Contents

1 Basic Test Results

```
Starting tests...
Thu May 30 11:35:45 IDT 2019
b744ec05d5b01f053150be9ee9cf21177a743ff1  -

Missing required file: README
result_code    missing_file    README    1
Extra file submitted: READMR
result_code    extra_file    READMR    1
**********************************************************************
********************     There is a problem:
********************     Archive does not contain the correct files (or is the wrong format).
**********************************************************************

Archive:  /tmp/bodek.n_0M9O/intro2cs2/ex9/noamoa/presubmission/submission
  inflating: src/board.py
  inflating: src/car.py
  inflating: src/game.py
  inflating: src/READMR


Running presubmit code tests...
12 passed tests out of 12 in test set named 'funcnames'.
result_code    funcnames    12    1
16 passed tests out of 16 in test set named 'carbase'.
result_code    carbase    16    1
6 passed tests out of 6 in test set named 'boardbase'.
result_code    boardbase    6    1
Done running presubmit code tests

Finished running the presubmit tests

Additional notes:

The presubmit tests are very partial. We will not accept appeals if any
of these tests fail.

```

2 READMR

```
noamoa
208533554
noa moalem

I discussed the exercise with:-
============================
= README for ex9: =
============================

==================
= Description: =
==================
We asked to write the game rush hour using opp.

======================
= Special Comments =
======================
```

3 board.py

```
1  ######################################################################
2  # FILE : board.py
3  # WRITER : noa moalem , noamoa , 208533554
4  # EXERCISE : intro2cs ex9 2019
5  # DESCRIPTION: we asked to write the game rush hour using opp
6  ######################################################################
7
8  BOARD_ROW = 7
9  BOARD_COL = 7
10 ORIENTATION = [0, 1]
11 EXIT_ROW = 3
12 EXIT_COL = 7
13 VERTICAL = 0
14 HORIZONTAL = 1
15
16
17 class Board:
18     """
19     Add a class description here.
20     Write briefly about the purpose of the class
21     """
22
23     def __init__(self):
24         """built empty board (an array)"""
25
26         self.cars = {}
27         self.board = []
28         self.create_empty_board()
29
30     def create_empty_board(self):
31         """This function create an empty board, that is array"""
32
33         for i in range(BOARD_ROW):
34             self.board.append([])
35
36         for row in range(BOARD_ROW):
37             if row != EXIT_ROW:
38                 for j in range(BOARD_COL):
39                     self.board[row].append("*")
40             if row == EXIT_ROW:
41                 for j in range(BOARD_COL + 1):
42                     if j == EXIT_COL:
43                         self.board[row].append("E")  # the target cell
44                     elif j != EXIT_COL:
45                         self.board[row].append("*")
46         return self.board
47
48     def __str__(self):
49         """
50         This function is called when a board object is to be printed.
51         :return: A string of the current status of the board ,* is empty place
52         """
53         str = ""
54         for i in range(BOARD_ROW):
55             if i != EXIT_ROW:
56                 for j in range(BOARD_COL):
57                     str += self.board[i][j]
58                     str += " "
59                 str += "\n"
```

```
            if i == EXIT_ROW:
                for j in range(BOARD_COL+1):
                    if j == 7:
                        str += self.board[i][j]
                        str += " "
                    elif j != 7:
                        str += self.board[i][j]
                        str += " "
                str += "\n"

        return str

    def cell_list(self):
        """ This function returns the coordinates of cells in this board
        :return: list of coordinates
        """
        list_of_coordinates = []
        for i in range(BOARD_ROW):
            for j in range(BOARD_COL):
                list_of_coordinates.append(tuple([i, j]))
                if i == 3 and j == 6:
                    list_of_coordinates.append(tuple([3, 7]))
        return list_of_coordinates

    def possible_moves(self):
        """ This function returns the legal moves of all cars in this board
        :return: list of tuples of the form (name,movekey,description)
            representing legal moves"""
        res_list = []
        for car in self.cars:
            if car[2] == 0:
                res_list.append(tuple([car, "u", "cause the car vertical"]))
                res_list.append(tuple([car, "d", "cause the car vertical"]))

            res_list.append(tuple([car, "r", "cause the car horizontal"]))
            res_list.append(tuple([car, "l", "cause the car horizontal"]))

        return res_list

    def target_location(self):
        """
        This function returns the coordinates of the location which is to be filled for victory.
        :return: (row,col) of goal location
        """
        return (EXIT_ROW, EXIT_COL)

    def cell_content(self, coordinate):
        """
        Checks if the given coordinates are empty.
        :param coordinate: tuple of (row,col) of the coordinate to check
        :return: The name if the car in coordinate, None if empty
        """

        if coordinate[0] < BOARD_ROW and coordinate[1] < BOARD_COL+1:
            if self.board[coordinate[0]][coordinate[1]] == "*" or \
                    self.board[coordinate[0]][coordinate[1]] == "E":
                return None  # the cell is empty
        return self.board[coordinate[0]][coordinate[1]]

    def check_location_empty(self, car, location):
        """This function check if the location is empty so we could put there
        the car"""

        check_location_empty = []  # will be filled with True/False
        if car.orientation == VERTICAL:
            for i in range(car.length):
                if location[0]+i < BOARD_ROW:
                    if self.cell_content(tuple([location[0] + i,location[1]]))\
```

```
                            is None:  # this cell is empty
                        check_location_empty.append(True)
                    else:
                        check_location_empty.append(False)
        if car.orientation == HORIZONTAL:
            for i in range(car.length):
                if location[0] + i < BOARD_COL:
                    if self.cell_content(tuple([location[0],location[1] + i]))\
                            is None:  # this cell is empty
                        check_location_empty.append(True)
                    else:
                        check_location_empty.append(False)
        # the next line check if all the cells are empty by checking if
        # check_location_empty is filled with True only
        if len(check_location_empty)==car.length and all(check_location_empty):
            return True
        return False

    def add_car_helper(self, car):
        """This function put the letter of the car name on the board"""

        if car.orientation == VERTICAL:
            for i in range(car.length):
                self.board[car.location[0] + i][car.location[1]] = car.name
        if car.orientation == HORIZONTAL:
            for i in range(car.length):
                self.board[car.location[0]][car.location[1] + i] = car.name

    def add_car(self, car):
        """
        Adds a car to the game.
        :param car: car object of car to add
        :return: True upon success. False if failed
        """
        if car.location[0] < 0 or car.location[1] < 0:
            return False
        if car.name in self.cars:
            return False
        if car.location[0] < BOARD_ROW and car.location[1] < BOARD_COL:
            if self.check_location_empty(car, car.location):
                self.cars[car.name] = car  # adding car to the dictionary cars
                self.add_car_helper(car)  # adding car to thr board
                return True
        return False

    def move_car(self, name, movekey):
        """
        moves car one step in given direction.
        :param name: name of the car to move
        :param movekey: Key of move in car to activate
        :return: True upon success, False otherwise
        """
        if name in self.cars:
            empty_place_needed = self.cars[name].movement_requirements(movekey)
            if not empty_place_needed:  # the car can't move in this move key
                return False
            if 0 <= empty_place_needed[0][0] < BOARD_ROW \
                    and self.cell_content(empty_place_needed[0]) is None:
                old_location = self.cars[name].car_coordinates()
                if empty_place_needed[0][0] == EXIT_ROW:
                    if 0 <= empty_place_needed[0][1] < BOARD_COL+1:
                        if self.cars[name].move(movekey):  # move in car
                            # the next lines delete the old car from the board
                            for i in old_location:
                                self.board[list(i)[0]][list(i)[1]] = "*"
                            # the next line adding the car in the new location
                            self.add_car_helper(self.cars[name])
                            return True
```

```
                        return False
                    return False

                elif 0 <= empty_place_needed[0][1] < BOARD_COL:
                    if self.cars[name].move(movekey):  # move in car
                        # the next lines delete the old car from the board
                        for i in old_location:
                            self.board[list(i)[0]][list(i)[1]] = "*"
                        # the next line adding the car in the new location
                        self.add_car_helper(self.cars[name])
                        return True
                return False
            return False
        return False
```

