AI Term Project Report

Student Names and Numbers

- Tarek Khaled El Talawi (30)

Description

Topic to be explored is Planning, where it's intended to solve the Rush Hour problem with the minimum number of movements.

Rush hour puzzle is basically a grid that contains cars and trucks, and there's a red car stuck in traffic and is trying to escape. Cars can be moved up and down or left and right. The goal is to clear a path so that the red car can escape past the a specific gate such as in the picture below.

The program basically takes as input a given state where the red car is stuck, and it's required to output a sequence of movements of trucks and cars such that the car can escape the traffic.

Motivation

I was keen on choosing this as my project because I have been playing this game since forever on my iPhone and it only seemed rational to be able to solve it using hints of my own instead of buying hints from the store.

Challenges

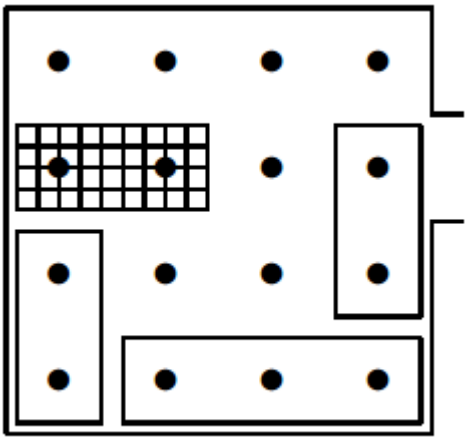
I have face some challenges during implementation:

- Representing the puzzle
- Implementing the A* algorithm + saving the predecessor to print the path taken
- Choosing a suitable heuristic which is admissible
- Finding the neighboring states for every state of the puzzle

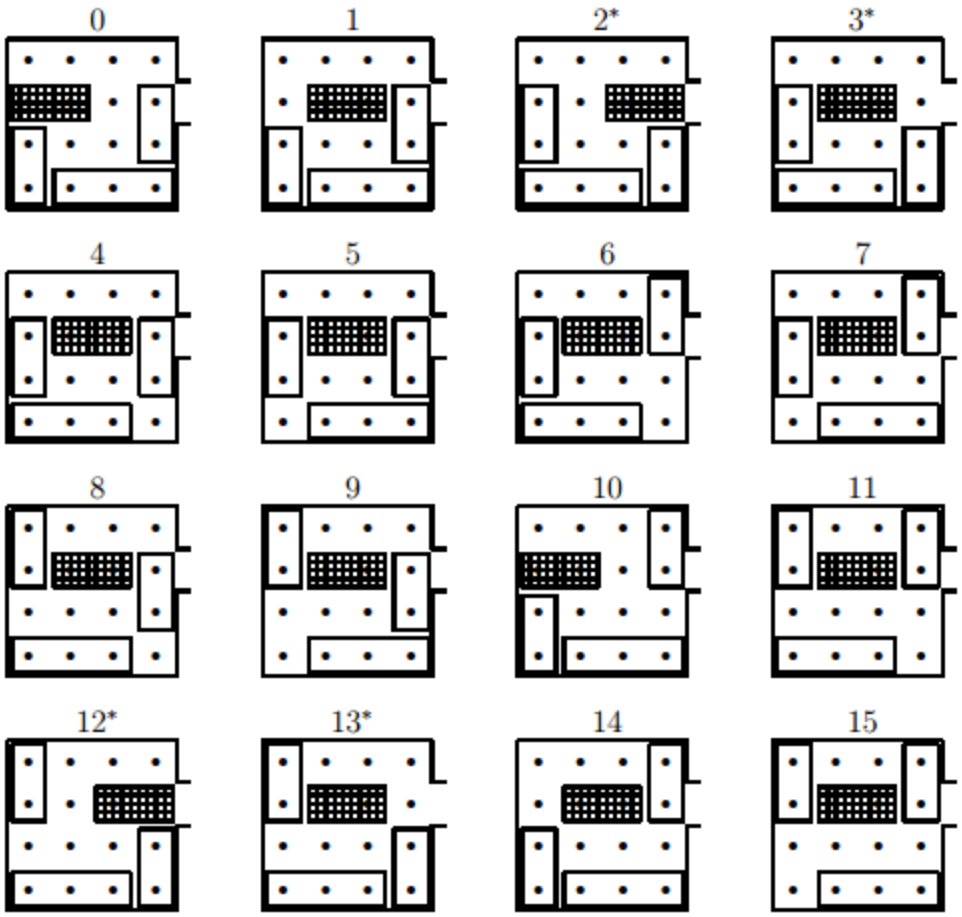
Design

The puzzle would be represented as a graph where nodes represent states of cars and trucks, and neighboring nodes would be every possible 1 move of any car or truck in the grid, then by applying A* algorithm using different given heuristics the program would find the shortest path or sequence of movements such that the red car escapes the traffic.

