative improvements. It can generally be split up into two main categories: model free and modelled reinforcement learning. In modelled reinforcement learning, there is some prior knowledge of the dynamics of the environment which can be used to improve the learning process. However, it is not always possible to have this information ahead of time. In these cases, model free reinforcement learning is a viable solution. One method of model free RL is what is known as Quality-Learning or Q-Learning.

The Q-Learning algorithm learns about the environment in real time using trial and error. The environment is broken up into finite, distinct states. For each of these states, there is a finite, discrete number of actions that can be taken. For each state, the q-value of an action is the representation of the estimated reward of taking that action at that particular state. Higher q-values indicate a more favorable outcome, while lower q-values indicate a diminished outcome. Each state also comes with an immediate reward, which is used in the learning process from updating the q-values. The equation is as follows:

Here is is the learning rate, which determine how much each q-value is updated at each step. is the reward at the state s with action a, and the term is the discount factor. The discount factor is used to weight future rewards vs immediate rewards. A high discount factor will mean future rewards are weighted heavily, which a low one will favour immediate rewards. The action that is chosen is randomly chosen at each step, and actions with higher q-values are favored over lower valued actions.

In scenarios where a complex action or reward structure is needed, it can be difficult to properly teach a q-learning model directly. This can be solved by the introduction of modules. A module in the Q-Learning context refers to a separate state-action q-value table for different rewards and actions. For instance, one could have a module for staying close to a desired goal point, and a second module for avoiding obstacles along the way. Then, when combining the modules in the final q-learning runtime, they can be weighted appropriately to induce different behaviors in the RL agent.

For this assignment, the goal is to have a sidewalk cleaning RL agent navigate a grid containing both litter (a reward) and obstacles (a penalty). The agent must move from one end of the grid to the other as quickly as possible while picking up as much litter as possible but while simultaneously avoiding the obstacles.

Method

The solution for our sidewalk cleaning agent will contain four modules. These will include: encouraging the agent to move **forward**, discouraging the agent from going off the **sides** of the sidewalk, encouraging the agent to pick up **litter**, and discouraging the agent from colliding with **obstacles**. The grid will be arranged vertically (i.e. the agent needs to move from the bottom to the top) and the objects in the grid will be randomly arranged. Once such layout is shown below

Chart

Description automatically generated with medium confidence

Once the agent reaches the final row of the sidewalk, or if it goes off the sides or bottom of the sidewalk, the episode is considered terminated. For all modules, the learning rate and the discount factor .

**Forward Module**

The simplest of the modules is the one that encourages the agent to move forward. In this case, that state of the agent is simply its vertical location in the grid, and the reward is defined as follows:

In this way, the module encourages moving forward, discourages moving backwards, and applies a neutral approach to moving laterally. Running this module through 1000 iterations with a random sidewalk gives the following result

Chart

Description automatically generated

Note that the blue line is the path of the agent. The agent performed as expected and chose the UP action as the ideal action in all scenarios.

**Middle Module**

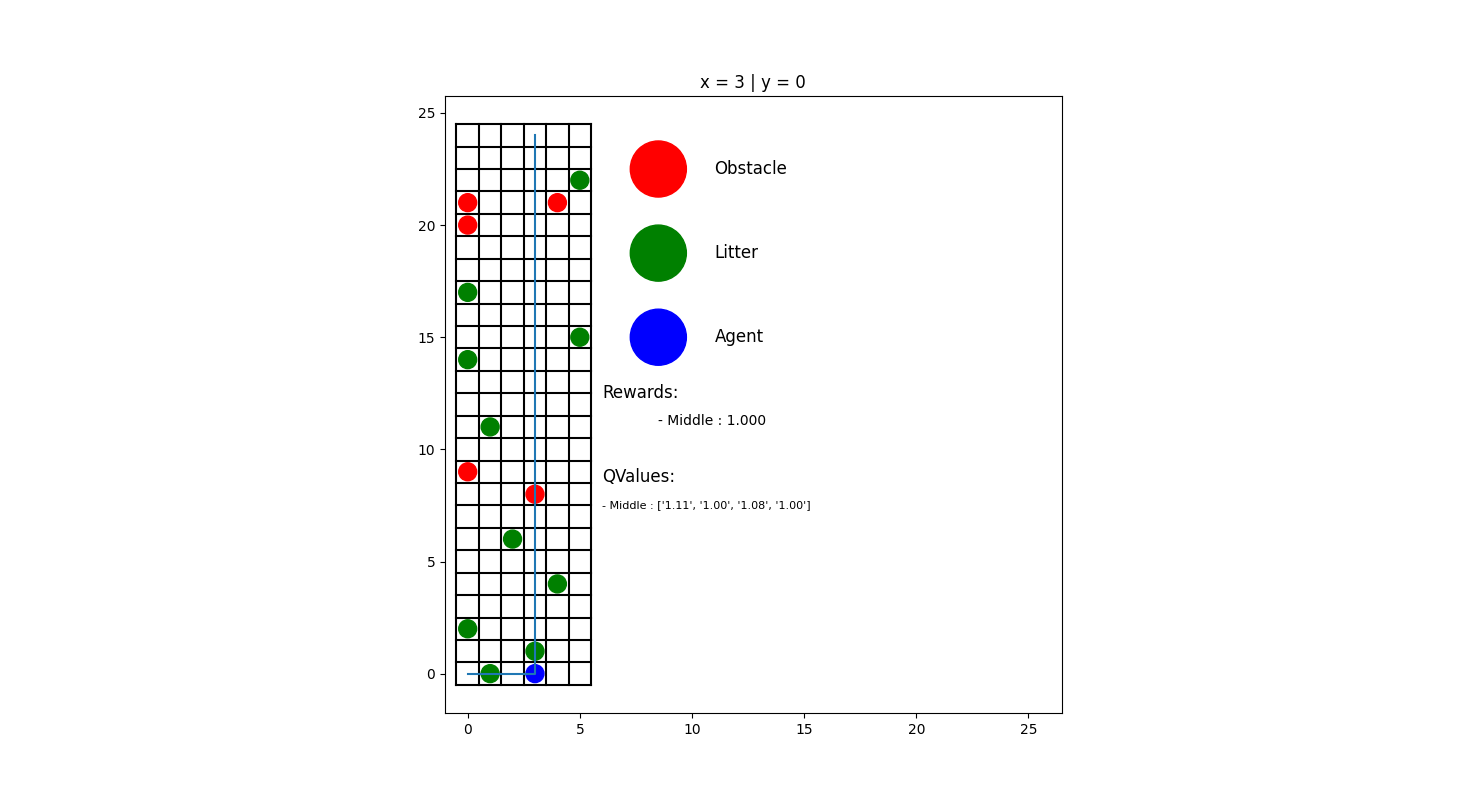
This module is designed to keep the agent from going off of the sides of the sidewalk. As such the only information it needs for its state is the horizontal location of the agent. At state 0, it is at the left of the sidewalk and at state 5 it is at the right. The reward function for this is then:

