Rush-hour by Spitsuur

Y. Kada, R.F.G. Wagid Hosain

19-12-2014

1. Introduction

For the course Programming Theory (Heuristics) at the University of Amsterdam team Spitsuur has chosen the case Rush-hour to solve. The case Rush-hour seemed interesting to solve, because it was a well-known game played by team Spitsuur and it was exciting to find out how a board could be solved in as little as possible steps.

1.2 The Case

The Rush-hour puzzle exists of a six by six grid which has one red car, a number of other cars, depending on the difficulty the number of other cars vary, and an exit. The red car is always horizontally placed on the same level as the exit and it always occupies two grids, the other cars’ length can vary from two to three grids and can be placed horizontally or vertically on the board. The goal of this puzzle is to move the red car to the exit by moving the other cars, which are blocking the red car, out of the way.

The case’s goal is, however, to move the car to the exit with as little as possible steps. A step can be interpreted in two ways. These ways are explained with the help of the figure below. Moving the orange car opposite of the red dot to the red dot can be interpreted as one step or as four steps. In other words: the number of steps for moving a car as far as it possibly can, get can be interpreted as the number of grids it used to move or as one step. Team Spitsuur has decided to interpret the number of steps as the number of grids the car used to move. This results in a higher number of steps used to solve a puzzle than when interpreting the car its far as possible move as one.

The case has provided seven puzzles to solve. Three of these puzzles are six by six and vary in difficulty, another three are nine by nine and also vary in difficulty and one of these are twelve by twelve, which is the hardest one to solve. The goal team Spitsuur has set is to create a computer program that solved at least the three six by six board in as little as possible steps and in as little as possible time.



2. Methods

To solve this puzzle and to reach the goal set, team Spitsuur has written a computer program in Python 2.7, which is called the skeleton and a computer program with algorithms to solve the puzzle. The reason for choosing Python 2.7 is because team Spitsuur was already familiar with this programming language.

2.1 The computer program - Skeleton

The computer program, rush.py, exists of two classes and another program, rushvisua.py, to visualize how the boards are being solved.

The program rush.py has two classes, namely, the Board class and a Car class. The Board class gets a .txt file as input and parses this information into a dictionary with all the existing car’s ids as key and the car’s position, meaning which coordinates on the board the car occupies as values in a tuple. The Board class has the full knowledge of all the cars on the board and distinguishes between the red car and all the other cars. This class further has knowledge of the dimensions of the board, where the exit is and where the empty spots are i.e. grids that are not occupied by a car.

With this class it is therefore possible to check whether a car can be moved. If the case is that the car can be moved because there are empty spots near the car, it is possible to move the car up, down, left or right depending on their direction, namely horizontally or vertically. If a car is placed horizontally the car can only move to the left of right and if a car is placed vertically it can only move up or down. After moving a car a new Board object is created with in the dictionary the new positions on the board.

The Auto class has knowledge of all the cars its properties such as the length of the car, the color of the car and the direction of the car, horizontally or vertically.

The computer program rushvisua.py visualizes the generated Board objects by creating a board which has the number of grids belonging to the dimensions given in the .txt file and an exit. After creating this, the cars are generated and placed on the given positions. If the color red is specified behind a car in the .txt file, that car will be colored red and the other cars will have a random color. After this the dictionaries of each board object are read by this program and are being visualized, resulting in moving cars and the red car being moved to the exit.

To test if these programs were working properly team Spitsuur has generated a .txt file with small dimensions and few cars.

2.2 The algorithms

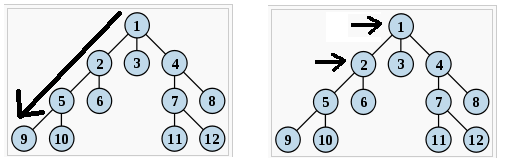
After the skeleton was build and was working properly the provided puzzles could be solved. To efficiently solve these puzzles, algorithms were implemented into the program.

2.2.1 Depth-first search

The first algorithm implemented was depth-first search. Depth-first search is an algorithm which searches each branch of a tree as far as possible, before going back to search in another branch. This algorithm was implemented recursively.

2.2.2 Breadth-first search

The second algorithm implemented was breadth-first search. Breadth first search is also an algorithm to search trees, however this three starts at a node and checks all its children instead of searching in each branch in the depth, it searches in the width of the tree. The figure below can make this explanation more clear.



3. Results

4. Discussion

5. Conclusion