CMSC 421 Programming Assignment 1

Due Date: Thursday, October 2, 5PM

For this programming, you will write a progrm to play the game *Rushhour*. We've provided some code infrastructure in C++ and Java. Please start early. While not a large amount of code is required for the final solution, you will need time for running and debugging, so plan accordingly!

RushHour

In this game, your goal is to drive a designated vehicle (your red car) out of the traffic jam and escape to freedom. To do so, you must maneuver all the blocking cars and trucks out of your way.

A 6x6 grid describes the traffic world. All vehicles occupy either 2 or 3 grid squares. A vehicle can only move along its axis of orientation (i.e. horizontal vehicles can only move left and right, verticle vehicles can only move up and down).

You can gain some intuition on how to solve this problem by checking
http://www.eagle-i.com/JAVA/rush.html

Algorithm

You will use A* to find a solve the RushHour game.

A* maintains a fringe of search nodes. Initially, we construct a node that points to the initial state, and insert it into the fringe. Then we repeatedly extract the node which has minimum cost from the fringe, check if it is a solution, and if not insert all the states this can lead to (through any legal move) in the fringe. We stop when we extract from the fringe a node corresponding to a state where the red car exits the grid.

Your function should find a solution if one exist. You also should keep track of the number of nodes that were expanded, the total number of nodes that were placed in the fringe and the runtime of your algorithm.

Code Infrastructure

Copy the code from `lg421075/pj1/` to your directory. Use the provided Makefile with `make` to compile the program. The provided options are:

`make` :	compiles the program building the c library.
`make clean` :	removes compiled files.
`make run` :	runs the project.

There are three basic classes defined:

class State: This is the class that describes each particular state of the problem. Each state describes the *locations[]* for the blocks.

class Node: This is the class that describes each particular node in the search space. Each node has a pointer to a state and a pointer to move that was made to arrive in this node.

class Fringe: This is the class that corresponds to the fringe. Its main task is to keep the Fringe updated by `adding` nodes and `extracting` the node with the minimum cost.

You should implement the function that will `solve` the problem by performing the A*. This function should return the depth of the solution (or -1 if no solution was found) and update the variables that contain the state of the solution found, the number of nodes the number of nodes that were expanded, the total number of nodes that were placed in the Fringe and the running time.

Check the files rushlib.h, rush_deliverable1.cpp and rush_deliverable2.cpp for a better description of the code infrastructure.

The rest Of Code

As you can see the application GUI is implemented in java. You can check the directory `lg421075/pj1_javasrc/` for the source code of the rest of the application and read the comments. This may not help you for the specific task, but it is always a good idea. You will also find there the file rushlib.cpp which contains the rest of the c functions. All the source files have comments. You can find out how the interaction between the gui and your functions is implemented

Inputs

To load a problem you can click the open button and select any file from the directory `lg421075/pj1/input` these files are in text format. you can edit them or create your own. The first line in any file should read "project 1 input format" the second line is the number of cars in this problem. Each of the following lines describes a the type and the position of the ith car. The first car described is always the red one.

Main Variables

`NoOfCars` The variable `NoOfCars` is the number of cars that are in the board (the red car also counts).

`CarTypes` The table `CarTypes` is where you can find information about the type of each car. It is global and you should not need to change it, only read from it.

`State.locations[]`: This array stores the location of cars 0 through NoOfCars-1. The first car (position 0 of the array) will always be the red one. You can find the position of the ith car by reading `locations[i].x` and `locations[i].y` these coordinates will be integers in the range [1..6]

`current_grid[]` This is a scratch pad that you can use to construct a 6X6 grid with the number of the car occuppying each grid cell filled in (-1 if no car is

present). It is offered for convenience and speed when trying to detect which moves are possible. You can use table `current_grid[]` if you want to find quickly if a square is free or what block occupies it. For example if `p[5][5]==0` that means that square is occupied by the first car. If `p[5][5]==-1` that means that square is empty.

1. [**40 points**] **rush_deliverable1.cpp** (classes Node, State & Fringe and function solve) Implement the methods described in rushlib.h that do not have a body. Each file contains comments that describe what each method should do. Your code will be run on several new input files to check that it can find a solution.

2. [**30 points**] Your solve function should find a solution and correctly update the number of nodes created and the number of nodes that were expanded. There are several different possible strategies for avoiding repeated states. Discuss the one that you have chosen. Extra points: compare and contrast different approaches.

3. [**30 points**] **rush_deliverable2.cpp**(Heuristic Function) Currently the heuristic function h(x) just returns 0. That means that the cost of each state is its depth in the search tree. Come up with a better heuristic function and see how that affects the number of nodes created, the number of nodes expanded and the running time of your algorithm.

Deliverables

The deliverables of this project are the two source files named
`rushlib_deliverable1.cpp` and
`rushlib_deliverable2.cpp` and a one page writeup.
Instructions for turning in project will be announced in class.