Rush Hour

Generated by Doxygen 1.7.6.1

Tue Feb 25 2014 22:45:01

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Vehicle Struct Reference

Public Attributes

- int **vehicleRow**
- int **vehicleCol**
- int **vehicleType**
- char **vehicleDirection**

The documentation for this struct was generated from the following file:

- rush.cpp

Chapter 4

File Documentation

4.1 rush.cpp File Reference

This program sets up a game board with vehicles on it and solves the game of rush hour in the least amount of moves.

```
#include <iostream>
```

Classes

- struct Vehicle

Functions

- void setBoard (char gameBoard[6][6], Vehicle vehiclesInGame[], int numberOfVehicles)
- bool moveVehicleForward (char gameBoard[6][6], Vehicle vehiclesInGame[], int vehicleNumber)
- bool moveVehicleBackward (char gameBoard[6][6], Vehicle vehiclesInGame[], int vehicleNumber)
- bool solveIt (char gameBoard[6][6], Vehicle vehiclesInGame[], int numberOfVehicles, int numberOfMoves, int &bestMoves)
- int **main** ()

4.1.1 Detailed Description

This program sets up a game board with vehicles on it and solves the game of rush hour in the least amount of moves.

Author

Amardeep Singh

Date

February 26th 2014

4.1.2 Function Documentation

4.1.2.1 bool moveVehicleBackward (char *gameBoard[6][6],* Vehicle *vehiclesInGame[],* int *vehicleNumber*)

This function moves a vehicle backward.

This function moves a vehicle backward on the game board. The vehicle checks a few different scenarios to see if it is actually possible to move the vehicle backward before it actually does. The function checks the bounds to see if the vehicle is already not at the edge of the gameBoard. If the vehicle is at the edge then it returns false. Given the type of vehicle & it's orientation it checks to see if there is a 'X' behind of the vehicle. If there is an 'X' this means that there is an empty spot in front of the vehicle and it is okay for it to move backward. If there is any other symbol there it means that the car cannot move backward and it returns false. If the vehicle can move backward then the function moves the vehicle backward, marks the new location on the board and updates the row or column of the vehicle.

Parameters

gameBoard[][]	The 2D char array to represent the game board.
vehiclesInGame	An array of vehicles containing all vehicles in current game.
vehicleNumber	This number represents the vehicle to be moved.

Precondition

The vehicle is in it's current location

Postcondition

If the vehicle can move backward the vehicle is moved backward and the gameBoard is updated with the new location of the vehicle.

Returns

The function returns a bool of whether or not the vehicle was moved backward.

4.1.2.2 bool moveVehicleForward (char *gameBoard[6][6],* Vehicle *vehiclesInGame[ ],* int *vehicleNumber*)

This function moves a vehicle forward.

This function moves a vehicle forward on the game board. The vehicle checks a few different scenarios to see if it is actually possibel to move the vehicle forward before it actually does. The function checks the bounds to see if the vehicle is already not at the edge of the gameBoard. If the vehicle is at the edge then it returns false. Given the type of vehicle & it's orientation it checks to see if there is a 'X' in front of the vehicle. If there is an 'X' this means that there is an empty spot in front of the vehicle and it is okay for it to move forward. If there is any other symbol there it means that the car cannot move forward and it returns false. If the vehicle can move forward then the function moves the vehicle forward, marks the new location on the board and updates the row or column of the vehicle.

Parameters

gameBoard[][]	The 2D char array to represent the game board.
vehiclesInGame	An array of vehicles containing all vehicles in current game.
vehicleNumber	This number represents the vehicle to be moved.

Precondition

The vehicle is in it's current location

Postcondition

If the vehicle can move forward the vehicle is moved forward and the gameBoard is updated with the new location of the vehicle.

Returns

The function returns a bool of whether or not the vehicle was moved backward.

4.1.2.3 void setBoard (char *gameBoard[6][6],* Vehicle *vehiclesInGame[ ],* int *numberOfVehicles*)

This functions sets the game board up.

This function sets up the game board with vehicles so the game will be ready for playing. The front of the vehicle is already set and it is used to mark the board with the appropriate amount of spaces taken up by the vehicle given its a car or a truck. Before the vehicles are place upon the board, the entire board is marked with an 'X' to represent a empty slot which contains no vehicle.

Parameters

gameBoard[][]	The 2D char array to represent the game board.
vehiclesInGame	An array of vehicles containing all vehicles in current game.
numberOfVehicles	The number of vehicles in the current game.

Precondition

The gameBoard is empty and uninitialized.

Postcondition

The gameBoard contains all the vehicles and is marked with an 'X' for empty spaces and the number of the vehicle in the appropriate amount of spaces.

4.1.2.4 bool solveIt (char *gameBoard[6][6],* Vehicle *vehiclesInGame[],* int *numberOfVehicles,* int *numberOfMoves,* int & *bestMoves*)

This funciton solves the rush hour game.

This function moves the vehicles forward and backward until a solution ten moves or under has been reached. The function uses recursion to accomplish this task. The base case of the function includes two stopping conditions. The first stopping condtion is attained if the vehicle reaches the end of the board. The second stopping condition is if the numberOfMoves exceeds the bestMoves or the least amount of moves required to solve the game. If neither stopping condition applies then the funciton calls itself recursively. The function loops through all of the vehicles and checks to see if they can move forward or if they can move backward. If either is true then it calls the function again by with numberOfMoves incremented by one. Once the recursive call reaches an end it backtracks and moves the vehicle back to it's original location. The lowest number of moves it took to solve the game is recorded and the function returns if the game was solved or not.

Parameters

gameBoard[][]	The 2D char array to represent the game board.
vehiclesInGame	An array of vehicles containing all vehicles in current game.
numberOfVehicles	The number of vehicles in the current game.
numberOfMoves	Records the number moves that have passed
bestMoves	records the lowest number of moves required to solve the game.

Precondition

the board will be in its original state

Postcondition

all possible solutions will have been found and the lowest number of moves required to solve the game will be records

Returns

The function will return a bool stating wheter or not the game has been solved.