**Game Description: "Rush Hour"**

Use object-oriented programming to implement this exercise.

**Objective:**

The goal of the game is to move the red car out of a traffic jam through an exit.

The game board consists of a two-dimensional grid.

**Game Board:**

* The game board is represented by a two-dimensional coordinate system.
* The numbering of the coordinates starts from 0. The coordinate (0,0) is the top-left corner of the board.
* Each point (row, col) on the board is identified by a pair of coordinates, with the first indicating the row and the second indicating the column.
* At any given time, only one car can occupy a given coordinate.
* The board you will implement is a square with a side length of 7. The target coordinate is (7,3).

**Red Car Target Coordinate:**

A horizontal car whose right edge is at (6,3) can reach the exit by moving right.

**Example:**

The rightmost column, except for the target coordinate, is for illustration purposes only and does not exist.

**Red Car Target Coordinate:**

**Car Properties:**

The car you will create in the exercise is a one-dimensional object characterized by:

* **Length:** An integer representing the number of cells the car occupies on the grid.
  + In this exercise, the car length is between 2-4.
* **Orientation:** Either vertical (represented by 0) or horizontal (represented by 1).
* **Name:** There are six valid names: Y, B, O, W, G, R, representing yellow, blue, orange, white, green, and red respectively. It is customary for the red car to be the one that needs to reach the target coordinate.
  + For example, if any car, regardless of its color, reaches the target coordinate, it is considered a win.
* **Location:** The car occupies several coordinates on the grid. For convenience, we represent the car's position using the minimum value coordinate among the car's coordinates, which is the top-left corner of the car.

**Example:**

* A horizontal car of length 3 located at [(0,0), (0,1), (0,2)] is represented by the coordinate (0,0).
* A vertical car of length 2 located at [(2,0), (3,0)] is represented by the coordinate (2,0).

**Car Movement:**

* Cars move back and forth on the grid according to their orientation.
* A horizontal car moves left and right only, while a vertical car moves up and down only.
* The orientation of a car is set at initialization and does not change during the game.
* The game is controlled by a single player who moves the cars sequentially on the board.
* On each turn, the player moves a car one step according to its orientation.
  + A horizontal car can move left or right by one step, and a vertical car can move up or down by one step.
  + A car cannot move outside the board or occupy a space already taken by another car.
* The board setup does not address spaces on the grid that are not occupied by cars.

**Note:**

Your task is to determine which class should perform each validation.

**Car Setup:**

You need to create a JSON file that defines the initial board configuration.

**Board Configuration:**

* The JSON file contains the name, length, orientation, and location of each car.
* You need to ensure that the initial configuration allows the game to be solved.
* Your program should read the JSON file using the load\_json function provided, and place all cars on the board in the order given.

**Game Logic:**

* Each turn, the user moves one car by one step. The board will be printed with the changes after each valid move.
* If the input is invalid, print a short error message.
* User input is given as two characters separated by a comma representing the car and direction (e.g., "Y,d" means move the yellow car down).

**Valid Directions:**

* u: up
* d: down
* l: left
* r: right

**Valid Car Names:**

* R, W, G, O, B, Y

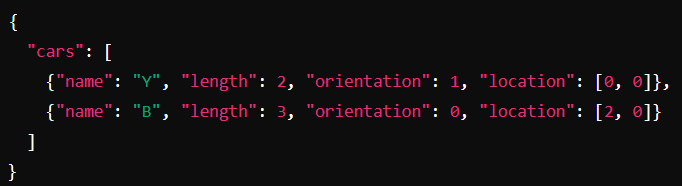
**To exit the game early, the user inputs "!".**

**End of the Game:**

* The game ends when any car reaches the target coordinate (7,3).
* For our board, this is the right edge of a horizontal car located at (6,3).

**Using JSON:**

* You need to create a JSON file to store the initial board configuration. The format is very simple and includes the necessary information for setting up the board.
* Example JSON content:



**Reading JSON:**

1. Import the json library.
2. Open and read the JSON file using the open function.
3. Use the json.load function to convert the file content into a dictionary.

**Note:**

* Coordinates are represented as tuples (row, col).
* Save the JSON file with a .json extension.
* The JSON file can be edited with any text editor.
* We will only check valid JSON files with the structure described above.
* Your program should receive the path to the JSON file as an argument and assume it is valid.

**Object-Oriented Programming:**

* The exercise includes skeleton files for several classes. Each class has method signatures and defines its API. Implement all the methods listed.
  + car.py: Class representing a car
  + board.py: Class representing the game board
  + game.py: Class representing the game
* Each class should have its own responsibilities and not depend on other classes except through the API.

**Additional Files:**

* helper.py: Contains the load\_json(filename) function to read the JSON file.
* car\_config.json: Example configuration file.

**Class Responsibilities:**

1. Each class has its own responsibilities and does not know about the other classes, except through the API.
2. Implement each class independently and ensure it works correctly with the defined API.
3. Invalid input types:

* Invalid commands (e.g., moving in an illegal direction).
* Invalid moves (e.g., moving a horizontal car vertically).

1. Creating a game object does not necessarily start the game. Ensure the game can be continued after initialization.
2. If the configuration file places a car on the target coordinate, the game should end immediately after placing all cars.

**Using the API:**

1. Any method defined in the API must be implemented, even if not used in the exercise.
2. Define new methods as needed, but do not call them from other classes.
3. A method returning an object or list no longer controls what happens to the returned object.
4. Use import only where necessary.

**Special Considerations:**

* Cars do not know about the board.
* The board does not know the game rules.
* The game does not know the board size directly.
* Ensure the str method in the Board class provides a reasonable representation of the board. Use print(board) to display the board.
* Run the game using: python3 game.py <path\_to\_json>
* Validate input as part of the exercise.

**Submission Instructions:**

Submit a zip file named ex9.zip containing:

* car.py
* board.py
* game.py

