Arisa Chue

Mr. Eckel

AI 1 Period 2

20 October 2020

Modeling Challenge II: Rush Hour

I first tackled the problem of how to set up the board. We had to differentiate between the vehicles (truck, car, red car, or blank) and also the direction they were facing (vertical or horizontal). To make sure two horizontal vehicles were not confused with two vertical vehicles, I made each vehicle a separate number and repeated the same number for the length. For example, a board would look like this, “12335x12445xRR76xxxx76xxxxx6xxxxxxxx”, which would be displayed as:

1 2 3 3 5 x

1 2 4 4 5 x

RR 7 6 x x

x x 7 6 x x

x x x 6 x x

x x x x x x

By separating each vehicle with a different number, I can avoid mixing two side by side vehicles and also still get the length of the car/truck. I coded a method that can print an input string into a readable board like the one shown above.

Next, I worked on my get\_children() method. My thinking was to take in a string of the board as the parameter and then use a for loop to traverse this string. At each index, my code would check if the character is a number or R (red car), and if it is, I would generate all the movable tiles of that vehicle (I called another method called move\_tile() for this). I would prevent calling move\_tile() multiple times for the same vehicle by adding the number or R to a visited set and check if I hadn’t called move\_tile() on the same vehicle before. I got my code to work for this method.

In the move\_tile(), I had to first check which direction the vehicle was in to know whether I could slide the block up/down or right/left. I do this by checking the character right next to the letter. If the character next to the letter is equal to the letter, I would set a boolean variable as True (True for horizontal, False for vertical). I then separated my method into two, one if statement for horizontal vehicles and the other else statement for vertical vehicles. Both for horizontal and vertical moves, I would create a while loop that keeps moving the block in the specified direction until the block would go outside the board’s boundaries or hit another block. Each time the block moves, I built a new string with the new block’s position, making sure to replace the blank “x” with the number and vice versa, and added this string to a list. move\_tile() would return a list of all the possible boards (formatted as strings) for that vehicle. To have my code work for this method, I also had to create two other methods that converted an index to x and y coordinates or x and y coordinates to an index. I had already created these two methods for sliding puzzles so I know these helper methods work successfully.

Theoretically, now my code should be able to generate all the possible moves for each board state using the methods get\_children() and move\_tile(). I then used a BFS search algorithm from sliding puzzles pt1. The only part I changed was the method that checked whether the current state was the goal state or not, which I checked by seeing if the exit coordinate and the one left to it was both an “R”.

After passing a sample board state into my BFS search and trying to solve the puzzle, my code would keep running forever, which is the problem I was stuck on when class ended. One of the difficulties of Rush Hour is that there are so many possible children board states since a block could move one tile, move across the board, or anywhere in between. I think my code currently does not work because the graph it creates is so large, with so many branches to go through to find the goal state. Using a BFS search is not the most efficient take on such a large graph since it would go through each level to find the shortest path. In addition, my code has a lot of for loops (I have three nested for loops to generate the children), which is also very inefficient. If I were to continue on this lab, I should implement an a\*star search or a biBFS search and see if my code can execute with a faster search method.