CS362 (Spring 2017) Machine Intelligence

Assignment 4 (Part 1) – Markov Decision Processes

Due date: 12/04/2017 (23:59)

This assignment consists of 2 parts (the next part is to be assigned next week). This part of the assignment has a weight of 5% on your final grade.

In this assignment you are going to help an agent navigate optimally through a maze by making use of MDPs. An example setup for your environment is shown below:

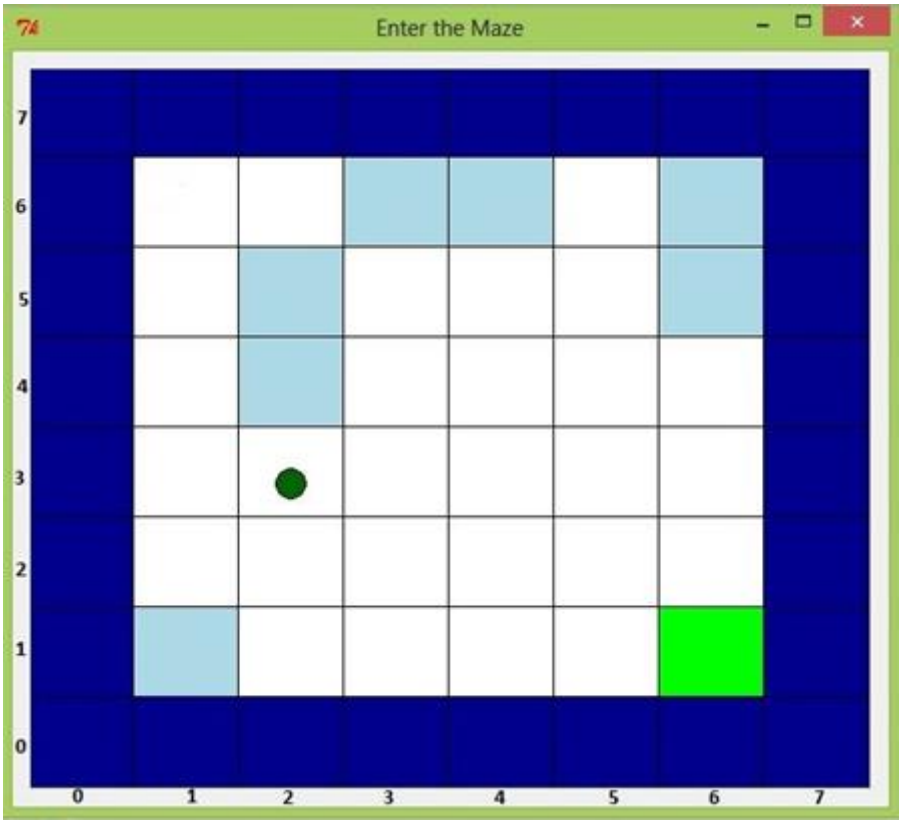

Figure 2.1 Representation of an agent navigating through a maze

In the Figure above, the coloured cells are terminal states. When the agent reaches those cells, it collects a reward and exits the game. The goal of the agent is to maximize the sum of its rewards.

- The dark blue regions have a reward of -10
- The light blue regions have a reward of -5
- The green region has a reward of +30
- The green circle denotes where the agent is in the grid
- Each move from one white block to the next has a reward of -1

- When the 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.

Do not assume that the location of any of the coloured blocks is fixed. We will test your program with different block configurations.

You may assume that the grid is (8 x 8). **Please follow the indexing scheme shown in Figure 2.1**. You should represent the grid by a 2D array. The array has values 0 – white block, 1 – light blue block, 2 – dark blue block and 3 – green block. For example, the grid shown above is represented by the following array:

```
[[2, 2, 2, 2, 2, 2, 2, 2],
 [2, 0, 0, 1, 1, 0, 1, 2],
 [2, 0, 1, 0, 0, 0, 1, 2],
 [2, 0, 1, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 1, 0, 0, 0, 0, 3, 2],
 [2, 2, 2, 2, 2, 2, 2, 2]]
In the array above the highlighted index is not the state (1,1) but the state
(6,1). See Figure 2.1 for clarification.
```

You should represent each state in the configuration by a coordinate tuple e.g. (1, 1). You may represent the actions as follows:

0 – North

1 – East

2 – South

3 – West

4 – Exit

Where each action denotes the direction that the agent ought to move in.