This reward function is defined as a quadratic equation where and . We can immediately see the correct effect coming from this module by looking at the q-table:

A screenshot of a computer

Description automatically generated with medium confidence

When the agent is to the left, the correct action is RIGHT. When the agent is to the right, the correct action is LEFT. And when the agent is in the center, the correct action is UP. This can also be seen in the test run at the end of training.



**Litter Module**

The litter module is where things get a little more complex. Since we want the agent to be able to deal with litter before it’s picked up, but not after being picked up, the state for this module has to be recalculated every step with the litter position information. The state also cannot relate to the absolute position of the litter, since the litter will be removed, and the positions will change. Instead, the state of the litter module will be related to the relative position of the *nearest* litter object to the agent at the time the action is selected. If two pieces of litter are at the same distance, then the lowest (x,y) pair will be considered. The state function in this case can be laid out explicitly as

where and . This formulation allows for the 0th state to be at and the final state to be at .

The reward function is then based on whether or not the agent is currently sharing a cell with a litter object.

Running this module through 1000 training episode gives the following result:

Chart

Description automatically generated

This module performs extremely well, passing through every piece of litter in the setup. Of course, it still passes through obstacles, but that is because we have not added any way for the agent to detect or act against obstacles yet, so that is expected.

Once downside to this formulation is that there are a lot of states compared to other modules. Considering the full range of motion of the agent, there are over 500 states for the agent relative to the nearest litter. A possible optimization would be to limit the “field of view” of the agent to within a few rows of its current position, reducing the states and ignoring litter that is very far away.

**Obstacle Module**

The obstacle module can be addressed in one of two ways. The first is to treat it similarly to the litter module, in that the state is relative to the nearest obstacle. The other, simpler way is to treat the obstacle locations as absolute, and simply have a q-cell for each cell of the grid. This is the solution implemented for this assignment. The downside to this is that the learning only applies to the presented configuration of obstacles; if the configuration changes then the q-table is invalid. The reward function is extremely simple

Despite the simplicity of the module, it works excellently for avoiding obstacles, as seen in the test run below

Chart

Description automatically generated

**Combined Q-Learning Modules**

Now that all of the modules are working independently, the only thing left to do is mix them together. At each step, the q-values of each module are layered according to the weights that we assign them. After some trial and error, the following set of weights was decided upon.

|  |  |
| --- | --- |
| **Module** | **Weight** |
| Forward | 0.05 |
| Middle | 0.30 |
| Litter | 0.60 |
| Obstacles | 1.00 |

Note that these weights also take some part in normalizing the rewards for each of the modules, so they do not directly indicate the relative effects of each module on the agent. A future improvement would be to incorporate the normalization into the reward functions so that the weights chosen would be more descriptive and representative of their effects.

Below are plots of the combined RL agent performing a test episode after 1, 10, 100, and 1000 training episodes. An episode is considered complete when the agent leaves the sidewalk.

Chart

Description automatically generated with low confidenceA picture containing chart

Description automatically generated  
 1 Iteration 10 Iterations

Chart

Description automatically generated with low confidenceChart

Description automatically generated with medium confidence  
 100 Iterations 1000 Iterations

We can see that after only 10 iterations it was capable of performing extremely well given its environment, even though it did collide with an obstacle. By the time it reached 100 iterations it was already fully trained, and extending the training to 1000 iterations did not produce any new behaviors. The agent was able to fully traverse the sidewalk while picking up litter and avoiding obstacles.