CS362 (Spring 2017) Machine Intelligence

Assignment 2 – Informed Search (Block World)

Due date: 08/03/2017

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

The block world is a sliding puzzle that consists of an area that is divided into a grid of size 4x4 (see Figure 2.1 below). Different coloured blocks occupy different proportions of the grid.

- Each blue block occupies a region of size 1x1 in the environment.
- Each green block occupies a region of size 1x2 or 2x1 in the environment.
- Each red block occupies a region of size 2x2 in the environment.
- There are 2 white blocks (each of size 1x1) representing blank space in the environment.

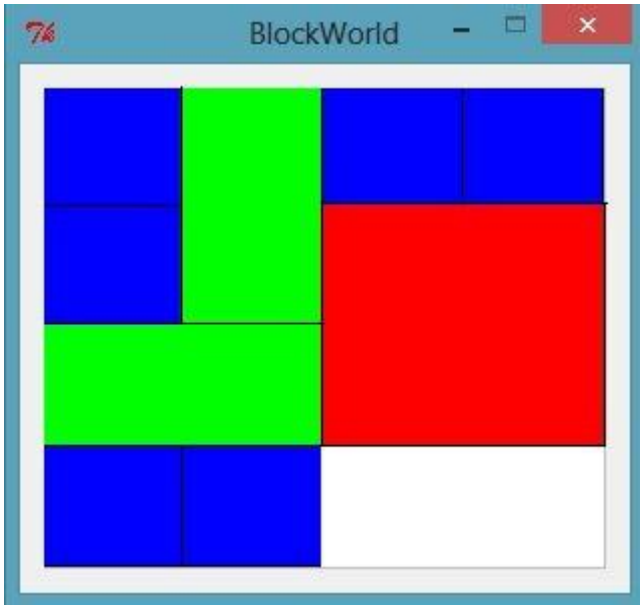

Figure 2.1 – Initial state of block world environment.

The configuration shown in Figure 2.1 is represented by the following 2D array:

[[1,2,1,1], [1,2,4,4], [2,2,4,4], [1,1,0,0]] where 0 represents a blank space, 1 represents a blue block, 2 represents a green block, and 4 represents a red block.

Each block in the block world can slide into an empty space adjacent to it, as long as it can fit into that space. Figure 2.2 illustrates the successor states that result from the initial configuration.

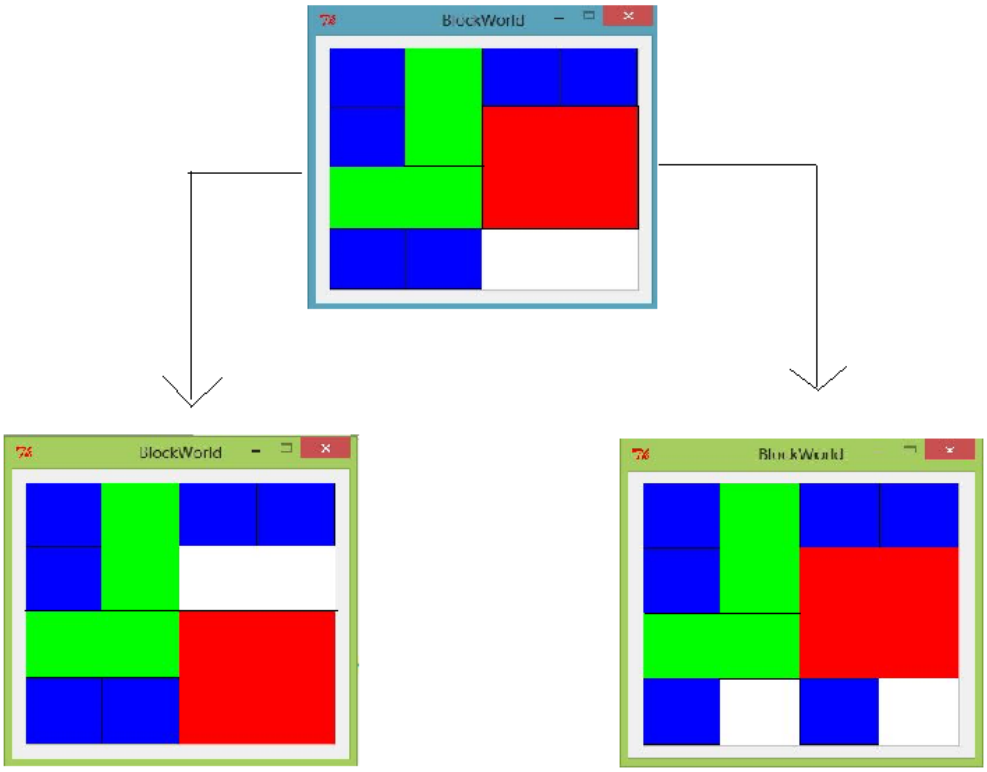

Figure 2.2 – Successor states resulting from the initial block world configuration.