4 car.py

```
1  ######################################################################
2  # FILE : car.py
3  # WRITER : noa moalem , noamoa , 208533554
4  # EXERCISE : intro2cs ex9 2019
5  # DESCRIPTION: we asked to write the game rush hour using opp
6  ######################################################################
7
8  NAMES = ["Y", "B", "O", "W", "G", "R"]
9  ORIENTATION = [0, 1]
10 HORIZONTAL = 1
11 VERTICAL = 0
12
13
14 class Car:
15     """
16     This class create car object ,and has function that help get information on
17      the car ,move the car and more"""
18
19     def __init__(self, name, length, location, orientation):
20         """
21         A constructor for a Car object
22         :param name: A string representing the car's name
23         :param length: A positive int representing the car's length.
24         :param location: A tuple representing the car's head (row, col) location
25         :param orientation: One of either 0 (VERTICAL) or 1 (HORIZONTAL)
26         """
27
28         self.name = name
29         self.length = length
30         self.orientation = orientation
31         self.location = location
32
33     def car_coordinates(self):
34         """
35         :return: A list of coordinates the car is in
36         """
37         list_of_coordinates = []
38         car_location = self.location
39         car_length = self.length
40         car_orientation = self.orientation
41
42         if car_orientation == VERTICAL:
43             for i in range(car_length):
44                 locat_to_add = (car_location[0]+i, car_location[1])
45                 list_of_coordinates.append(locat_to_add)
46         if car_orientation == HORIZONTAL:
47             for i in range(car_length):
48                 locat_to_add = (car_location[0], car_location[1]+i)
49                 list_of_coordinates.append(locat_to_add)
50         return list_of_coordinates
51
52     def possible_moves(self):
53         """
54         :return: A dictionary of strings describing possible movements permitted by this car.
55         """
56         if self.orientation == VERTICAL:
57             possible_direction = {"u": "cause the car vertical ",
58                                   "d": "cause the car vertical "}
59         else:
```

```
            possible_direction = {"r": "cause the car horizontal ",
                                  "l": "cause the car horizontal "}

        return possible_direction

    def movement_requirements(self, movekey):
        """
        :param movekey: A string representing the key of the required move.
        :return: A list of cell locations which must be empty in order for this move to be legal.
        """

        possible_move = self.possible_moves()
        if movekey in possible_move:
            if self.orientation == VERTICAL:
                if movekey == "d":
                    last_cell_location = self.car_coordinates()[-1]
                    empty_cell_neede = [tuple([last_cell_location[0] + 1,
                                               last_cell_location[1]])]
                if movekey == "u":
                    last_cell_location = self.car_coordinates()[0]
                    empty_cell_neede = [tuple([last_cell_location[0] - 1,
                                               last_cell_location[1]])]
                return empty_cell_neede

            if self.orientation == HORIZONTAL:
                if movekey == "r":
                    last_cell_location = self.car_coordinates()[-1]
                    empty_cell_neede = [tuple([last_cell_location[0],
                                               last_cell_location[1] + 1])]
                if movekey == "l":
                    last_cell_location = self.car_coordinates()[0]
                    empty_cell_neede = [tuple([last_cell_location[0],
                                               last_cell_location[1] - 1])]
                return empty_cell_neede

        return False  # because direction is not valid

    def move(self, movekey):
        """
        :param movekey: A string representing the key of the required move.
        :return: True upon success, False otherwise
        """

        possible_move = self.possible_moves()
        if movekey in possible_move:
            old_location = self.location

            if self.orientation == VERTICAL:
                if movekey == "d":
                    self.location = [old_location[0] + 1,
                                     old_location[1]]
                if movekey == "u":
                    self.location = [old_location[0] - 1,
                                     old_location[1]]
            if self.orientation == HORIZONTAL:
                if movekey == "r":

                    self.location = [old_location[0],
                                     old_location[1] + 1]
                if movekey == "l":
                    self.location = [old_location[0],
                                     old_location[1] - 1]
            return True
        return False

    def get_name(self):
        """
        :return: The name of this car.
```

```
128             """
129             return self.name
130
```

