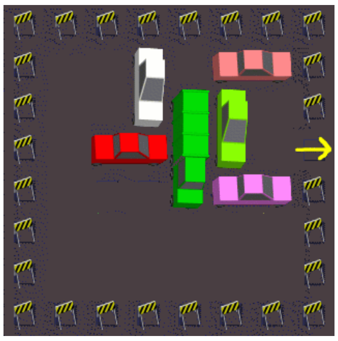
CS 180: Artificial Intelligence
Machine Problem 1: Rush Hour Puzzle (Individual work OR by pairs)
Date Due: Tuesday, March 29, 2016

Problem Description

Rush Hour Puzzle belongs to the family of sliding block puzzles usually used as a toy problem in Artificial Intelligence. As an example in the given image below, the red car is stuck in traffic and tries to escape by going to the exit (indicated by the arrow). The cars can be moved up and down or left and right. The goal is to clear the path so that the red car is able to escape past the arrow.



You are to solve this puzzle using A* search with no repeated states in the fewest possible moves. You should implment A* search in C programming language using the following heuristic functions:

1. The trivial *zero heuristic* whose value is equal to zero in all states which is equivalent to breadth-first search (BFS).
2. The *blocking heuristic* which is equal to zero at any goal state, and is equal to one plus the number of cars blocking the path to the exit in all other states. For instance, in the state above, there are two cars (namely, the two green ones) on the path between the red car and the exit. Therefore, in this state, the blocking heuristic would be equal to three.
3. A third, *advanced heuristic* of your own choosing and invention. Your heuristic should be consistent and you should aim for a heuristic that will be at least as effective as the blocking heuristic.

In your written report, you should include a clear and precise description of the advanced heuristic that you chose to implement. You also should include a brief but convincing argument of why both the blocking heuristic and your advanced heuristic are consistent, and therefore appropriate for use with A* graph search.
Note that every car is constrained to only move horizontally or vertically. Therefore, each car has one dimension along which it is fixed, and another dimension along which it can be moved.

Locations on the grid of a Rush Hour puzzle are identified by their (*x*, *y*) coordinates, where the upper left corner is square (0,0). The X coordinate is the row (horizontal) ; Y coordinate is the column (vertical). The top-left square is with coordinate (0,0). So the board cells are indexed thus :

0,0	1,0	2,0	3,0	…
0,1	1,1	2,1	3,1	…
0,2	1,2	2,2	3,2	…
0,3	1,3	2,3	3,3	…
…	…	…	…	…

For instance, in the puzzle above, the red car occupies squares (1,2) and (2,2). The goal is to move the red car so that it occupies squares (5,2) and (6,2).
Puzzles should be read from a file. Such puzzles should be encoded as in the following example representing the puzzle above:

```
6
1 2 h 2
2 0 v 2
4 0 h 2
3 1 v 3
4 1 v 2
4 3 h 2
```

The first line, "6", gives the size of the grid, i.e., this puzzle is defined on a 6x6 grid. The next line, "1 2 h 2", gives a description of the red car. Note that the goal car must always be given first. The first two numbers (1,2) give the (x, y) coordinates of the upper left corner of the car. The "h" indicates that the car is horizontally oriented ("v" would have indicated vertical orientation). The last number "2" indicates that the car has size (i.e., length) 2. The next line, "2 0 v 2" describes the white car, and so on.

Puzzles can be printed in a rudimentary ASCII form. For instance, the puzzle above would be printed as follows:

```
+-------------+
| . . ^ . < > |
| . . v ^ ^ . |
| . < > | v .
| . . . v < > |
| . . . . . . |
| . . . . . . |
+-------------+
```

The first line of the output file has the number of moves taken to get the red car out. If no solution exists, this should be -1. Each of the next lines gives individual moves for the solution. The format of the lines is:

- 1st field (Car ID) : character : We associate the first car with "A", the second car with "B", the third car with "C", etc..
- 2nd field (direction) : character : (L/R/U/D, denoting left/right/up/down)
- 3rd field : integer : number of spaces to move in the direction given by the second field.

For the example, the output would be:

```
5
D U 1
F L 4
D D 3
E D 2
A R 3
```

The program should also provide the following values as output of the program: (a) number of expanded nodes, (d) the depth of the search tree (e) the actual running time of the program (in milliseconds).

Written Report

You are expected to submit a written report which should include the following:

1. A description of your implementation of A* search using the three heuristic functions described above. You should be able to explain and justify your chosen heuristic function.
2. Describe and show the diagram/picture of the input files that you have created and the solution path generated by the three search algorithms on the said input files.
3. An analysis and comparison of using A* using three different heuristics based on relevant criteria, along with appropriate conclusions.

Deliverables

Submit the source code of your program, an executable file of your program, input files and written report via email (carlo.raquel@gmail.com). You should receive a confirmation email from me that your submission has been accepted. Otherwise, you need to contact me to confirm your submission. Your MP will be graded as follows:

Program 40%
Written Report 60%

There will be an MP demonstration for this project. The schedule will be after the deadline.