2020-01-06

Programming Assignment: Rush Hour Puzzle

Shin Hong

1. Overview

This programming assignment is to construct a C program for a user to play the Rush Hour puzzle using command-line interface. You are given skeleton code which declares the design and the basic structure of the program. You are asked to understand the given program and complete this by exercising various aspects of C programming properly to fulfill all functionalities of the Rush Hour puzzle.

2. Background: Rush Hour Puzzle

A Rush Hour puzzle consists of a 6-by-6 grid (e.g., Figure 1) where n number of cars, c_1, c_2, ..., c_n, are placed without any overlapping. This grid represents a parking lot where the exit is located right-side of F4. The player is the driver of car c_1 who wants to move c_1 to F4 in order for exit.

Each car c_i takes 1 to 6 cells consecutive lined either vertically or horizontally on the grid. A car can move by the player vertically if it is initially laid vertically, or it can move horizontally if it is initially laid horizontally. At each turn, the player can move a car by one cell in the possible direction. In this game, a car cannot rotate, or go outside the board by any cell. In addition, no two cars can be placed on the same cell at the same time (i.e., no overlap). The player moves cars over multiple turns. The player accomplishes the goal of the game when c_1 is placed on F4

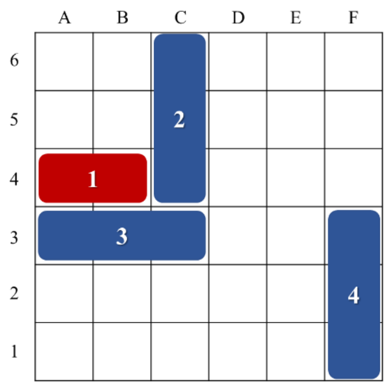

Figure 1. The game board and 4 cars.

3. Program Design

3.1. User interface

This program uses command-line interface: it interacts with a user (the player) via the standard input and the standard output. At each turn, the program first receives a line from the standard input as a command. After that, the game program handles each command by updating the program state, and, if needed, displaying the results. The following is the list of commands that a user can give:

- `start` *file-name*
  Read a file *file-name* to load the initial setting of a game (find more details in Section 3.2), and begin the game. Display the game board if the file is properly loaded. Or, print out "invalid data" if *file-name* does not have valid data.
- `right` *i*
  Move c_i toward right by one cell. Display the game board after the move. Print "`impossible`" if the move is not possible.
- `left` *i*
  Move c_i toward left by one cell. Display the game board after the move. Print "`impossible`" if the move is not possible.
- `up` *i*
  Move c_i upward by one cell. Display the game board after the move. Print "`impossible`" if the move is not possible.
- `down` *i*
  Move c_i downward by one cell. Display the game board after the move. Print "`impossible`" if the move is not possible.
- `quit`
  Terminate the program immediately.

The program must notify that the game is done when c_1 get placed on F4 by a command. For miscellaneous invalid commands, the program must print "`invalid command`" to the standard output.

3.2. File format

The program reads the game board setup from a text file following certain rules: first line of the file must have one integer that represents the number of cars, n. This number is at least 2 and at most 36. Second to (n+1)-th lines declare the placements of cars c_1 to c_n. A placement is represented as a string "*cell*:*direction*:*span*". *cell* is the point of the top and left-most cell of the car (between "`A1`" to "`F6`"). direction is either "`vertical`" or "`horizontal`". *span* is an integer between 1 and 6, which indicates the number of cells on which that the car take place. For example, the input file that represents the game board of Figure 1 is as follows:

```
4
A4:horizontal:2
C6:vertical:3
A3:horizontal:3
F3:vertical:3
```

3.3. Program skeleton

The skeleton code can be found at the course github repository[1]. You must follow the overall structure of the program and complete the code, especially by completing missing or broken code marked with "`FIXME`". You must not edit code lines marked with "`NO-FIX`".

The `main` function works as a command-line interface which reads a line from a user and then finds a proper operation for the command and executes it. A structure `car_t` is to represent the current placement of a car, and global array `cars` holds the information of all cars in the play. A 2-dimensional array `cells` is derived from `cars` to indicate which car is placed on each cell. The `move` function is to apply a move to `cars`, and the `update_cells` function updates `cells` according to `cars`, and the `display` function shows the current game board via the standard output.

4. Submission and Evaluation Criteria

You must submit `rushour.c` and `checklist.docx` via Hisnet. Your program code must be built successfully and runs correctly on Repl.it. Your program will be tested with several data files and different command scenarios to check whether it properly works, especially for different invalid command cases.

In addition, you must list all things-to-check (i.e., requirements) and self-check reports in `checklist.docx`. Your listing will be evaluated by checking whether it finds the essential requirements correctly and comprehensively, and whether it demonstrates convincingly that your program fulfills these requirements.

[1] `https://github.com/hongshin/LearningC/tree/master/rushhour`