Rush Hour

Tim Kwist

Version 1.0

October 1, 2014

Table of Contents

Rush Hour

Author:
Tim Kwist

Version:
1.00

Date:
Wednesday, October 1, 2014

File Index

File List

Here is a list of all documented files with brief descriptions:

File Documentation

RushHour.cpp File Reference

```
#include <iostream>
#include <stdio.h>
```

Functions

- void **buildBoard** ()
- bool **forward** (int)
- bool **backward** (int)
- int **solve** (int, int, int, int)
- bool **solved** ()
- int **main** ()

Variables

- char ** **board** = new char*[6]
- int ** **listOfCars**
- int **MAX** = 10
- int **LOW** = 11
- int **numCars** = -1

Function Documentation

bool backward (int *carNumber*)

Move the car backwards if possible. For vertical cars, this is down. For horizontal cars, this is right

Parameters:

carNumber	Car being moved backwards

Returns:

True if car can be moved backwards, false otherwise

Precondition:

There are cars on the board to be moved

Postcondition:

The car specified will be moved backwards if possible

void buildBoard ()

Function headers

Method implementation Get input from the user to build the board Input should come in the following order:

- number of cars
- [length of car] [car orientation] [row of car] [col of car] if number of cars is 0, skip getting any more input and assume end of file This method will also update the list of cars

- **Parameters:**

None	

- **Returns:**
  None
- **Precondition:**
  None
- **Postcondition:**
  The board will be generated based on given input, and list of cars will also be updated
-

bool forward (int *carNumber*)

Move the car forward if possible. For vertical cars, this is up. For horizontal cars, this is left.

Parameters:

carNumber	Car being moved forward

Returns:
True if car can be moved forward, false otherwise

Precondition:
There are cars on the board to be moved

Postcondition:
The car specified will be moved forward if possible

int main ()

Main method implementation This main method will set up the initial board, and will, on loop, ask for input of the rush hour problem, solve the board, and output how many moves the solution takes.

Precondition:

int solve (int *carNumber*, int *currentMoveNumber*, int *moveDirection*, int *currentStack*)

Use a brute force, breadth-first search to try to solve the rush hour puzzle. If a puzzle takes more than 10 moves, its node will be terminated. If the ratio of moves to stack calls is too high, it is assumed that the program is stuck in an infinite loop and the node will be terminated as well.

Parameters:

carNumber	car currently being evaluated for solving
currentMoveNumber	Current number of moves used for current stack call
moveDirection	Direction that current car is being moved
currentStack	Current number of stack call of this method

Returns:
Always returns 0

Precondition:
None

Postcondition:
The grid will be set back to its original state, and nothing will be changed except for the global variable LOW

int carNorth = -1; int carSouth = -1; int carWest = -1; int carEast = -1;

for(int i = curRow; i > 0; i–) { if(board[i][curCol] != '.' && board[i][curCol] != carNumber) { carNorth = board[i][curCol]; } } for(int i = curRow; i < 6; i++) { if(board[i][curCol] != '.' && board[i][curCol] != carNumber) { carSouth = board[i][curCol]; } } for(int j = curCol; j > 0; j–) {

if(board[curRow][j] != '.' && board[curRow][j] != carNumber) { carWest = board[curRow][j]; } } for(int j = curCol; j < 6; j++) { if(board[curRow][j] != '.' && board[curRow][j] != carNumber) { carEast = board[curRow][j]; } }

if(carNorth != -1) { solve(carNorth, currentMoveNumber, 1, currentStack); // Forward solve(carNorth, currentMoveNumber, 1, currentStack); // Backward } if(carWest != -1) { solve(carWest, currentMoveNumber, 0, currentStack); // Forward solve(carWest, currentMoveNumber, 1, currentStack); // Backward } if(carSouth != -1) { solve(carSouth, currentMoveNumber, 0, currentStack); // Forward solve(carSouth, currentMoveNumber, 1, currentStack); // Backward } if(carEast != -1) { solve(carEast, currentMoveNumber, 0, currentStack); // Forward solve(carEast, currentMoveNumber, 1, currentStack); // Backward } if(moveSuccessful) { solve(carNumber, currentMoveNumber, moveDirection, currentStack); }

bool solved ()

Determine whether the rush hour puzzle is solved by checking if the car is on the last column

Parameters:

None	

Returns:

True if car is on 5th column, false otherwise

Precondition:

None

Postcondition:

No variables will be changed

Variable Documentation

char** board = new char*[6]

Global Variables

Index

INDEX