Spring 2021

Project Documentation

Sadaf Ahmadi: 301437929
Elmira Amirian: 301446534

CMPT 225

IGOR SHINKAR

In this project we were trying to solve rush hour game using BFS, DFS and A* algo. At the first step what we did was modeling rush hour to the style of BFS, DFS and A*, we used class board as a data structure to do this.
We would like to discuss class Car first and then we'll get back to class board again. In class Car, we're saving information about cars in rush hour. Here is an explanation of this class:

1. It has the following variables inside it:
    - 1.1. Starting point including row and column as two integers: start_row and start_column
    - 1.2. As we only have two options of being horizontal or vertical, we don't really need to have both end_row and end_column, so there's only one variable "end"
    - 1.3. Variable length which simply contains the length of the car
    - 1.4. Variable orient which contains the orientation of the car
      1 for horizontal and 0 for vertical
    - 1.5 Last variable is just a string, which contains the name of the car

2. In terms of methods for this class, there are its getters and setters and constructor and 3 other important methods:
   2.1 **Change method:** when we're reading a car from file, we might face a car that its length is more than two, this method will change the end position and obviously increases length by one each time, based on the fact that car is either vertical or horizontal. If it's horizontal, with the same row, column would increase and if it's vertical, with the same column, row would increase.
   2.2 **isTouch method:** it returns true if the end of the car is equal or greater than 5 which means it reaches the end of the path or not
   2.3 **equals method:** if entry object is null=>false
   If it has the same reference=> true
   If it's not an instance of car => false
   If none of above it just returns "and" of all its variables

Now, it's time to get back to class board which is the states where we use BFS, DFS and A* algos. In this class we would like to set number of cars, place of them with some other variables. We're using Hash Map for this purpose. In line 14 of the code, when we say HashMap< String, Car>, string would be keys and Car would be the value for that key, the reason that we chose this data structure is because in files we're showing cars with some letters that are basically strings which can be mapped to specific cars.
We would like to get the first board and create next steps using that. Successor method is where this is happening. In a simple language in Successor, we're creating a list of all possible states that can happen as a next move and we return that list. Let me give an example of that to make it clearer, let's assume we have a board with three cars, two of them are vertical and one is horizontal, and let's assume that the one is horizontal can only move one step forward and can't go back as there's a wall there while the other two vertical ones can move both up and down. In this case successor would return a list of size 5 which is these 5 possible moves: (let's name cars A, B and C)

1. A moves one step forward, B and C don't move

2. B moves one step upward, A and C don't move
3. B moves one step downward, A and C don't move
4. C moves one step upward, A and B don't move
5. C moves one step downward, A and B don't move

So, the list of all these possible moves would be what Successor returns and later we'll use this for BFS, DFS algos.
Another method, I would talk about is isGoal which checks if a car with name "X" is touched or not.
There are also some other methods that we'll use them somewhere else like getFreeCount which is the heuristic that we use in A*.
Next method is getKey, for each board we need a key, and we're assuming the board as string for its key. When board changes, we need a new key for new board that's why there's method getKey to create key for each board.
The rest is just constructor, getter, setter that doesn't really need any explanation I believe.
Last class would be class Algorithm, it has 4 methods. BFS, DFS, A* and heuristic.
To avoid getting stock in infinite loops we need a data structure to keep track of visited states.
BFS:

> Adds the first state to the queue
> While queue is not empty, it creates new states, using successor method that we already discussed, and checks if the new states isGoal or not. If it's goal, that's the solution. If not keep going.
> There's also one more thing here, which is "visited" HashMap: key is string (using getKey from board that is already discussed) and value is integer (for the ones that are already visited value is 1)

DFS is exactly the same thing, the only difference is the data structure which is Stack instead of Queue and same with A* algo which is priority Queue.

In terms of the time that we spent in different parts of the code, I would say one of the very first challenges were figuring out from where and how we should start and then when we had a rough idea, class Car was the easiest class to write while class Board and class Algorithm were the most challenging ones respectively.
We did most parts together (mostly in person and some through zoom calls) as we thought it would be easier to figure out the solution as we talk to each other about it. The only part that we did it separately was in class Algorithm, as we were getting close to due date and wanted to do it faster, so Sadaf worked on BFS and Elmira worked on DFS and then at last we wrote A* and this documentation file together.