5 game.py

```
1  ######################################################################
2  # FILE : game.py
3  # WRITER : noa moalem , noamoa , 208533554
4  # EXERCISE : intro2cs ex9 2019
5  # DESCRIPTION: we asked to write the game rush hour using opp
6  ######################################################################
7
8  NAMES = ["Y", "B", "O", "W", "G", "R"]
9  DIRECTION = ["d", "u", "l", "r"]
10 possible_car_length = [2, 3, 4]
11 ORIENTATION = [0, 1]
12
13 from helper import *
14 from board import *
15 from car import *
16 import sys
17
18
19 def check_input(userinput):
20     if len(userinput) != 3 or "," not in userinput:
21         print("its not the right format")
22         return False
23     if userinput[0] not in NAMES:
24         print("there is no such a car")
25         return False
26     if userinput[2] not in DIRECTION:
27         print("invalid direction")
28         return False
29     return True
30
31
32 class Game:
33     """ This class create a game and drive the game """
34
35     def __init__(self, board):
36         """
37         Initialize a new Game object.
38         :param board: An object of type board
39         """
40         self.board = board
41
42     def __single_turn(self):
43         """The function runs one round of the game :
44             1. Get user's input of: what color car to move, and what
45                 direction to move it.
46             2. Check if the input is valid.
47             3. Try moving car according to user's input"""
48
49         print(self.board.__str__())  # display the board at first
50         user_input = input("please enter car and direction: ")
51         while not check_input(user_input):  # check the input valid
52             user_input = input("please enter car and direction: ")
53         if self.board.move_car(user_input[0], user_input[2]):#try move tha car
54             print("the car moved")
55             return
56         else:
57             print("car didn't move")  # didn't move the car
58
59     def play(self):
```

```
60          """
61          The main driver of the Game. Manages the game until completion.
62          :return: None
63          """
64
65          while self.board.board[self.board.target_location()[0]]\
66                  [self.board.target_location()[1]] == "E":  # the car didn't
67                                                             # arrive the exit
68              self.__single_turn()
69          print("you won!")
70
71
72  if __name__ == "__main__":
73      args = sys.argv
74      board = Board()
75      cars = load_json(args[1])
76      for car in cars:
77          if car in NAMES and cars[car][0] in possible_car_length and cars[car][2] \
78                  in ORIENTATION:
79              car = Car(car, cars[car][0], cars[car][1], cars[car][2])
80              board.add_car(car)
81      game = Game(board)
82      game.play()
```