So for example consider the following state



There are two cars and one truck, therefore there are three placements for each car and two for the truck, for a total of 2 · 3^3 = 54), however only 16 of them are not overlapping.



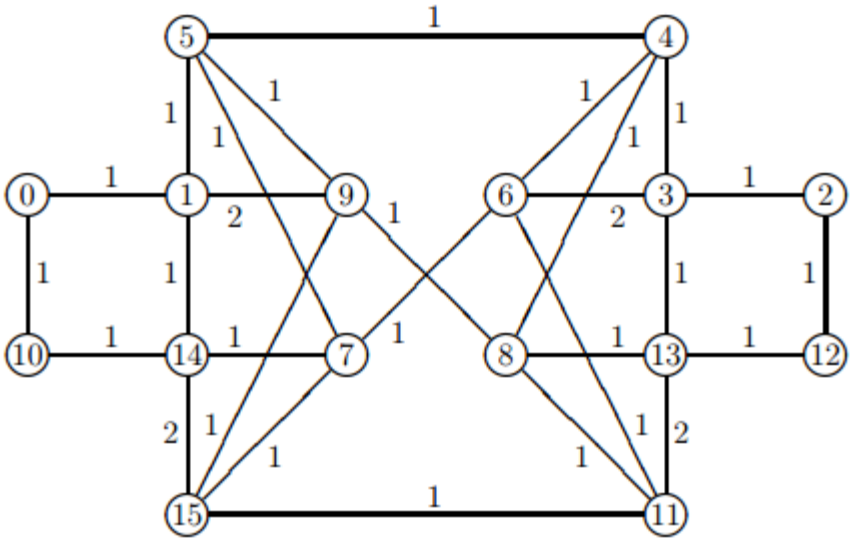
Where the state with asterisks above like 2*, 3* and 12* are considered winning states since they are just one move from final state.

The initial state is 0 and its neighbors are 1 and 10, where only one move is made on some car/truck.

The minimal sequence to escape the car in this example would be 0-1-5-4-3.

The weights of the edges between nodes are represented by the number of squares moved by the car, for example the weight of the edge from state 1 to state 9 would be 2.

Therefore, the graph representing the previous puzzle would be:



In case of zero heuristic algorithm, this will be solved by BFS where the edges with weight 2 would be eliminated as they can already be accessed from other neighboring nodes.

Implementation Details

The project mainly consists of 5 classes:

- Rush Hour
- Puzzle
- State
- Car
- Heuristic

Heuristic

It is an abstract class extended by another children classes Heurisitc1 and Heuristic2.

Heuristic1 calculates the heuristic based on the number of cars blocking the way of the red car to the gate, also if a blocking car is blocked then it will add 2 movements instead of 1.

So for this example:

The red car is followed by a truck which is blocked(can't be moved up or down) therefore number of movements will be incremented by 2, another truck is in the way of the red car at the end of the grid, however it is not blocked as it can be moved downwards therefore it adds only 1.

Therefore the heuristic value = 3 in this state.

Car

It is a class representing each car from the pov of its position, orientation and size. Also the class has some of the car functions like moveUp, moveDown, moveLeft and moveRight.

Puzzle

It basically represents the puzzle as a list of cars and the size of the grid. It implements some functions like:

- Check if a car can move up, down, right or left
- Check if a given point the grid crashes with or intersect with a car

State

This class represents a state of the puzzle where cars can be moved, it is represented by the original puzzle and the cost taken to reach that state from the starting position.

It implements the following functions:

- GetNeighbors() which is responsible for finding all neighboring states that can be reached by moving only one car.
- IsGoal() to determine whether this state is an end state or not.
- ToString() which returns a representation of the state in a string, which is to be used in printing the path later.

Rush Hour

This is the Main class that takes the input first, then runs the A* algorithm twice using the two heuristics provided and calculates the time taken by each.

Functions implemented:

- SearchAStar() which implements the A* algorithm and sets the predecessor list.
- GetPath() uses the predecessor list generated and the goal state to get a list of states from the starting state till the goal state.
- Print() simply prints the path found.
- ReadInput() responsible for the format of the input and generating the corresponding puzzle.

