# Rush Hour with STL

Tim Kwist
Version 1.0
December 5, 2014

# Table of Contents

Table of contents

# Rush Hour Using STL

**Author:**

Tim Kwist

**Version:**

1.00

**Date:**

December 5, 2014

# Class Index

## Class List

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

# File Index

## File List

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

# Class Documentation

## Board Class Reference

### Classes

- struct **Car**

### Public Member Functions

- **Board** (int r=6, int c=6)
- **Board** & **operator=** (const **Board** &other)
- void **addCars** ()
- void **swapCells** (int, int, int, int)
- bool **moveForward** (int)
- bool **moveBackward** (int)
- bool **canMoveHere** (int row, int col) const
- bool **isSolved** () const
- string **boardToString** () const
- void **printEverything** () const

### Public Attributes

- int **rows**
- int **cols**
- int **numberOfCars**
- int **numberOfMoves**
- vector< **Car** > **listOfCars**
- vector< vector< char > > **board**

---

### Constructor & Destructor Documentation

#### Board::Board (int *r* = 6, int *c* = 6)

**Board** constructor. Sets rows and columns to parameters or default of 6. Sets number of cars and number of moves to 0 Initializes 2D vector of all periods

**Parameters:**

| | |
|---|---|
| *r* | : rows |
| *c* | : cols |

**Returns:**

None

**Precondition:**

None

**Postcondition:**

None

---

## Member Function Documentation

### void Board::addCars ()

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

-

### string Board::boardToString () const

Convert the board to a string and return it. Pseudo-code: create a temp string, then concat every character on the board to the string and return the string.

**Parameters:**

| | |
|---|---|
| *None* | |

**Returns:**

string : string containing all the characters in the board.

**Precondition:**

None

**Postcondition:**

No variables will be changed

### bool Board::canMoveHere (int *row*, int *col*) const

Determine whether the given position in the 2D vector can be moved to. Pseudo-code: Check if the row and col given are within the bounds of the array ( >= 0, < #rows/cols) Check if spot on board is a period. If it is, true; else, false.

**Parameters:**

| | |
|---|---|
| *row* | : row of the spot being checked if valid move |
| *col* | : column of the spot being checked if valid move |

**Returns:**

bool : true if the spot is valid to move to, false if not

**Precondition:**

None

**Postcondition:**

No variables will be changed

### bool Board::isSolved () const

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

**Parameters:**

| | |
|---|---|
| *None* | |

**Returns:**

True if car at index 0 is on 5th column, false otherwise

**Precondition:**

None

**Postcondition:**

No variables will be changed

### bool Board::moveBackward (int *i*)

Move the car backwards if possible. For vertical cars, this is down. For horizontal cars, this is left Pseudo-code: Checks to see if the desired location to move the car is A. valid and B. empty If it is, swap cells and change row/col to update position after move.

**Parameters:**

| | |
|---|---|
| *i* | : Index of **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

### bool Board::moveForward (int *i*)

Move the car forward if possible. For vertical cars, this is up. For horizontal cars, this is right. Pseudo-code: Checks to see if the desired location to move the car is A. valid and B. empty If it is, swap cells and change row/col to update position after move.

**Parameters:**

| | |
|---|---|
| *i* | : Index of **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

### Board & Board::operator= (const Board & *other*)

Equal operator overload for board Creates deep copy of other board

**Parameters:**

| | |
|---|---|
| *other* | : other board that this one is being set to |

**Returns:**

**Board**& : returns this board after it has been deep copied

**Precondition:**

None

**Postcondition:**

None

### void Board::swapCells (int *r1*, int *c1*, int *r2*, int *c2*)

Swap cells on the 2D vector

**Parameters:**

| | |
|---|---|
| *r1* | : row of the first item to be swapped |
| *c1* | : column of the first item to be swapped |
| *r2* | : row of the second item to be swapped |
| *c2* | : column of the second item to be swapped |

**Returns:**

None

**Precondition:**

None

**Postcondition:**

None

---

### The documentation for this class was generated from the following file:

- Board.h

# Board::Car Struct Reference

## Public Member Functions

- **Car** & **operator=** (const **Car** &other)

## Public Attributes

- int **x**
- int **y**
- int **length**
- char **orientation**

---

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

- Board.h

# File Documentation

## rushHour.cpp File Reference

```
#include <iostream>
#include <stdio.h>
#include <set>
#include <queue>
#include "Board.h"
```

### Functions

- int **solveBoard** ()
- int **main** ()

---

### Function Documentation

#### int main ()

Main method: continue through multiple scenarios of rush hour until calling the solveBoard method returns -1.

**Parameters:**

| | |
|---|---|
| *None* | |

**Returns:**

int : mandatory for main method

**Precondition:**

None

**Postcondition:**

None

#### int solveBoard ()

Solve the rush hour puzzle given to us using a breadth-first search. Pseudo code:

1. Set up a queue that will hold each possible move and a set that will hold a "snap shot" of each board to check whether we've encountered it before or not.
2. Call the board's addCars method to get all the input of the cars and set them up on the board. If the number of cars on the board is 0, exit out of the method. Otherwise, push this board onto the queue.
3. While loop 3a. Set the current board to the board from the front of the queue, and pop the queue. 3b. Check if the board is solved. If it is, return the number of moves stored on the board. 3c. Otherwise, create a snapshot of this board in the form of converting the board to a string. 3d. If the snapshot is found in the set (dejaVu), don't go any farther in the loop, go back to the beginning. 3e. Otherwise, insert the snapshot into dejaVu and go into our for loop 3f. For each car on the board: 3fa. Try to move the car forward; if we can, increase the number of moves, push it onto the queue, then decrease the number of moves and move it back to where it was. 3fb. Do the same with backwards; check if we can do it, and if we can then increase the number of moves, push it onto the queue, then decrease the number and move it back to where it was.

**Parameters:**

| | |
|---|---|
| *None* | |

**Returns:**

int : Smallest number of moves required to solve this board; if number of cars on the board is 0, return -1

**Precondition:**

None

**Postcondition:**

None

# Index

INDEX