(a) The function ***compute_v_values*** takes as input a grid configuration like Figure 2.1 above, and should return for every state the V-values for that state after performing value iteration (Note that you should also compute the Q-values for every state to achieve this goal). The return type should be an array of tuples where each tuple has 2 elements. The first element should be a state, and the second element should be the V-value for that state, for instance

```
[((0,0), -10), ((0,1), -10),... ((4,5), 3.2), ...]
```

The array above shows that cell (0, 0) has a V-value of -10 and cell (4, 5) has a V-value of 3.2 for an arbitrary grid configuration. – **70 points**

(b) For this section you will return the optimal policy for a given grid configuration. The function ***get_optimal_policy*** takes as input a grid configuration and should return for every state, the optimal policy (direction to move) for that state. The return type should be an

array of tuples where each tuple has 2 elements. The first element should be a state, and the second element should be the optimal action for that state, for instance

```
[((0,0), 4), ((0,1), 4),... ((4,5), 2), ...]
```

The array above shows that optimal policy in cell (0, 0) is to take the exit action and the optimal policy in cell (4, 5) is to move south. – **30 points**

We also provided you with a graphical user interface to visualise the actions taken by your agent, given a policy. Details about how to use this interface are given in the script ***visualise.py***.

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 two functions and name this script as *<your first name>_<your last name>_ assignment4_part1.py*

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).

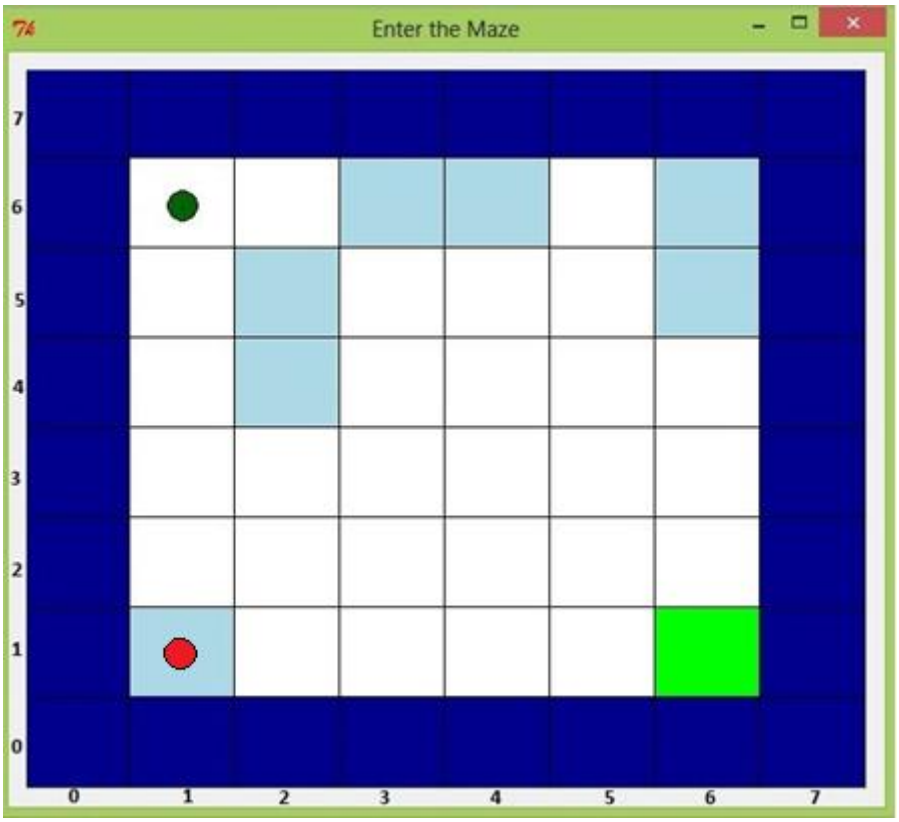

Figure 4.1 – Game playing grid configuration (see part1 of the assignment for details on how to represent this environment)

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*