Project 3

Due April 28th, 2017, 11:59PM

1 Introduction

In this project, we will help Wheelbot determine an optimal *policy* that will enable it to gather an optimal amount of reward *reward* (in expectation) in a stochastic but fully-observable environment.

In Project 2, Wheelbot moved deterministically. Telling Wheelbot to move a certain direction resulted, with probability 1, in Wheelbot moving the correct direction. In this project, Wheelbot will move stochastically around its environment. Additionally, Wheelbot will be given a *reward* signal upon transitioning into a new state. Obstacles of known location will still be present.

2 Reinforcement Learning

We can formalize this environment as a Markov Decision Process with states S, actions A, transition probabilities T and rewards R. The reward function, $R(s)$ will give Wheelbot a reward signal for being in state $s \in S$. There will be a transition probability $T^a_{s,s'}$ for each possible combination of $a \in A$ and $s, s' \in S$.

The states of the environment are the grid cells. The environment is fully observable because Wheelbot always knows which grid cell it is in. Wheelbot's transition model is as follows: Wheelbot makes the "correct" transition with probability $0 \leq p_c \leq 1$ for any action a. The remaining probability mass is divided equally amongst incorrect transitions to neighboring grid cells. For example, if Wheelbot is told to go Up and $p_c = 0.85$, it will transition Up with probability 0.85. With probability 0.05 it will go Right. With probability 0.05 it will go Left. With probability 0.05 it will go Down. Note that care must be taken when considering the edges of the grid. When Wheelbot

tries to transition into a wall, it will stay where it is. Along the edges of the environment and in the corners, self-transitions will have non-zero probability, and the transition function will be different than it is in the rest of the grid. **To simplify calculations, we will not allow diagonal transitions in this project. Only the actions $\mathbf{A} = \{\mathbf{Up}, \mathbf{Down}, \mathbf{Left}, \mathbf{Right}\}$ are valid.**

3 Project Details

Your task in this project is as follows: given a list of known obstacle locations, a goal location, and p_c, solve this MDP using the value iteration algorithm to generate an optimal policy $\pi(s) \rightarrow a$ that maps states to optimal actions so as to maximize Wheelbot's expected utility. Wheelbot's expected utility will be defined in terms of a reward function $R(s)$ that you define. This reward function should be generalizable to any set of obstacle locations and goal location. You will also need to define the specifics of $T^a_{s,s'}$ given the definition of Wheelbot's transition model above.

Once you have generated an optimal policy, Wheelbot should follow that policy to (hopefully) make it to the goal. Note that, due to the stochastic nature of the environment, it is possible for Wheelbot to run into an obstacle, even when following an optimal policy. Therefore, it is important to test your program with a number of values of p_c, including $p_c = 1$, to ensure that this does not happen when the environment becomes deterministic again. Another possible method of testing is to set p_c to several different (increasingly large) values and try your program a large number of times for each such value, keeping track of the number of times Wheelbot runs into an obstacle. The environment should be kept the same through all runs. In general, the more deterministic the environment becomes, the less often Wheelbot should run into an obstacle. The environments that we use to test your program will be solvable in the sense that there will be at least one possible path to the goal state from the start state.

Wheelbot will have the same sensors as in Project 2, with the exception that the local obstacle sensor will not be used in this project (as there are not hidden obstacles).

4 Implementation Details

This project will require you to modify the files Project3.h and Project3.cpp files in the project source code. You are not to modify any other file that

is part of the simulator. You can, however, add new files to the project to implement new classes as you see fit. We will test your project by copying your Project3.h and Project3.cpp files, as well as any files you have added to the project, into our simulation environment and running it.

Feel free to use the C++ STL and STD library data structures and containers. Additionally, if you have previously implemented data structures you wish to reuse, please do. However, you must not use anything that trivializes the problem. For instance, do not use a downloaded value iteration algorithm package. You must implement value iteration yourself.

For full credit your $R(s)$ must be valid (capable of solving the problem), and your program must compute and follow the optimal policy relative to this reward function. Importantly, if $p_c = 1$, your robot should follow a shortest path (there may be multiple such paths) to the goal. *pc* may be adjusted in main.cpp during your testing. Set the discount factor $\gamma = 0.9$ for all testing.

5 Submission

Please submit your modified Project3.h and Project3.cpp, as well as any additional files you added to complete this project, to Blackboard by April 28th, 2017, 11:59PM.