Assume that the red block is trying to find its way out of the environment, and the only door is at the bottom left corner. The goal test for this puzzle is therefore to find a configuration that has the red block at the bottom left corner. An example of such a goal state is given in Figure 2.3.

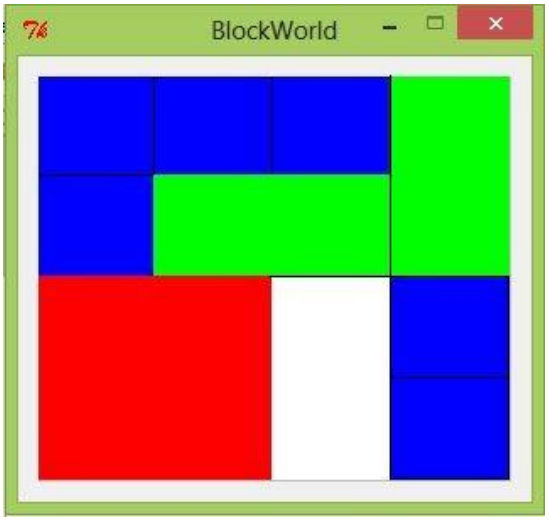

Figure 2.3 – An example goal state for the block world puzzle.

You may assume that each move by any block into an empty space has a cost of 1. We have provided you with a couple of skeleton functions to use in your implementation.

a) The function *get_successors* takes as input a 2D array representing a state and should return an array of all the successor states (as a python list). Therefore, if the input to this array is [[1,2,1,1], [1,2,4,4], [2,2,4,4], [1,1,0,0]], the output of this function should be,

[[[1,2,1,1], [1,2,0,0], [2,2,4,4], [1,1,4,4]], [[1,2,1,1], [1,2,4,4], [2,2,4,4], [1,0,1,0]]]

Note that the return type here should be a list of 2D arrays (Do not flatten the 2D arrays into 1D, if you do, you will lose 25% of the points for this section). – **40 points**

(b) You will solve this part of the problem using **uniform cost search**. The function *uniform_cost_search* takes as input a 2D array representing an initial state and should return a sequence of states (as a python list) to get us from that initial state to a goal state. This list should include the initial state (as the first element) and the goal state (as the last element). **– 20 points**

(c) For this section you will implement a heuristic function that you will use to implement the **A* search** algorithm. The function *a_star_heuristic* takes as input a 2D array representing a state and should return the Euclidean distance between the position of the red block in the current state and the position of this block if it were in the goal state. Assume that the position of the red block is represented as shown in Figure 2.4 below:

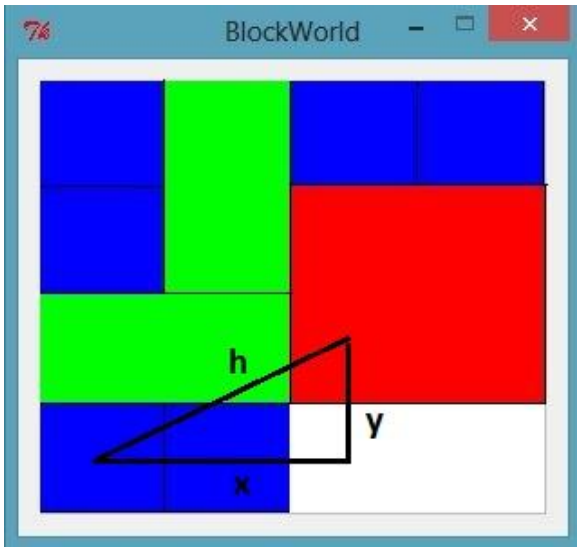

Figure 2.4 – Euclidean distance between lower left sub-section of the red block and the position of this sub-section if it were in the goal state.

In Figure 2.4, y = 1, x = 2, and h = $\sqrt{x^2 + y^2}$ = 2.236. Therefore if your function receives as input the configuration in figure 2.4, it should return the Euclidean distance (which is equal to 2.236 in this particular instance). – **20 points**

(d) Using the heuristic function that you defined in part (c), you will now implement the A* algorithm. Remember that each move by any block to an empty space has a cost of 1. The function *a_star_search* takes as input a 2D array representing an initial state and should return a sequence of states (as a python list) to get us from that initial state to a goal state. This list should include the initial state (as the first element) and the goal state (as the last element). – **20 points**

Note: For sections (b) and (d) we have provided you with an interface to visualise your output, and you may use it to make sure that the results you get are correct. The script *visualise.py* illustrates how you can make use of this interface. Just feed the output you get in sections (b) or (d) into the *start_simulation* function and make sure it is working as expected.

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 your python script which contains all the specified functions and name this script as *<your first name>_<your last name>_ assignment2.py*

Send email to both the instructor and the TA if you decide to return your assignment late