Testing

Used the gaming application on my phone to get different puzzles with different difficulties.

Used the following to represent cars:

- ### for horizontal cars with size 3
- ** for horizontal cars with size 2
- == for red car
- ++ for vertical cars with size 2
- @@@ for vertical cars with size 3

Sample

```
Beginner

    ###..@
    ..@..@
    ==@..@
    +.@.**
    +...+.
    ###.+.

Heuristic = 16
no heuristic = 15

================================================================================

Intermediate

    .+@+**
    .+@+**
    ==@...
    +###.@
    +....@
    ****.@

Heuristic = 22
no heuristic = 21

================================================================================


Advanced

    .+**+.
    .+**+.
    ==+@..
    @.+@..
    @.+@**
    @.+###


Heuristic = 28
no heuristic = 28

================================================================================

Expert

    @.+###
    @.++..
    @==+.@
    ****.@
    ****.@
    ......

Heuristic = 34
no heuristic = 32
```

Sample output for the beginner puzzle

```
Solution using Heuristic.
#########################
Number of optimal movements = 16

Initial state:
###..@
..@..@
==@..@
+.@.**
+...+.
###.+.

Step 1:     Step 2:     Step 3:     Step 4:     Step 5:     Step 6:
###..@      ###...      ...###      ..@###      ..@###      ..@###
..@..@      ..@...      ..@...      ..@...      ..@...      ..@...
==@..@      ==@...      ==@...      ==@...      ==@...      ==@...
+.@**.      +.@**@      +.@**@      +..**@      +..**@      ...**@
+...+.      +...+@      +...+@      +...+@      +...+@      +...+@
###.+.      ###.+@      ###.+@      ###.+@      .###+@      +###+@

Step 7:     Step 8:     Step 9:     Step 10:    Step 11:    Step 12:
..@###      ..@###      ..@###      ..@###      ...###      ...###
..@...      ..@...      ..@...      ..@...      ......      ......
==@...      ==@...      ==@..@      ==@..@      ==...@      ...==@
**...@      **..+@      **..+@      **..+@      **@.+@      **@.+@
+...+@      +...+@      +...+@      +...+@      +.@.+@      +.@.+@
+###+@      +###.@      +###..      +..###      +.@###      +.@###

Step 13:    Step 14:    Step 15:    Step 16:
...###      ...###      ...###      ...###
......      ......      ......      ......
..@==@      ..@==@      ..@==.      ..@.==
**@.+@      **@.+@      **@.+@      **@.+@
+.@.+@      +.@.+@      +.@.+@      +.@.+@
+..###      +###..      +###.@      +###.@

Time taken using heuristic : 93
```

```
Solution with no Heuristic used.
################################
Number of optimal movements = 15

Initial state:
###..@
..@..@
==@..@
+.@.**
+...+.
###.+.


Step 1:     Step 2:     Step 3:     Step 4:     Step 5:     Step 6:
###..@      ###...      ...###      ..@###      ..@###      ..@###
..@..@      ..@...      ..@...      ..@...      ..@...      ..@...
==@..@      ==@..@      ==@..@      ==@..@      ==@..@      ==@..@
+.@**.      +.@**@      +.@**@      +..**@      +..**@      ...**@
+...+.      +...+@      +...+@      +...+@      +...+@      +...+@
###.+.      ###.+.      ###.+.      ###.+.       .###+.      +###+.


Step 7:     Step 8:     Step 9:    Step 10:    Step 11:    Step 12:
..@###      ..@###      ..@###      ...###      ...###      ...###
..@...      ..@...      ..@...      ......      ......      ......
==@..@      ==@..@      ==@..@      ==...@      ...==@      ..@==@
**...@      **..+@      **..+@      **@.+@      **@.+@      **@.+@
+...+@      +...+@      +...+@      +.@.+@      +.@.+@      +.@.+@
+###+.      +###..      +..###      +.@###      +.@###      +..###


Step 13:    Step 14:    Step 15:
...###      ...###      ...###
......      ......      ......
..@==@      ..@==.      ..@.==
**@.+@      **@.+@      **@.+@
+.@.+@      +.@.+@      +.@.+@
+###..      +###.@      +###.@

Time taken without using heuristic : 94
```

References

- http://www.cs.sjsu.edu/~stamp/cv/papers/rh.pdf
- http://www.cs.princeton.edu/courses/archive/fall04/cos402/assignments/rushhour/