CMPT225 Final Project

Course: CMPT 225 Spring 2021

Instructor: Igor Skinkar

Wen Luo Lui (Thomas) [thomas\_lui\_3@sfu.ca](mailto:thomas_lui_3@sfu.ca)

Student Number: 301310026

My choices of classes are decided at the thinking stage of the project and later modify during the implementation stage of the project. The classes’ structure consists of 4 classes: the Solver class, the RushHourGraph class, Vertex class, and Board class. The Solver class consists of the key method called solveFromFile which takes in a string input path and a string output path. This method is used for the initial reading of the rush hour board from the file given and used to creating the rushhourgraph object with the input of a type Board object (the original board). And such rushhourgraph object is then used to call the startingSolver method with the input of the original board object and return of a type Stack object with vertex of board inside it. This stack indicates the path to solving the board and each of the vertex in the stack is the required car move data (car name, moved slot, direction) to successfully solve the board. Furthermore, the stack is used to generate the solution files based on all of the correct movement data such as (EL3 which indicate car E, left direction, and moved 3 slot left). And, for the Rushhourgraph class, I chose to implement the core of the project with a basic Breadth First Search algorithm as it can potentially find the shortest path to solve each of the rush hour board test cases. Also, the Vertex class and the board class are used to store each of different board’s data and such that the target board can locate the path to the original board through the use of setting parent vertex. And, since I chose the Breadth First Search algorithm, heuristics was never considered as BFS algorithm does not require it. Furthermore, the BFS algorithm implementation approach of the project is rather simple. The original board vertex is first added to the queue and while the queue is not empty we removed the first/current vertex and evaluate if the vertex have already been visited/explored such that none of the identical vertex will be created further down the program. And, if the vertex has never been visited then we can add it to a hash set that consist of explored board (board that is not the target board). Then, we use the current vertex’s board to get all its neighbouring board and assigned to a hash set. And, a for loop will be used to iterate through all neighbouring board or the child board of the current vertex board. If one of the board that has not been visited (or not exist in the explored hash set), then we will create a new vertex based of unexplored board and set the vertex parent to be the current vertex and add such vertex to the queue that we created earlier on. This process will be repeated until the target board is found.

During the implementation process of the project, some of the difficulties that I have encountered includes the generation of all the possible states of the board and the process of constructing a memory efficiently representation each of the board. The generation of all possible state of the board is far the hardest portion of the project as I need to ensure all possible state are created and each state of the board will not be repeatedly constructed more than once despite part of the BFS algorithm have taken care of this issue. For each of the state of the board, the car movement needs to follow the proper rule of the rush hour game such that no cars will be replaced with other cars in the board or moved out of the board. Furthermore, another issue that I encountered is the process of constructing a memory efficient representation of the board. At first, I have store each of the board with the data type of char multidimension array however, this led me into an out of memory error. Then later, I switched the representation of the board with only a single string data type which helped me resolved the out of memory error. To conclude, the project is rather simple with the BFS approach as heuristics does not have to be considered.