CS362 (Spring 2017) Machine Intelligence

Assignment 4 (Part 2) – Expectiminimax & Approximate Q-Learning

Due date: 19/04/2017 (23:59)

This part of the assignment has a weight of 5% on your final grade.

In this part of the assignment you will use the same environment as in Part 1. You may now assume that there exists an adversarial agent in this environment as shown in Figure 4.1 (indicated by a red circle). Assume that your agent (which we will call the green agent) always starts in the position (6, 1), and that the adversarial agent (which we will call the red agent) always starts in the position (1, 1).

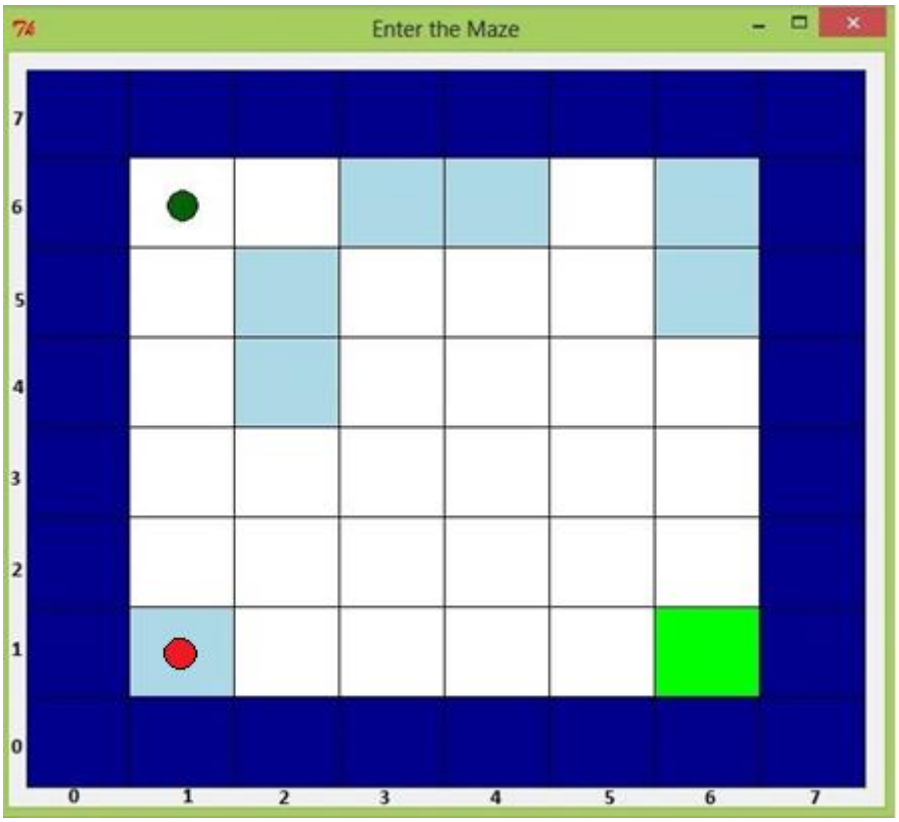

Figure 4.1 – Game playing grid configuration (see part1 of the assignment for details on how to represent this environment)

Remember when the green agent attempts to move in a particular direction, it moves in its intended direction with a probability of 70%, otherwise it moves in the directions perpendicular to it with equal probability. However when the red agent attempts to move in a particular direction, it moves in its intended direction with 100% probability.

Gameplay: The first player to move will be the green agent, then after that agents alternate taking turns. A running score is kept, and is updated under the following conditions (Note that this is the criteria which you will use to evaluate the leaf nodes in your game tree):

- If the green agent lands on the green block, the score will be +30, and the game ends.
- If the green agent lands on a dark blue block, the score will be -10, and the game ends.
- If the green agent lands on a light blue block, the score will be -5, and the game ends.
- If the red agent lands on a block that the green agent is currently occupying, the score is -50, and the game ends.

Note that terminal states only apply to the green agent, the red agent is free to move in any state.

For this problem, a single search ply is considered to be the movement of the green agent, and the red agent's response to it. So depth 2 search would involve the green and the red agent each moving 2 times.

(a) For this section, assume non-terminal states (non-leaf nodes) have a value of 0. The function *emm_no_pruning* takes as input a grid configuration like the one shown in Figure 4.1, and a depth limit. This function should return the value of the topmost max node, after constructing a game tree of the given depth. **– 30 points**

(b) For this section, also assume that non-terminal states (non-leaf nodes) have a value of 0. The function *emm_ab_pruning* takes as input a grid configuration and a depth limit. This function should return the total number of nodes (only count the max and the min nodes, **not** the chance nodes) that are pruned after applying the alpha-beta pruning algorithm on your game tree. ***You must not prune on equality***. – **30 points**

(c) For this section, you will design your own evaluation function using approximate Q-Learning. The evaluation function will have two components. The first feature f1 is going to be the Manhattan distance of the agent to the goal state and the second feature f2 will be the Manhattan distance of the agent to the adversary. In other words, we will express Q(s, a) as follows:

$Q(s, a) = w_1 * f_1(s, a) + w_2 * f_2(s, a)$ where w1 and w2 are the weights that you will compute by using gradient descent as discussed in class. The function *approximate_q_learning* takes as input a grid configuration and should return w_1, w_2. You may assume that the initial values for the weights to be $w_1 = 1$ and $w_2 = 1$. Use a learning rate of 0.1. – **40 points**

You may write additional functions of your own, but make sure that the names of the functions that we have provided you remain unchanged. Prepare and upload one python script which contains these functions and name this script as *<your first name>_<your last name>_ assignment4_part2.py*